PROCEEDINGS

13th INTERNATIONAL MODELICA CONFERENCE

March 4–6, 2019
Ostbayerische Technische
Hochschule Regensburg, Germany

Editor: Prof. Anton Haumer

OTH OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

MODELICA

Proceedings of the 13th International Modelica Conference
Regensburg, Germany, March 4 – 6, 2019

Editor:
Prof. Anton Haumer

Published by:
Modelica Association and Linköping University Electronic Press

Series:	Linköping Electronic Conference Proceedings, No. 157
ISBN:	978-91-7685-122-7
ISSN:	1650-3686
eISSN:	1650-3740
DOI:	10.3384/ecp19157
URL:	http://www.ep.liu.se/ecp/contents.asp?issue=157

Organized by:

OTH Regensburg	Modelica Association
Seybothstr. 2	c/o PELAB, IDA Linköpings Universitet
D-93053 Regensburg	S-58183 Linköping
Germany	Sweden

Conference location:
OTH Regensburg
Seybothstr. 2
D-93053 Regensburg
Germany

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 and to the Functional Mockup Interface.

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

Highlights of the Conference:

- 76 oral presentations and 13 poster presentations, 4 libraries for the Modelica Library Award
- 2 Keynotes
- 7 Tutorials and 2 Industrial User Presentations Sessions
- 14 Vendor Sessions and 17 Sponsors & Exhibitors

Welcome

I warmly welcome you to Regensburg, a city with history going back to Roman times, and to OTH the Technical University of Applied Sciences Regensburg.

Starting with this conference, you will notice some changes:

First, we are going to organize the International Modelica Conference every two years in spring. In the years between International Modelica Conferences, Modelica Conferences are organized on other continents with country specific focus.

Although in 2018 there have been two very successful conferences in Japan and in the United States, we received 101 submissions from authors all over the world which have been thoroughly reviewed.

Second, additional to the tutorials and vendor presentations on the first day of the conference, we are going to have Industrial User Presentations related to the Modelica Association Projects. These presentations are not included in the proceedings, but they should provide a nucleus for discussions and broadening the users groups.

I want to thank the members of the Program Committee for their work during the review process, as well as the members of the Organizing Committee – without their support this conference wouldn't have been a success.

Prof. Anton Haumer
OTH Regensburg
Conference Chair

Modelica News

In the name of the Modelica Association that is co-organizing this event, I also would like to welcome you in

Regensburg. It is now already the 13th conference on Modelica, the Functional Mockup Interface and related technology. Since the number of projects and standards of the Modelica Association is growing, we would like to give you an overview about the current status in the traditional "Modelica Association News" section on Tuesday morning: All the Modelica Association Project leaders will give a short overview about their project and about their future plans.

Prof. Dr. Martin Otter
DRL, Wessling, Germany
Chair of Modelica Association

Keynote: Modelica and virtual education

Dr. Christian Kral
TGM, Vienna, Austria

Good education of engineering students requires theoretical knoweldge and lots of calculation experience to better understand theory and applications. Laboratory courses are offered to better relate theory and practical understanding. Simulations even more improve the linking of theory and practice, as systemic thinking is supported. Students learn to understand the interaction of simple models and more advanced systems.

In the keynote speech two virtual education scenarios in engineering will be presented: First, a workflow of creating and evaluating calculation and simulation examples is proposed. The workflow is based on Modelica and the online tool Letto. Second, virtual lab experiments of electric machines and drives are shown. In the virtual lab Modelica variables are controlled and viszualized by Labview. The presented approaches are possible steps in the direction of virtual education to improve and strengthen the students' expertise and knowledge and with the particular intention to motivate students.

Bio: Christian Kral received the diploma and doctoral degrees from the Vienna University of Technology, Vienna, Austria, in 1997 and 1999, respectively. From 1997 to 2000, he was a Scientific Assistant in the Institute of Electrical Drives and Machines, Vienna University of Technology. Since 2001, he has been with the AIT Austrian Institute of Technology GmbH (the former Arsenal Research) in Vienna. From January 2002 until April 2003, he was a Visiting Professor at the Georgia Institute of Technology, Atlanta. Dr. Kral is teaching electric machines and drives at the higher college of engineering »TGM« in Vienna and the university of applied research, »Technikum Wien« since 2013. His research interests include the modeling and simulation of electrical systems, machines and drives. He is a member of the Austrian Electrotechnical Association (OVE) and the Modelica Association. Dr. Kral published over 150 scientific papers and one book on Modelica and the object oriented modeling of electric machines.

Keynote: Simulation Guided Design for New Automotive Applications

Dr. Gerd Rösel
Continental, Regensburg, Germany

The Automotive Industry has to cope with disruptive technology and business changes within the next decade. Connected vehicles become reality and drive the development to automated driving. New mobility solutions will have to answer shared economy demands. The regulatory requirement on significant reduction of CO2- and pollutant emission leads to fast changing parallel development of additional propulsion systems in the same period. Consequently, the variety of solutions within a vehicle will have to serve a furthermore increasing complexity from embedded-systems to system-of-systems to cyber-physical-systems.

Simulation guided design is the key to handle such complexity in all areas of application for an automotive supplier to keep quality, time to market and costs under control. The speech covers the main directions of disruptive technology changes and examples of dedicated solutions. There will be examples given which cover virtual function development for embedded systems as well as solutions for predictive maintenance and connected energy management as system-of-systems. The focus will be to point out the necessity to design and optimize such systems by simulation.

Bio: Dr. Gerd Rösel is heading the departments Advanced System Engineering for Engine Systems (since 2015) as well as Hybrid Electric Vehicle Business Unit (since 2018) for Continental Powertrain. The application and further development of simulation methodologies is a significant building block in these responsibilities. The variety in simulation technology covers propulsion system simulation as well as specialized simulation in areas like electric machines, mixture formation and NVH.

From 1996 until 2015 he has been responsible in different positions for Gasoline- and Diesel-System-Development for serial and advanced applications. From 1992 to 1997 he was a research associate at Technical University of Dresden and finished with the graduation of Dr.-Ing. in 1997. The Diploma degree in electrical engineering from Technical University of Dresden was achieved in 1992.

ORGANIZING COMMITTEES

Conference Chair

Prof. Anton Haumer, OTH Regensburg, Germany

Conference Board

Dr. Hilding Elmqvist, Mogram, Sweden
Prof. Peter Fritzson, Linköping University, Sweden
Prof. Martin Otter, DLR, Germany
Dr. Michael Tiller, Xogeny, USA

Program Committee

Johan Åkesson, Modelon AB and Department of Automatic Control, Lund University, Sweden
Markus Andres, Dassault Systemes Deutschtland GmbH, Germany
Maximilian Apfelbeck, Max Streicher, Germany
Bernhard Bachmann, Fachhochschule Bielefeld, Germany
John Baras, University of Maryland, USA
John Batteh, Modelon, Inc., USA
Albert Benveniste, INRIA, France
Christian Bertsch, Robert Bosch GmbH, Germany
Volker Beuter, VI-grade GmbH, Germany
Thomas Beutlich, ESI ITI GmbH, Germany
Marco Bonvini, Lawrence Berkeley National Laboratory, USA
Scott Bortoff, Mitsubishi Electric Research Laboratories, Japan
Timothy Bourke, INRIA, France
Daniel Bouskela, EDF, France
David Broman, KTH Royal Institute of Technology, Sweden
Dan Burns, MERL, USA
Felix Bünning, Empa/ETH Zürich, Switzerland
Francesco Casella, Politecnico di Milano, Italy
Massimo Ceraolo, University of Pisa, Italy
Yan Chen, Pacific Northwest National Lab,
Massimo Cimmino, McGill University, Canada
Christoph Clauss, Fraunhofer IIS EAS Dresden, Germany
Johan de Kleer, Palo Alto Research Center, Inc., USA
Mike Dempsey, Claytex, GreatBritain
Hilding Elmqvist, Mogram, Sweden
Olaf Enge-Rosenblatt, Fraunhofer, Germany
Gianni Ferretti, Politecnico di Milano, Italy
Peter Fritzson, Linköping University, Sweden
Leo Gall, LTX Simulation GmbH, Germany
Anton Haumer, OTH Regensburg, Germany
Dan Henriksson, Dassault Systemes, Sweden
Yutaka Hirano, Toyota Motor Corporation, Japan
Jianjun Hu, Lawrence Berkeley National Laboratory, USA
Christoph Höger, TU Berlin, Germany
Bengt Jacobson, Chalmers University of Technology, Sweden
Filip Jorissen, Katholieke Universiteit Leuven, Netherlands
Jochen Koehler, ZF Friedrichshafen AG, Germany
Jiri Kofranek, Charles Univesrity, CzechRepublic
Christian Kral, Electric Machines, Drives and Systems, Austria
Christopher Laughman, MERL, USa

Organizing Committee

CONTENTS

INDEX OF AUTHORS

SESSION 1A: BUILDINGS 1

A virtual test-bed for building Model Predictive Control developments

Raymond Sterling[1], Jesús Febres[2], Andrea Costa[3], Adeleh Mohammadi[1], Rafael E. Carrillo[4], Baptiste Schubnel[4], Yves Stauffer[4], Pietro De Cinque[3], Krzysztof Klobut[5], Marcus M. Keane[1]

[1]Department of Civil Engineering, National University of Ireland, Galway, Ireland
{raymond.sterling,adeleh.mohammadi,marcus.keane}@nuigalway.ie
[2]Fundación IK4 Tekniker, Spain, jesus.febres@tekniker.es
[3]R2M Solution SRL, Italy, {andrea.costa2, pietro.decinque}@r2msolution.com
[4]CSEM SA, PV-center, Neuchâtel, Switzerland
{rafael.carrillo,baptiste.schubnel,yves.stauffer}@csem.ch
[5] VTT Technical Research Centre of Finland, Espoo 020400, Finland Krzysztof.Klobut@vtt.fi

Abstract

This paper presents the result of the work performed to develop a virtual test-bed for development and testing of Model Predictive Controllers within the district cooling networks field. These controllers are used for its application in improving the cooling energy efficiency in the network's building. The article explains the use of the different tools to develop and simulate the models with an emphasis on the advantages and challenges of co-simulation and model exchange using the Functional Mockup Interface.

Keywords: District Cooling, Modelling, FMI, Modelica, Model predictive control

1 Introduction

INDIGO[1] is a Horizon 2020 EU-funded project carried out by six partners from across Europe that aims to realise more efficient and economic planning, control and management of existing District Cooling (DC) networks. This will be achieved through two specific objectives. The first one is to widen the use of DC systems and motivate the competitiveness of European DC market by the development open-source tools for planning and modelling DC systems (del Hoyo Arce *et al.*, 2018). The second objective is to reduce primary energy consumption via improved DC system management strategies aimed at system efficiency maximisation and cost minimisation.

In this paper we present the results of the work performed to improve the energy consumption of the DC systems across several tasks of the project. This includes modelling and simulation of various buildings and the development and implementation of Model Predictive Controls (MPC) to reduce energy use in buildings.

Modelling and simulation within this paper is presented for the Building models. The geometry, materials, weather, air infiltration and internal gains of the models are developed in EnergyPlus and the model of the energy systems, focusing on the air distribution system while air handling units are built in Modelica.

The aim of the modelling was two-fold. To provide an accurate and validated test-bed for testing the behaviour of the MPC and, at the same time, generate the synthetic data used for the initial development of said controllers.

Model integration across different platforms is performed via Functional Mock-up Interfaces and this article presents the full workflow on the implementation from initial building model development to the generation of results from the MPC.

2 Building Models – EnergyPlus

In the INDIGO project, all the geometrical models of the buildings are created considering the external dimensions. This approach influences the way in which the linear transmittances of the thermal bridges are calculated. To create the model, the following information has been collected:

- Geometry of the building;
- Geometry and position of the shading objects (e.g. other buildings or trees) located around the modelled buildings;
- Distribution of the mechanical ventilation and the relative control;
- Position and properties of opaque and transparent elements (walls, roofs, windows, floors, internal partitions);
- Electrical consumption for the different buildings and for the main equipment that is installed in them, to estimate the internal gains.

For the development of the models related to the buildings, DesignBuilder v4.7.0.027 and EnergyPlus[2] V8.6 were used (Figure 1). It manages input files in .idf

[1] www.indigo-project.eu

[2] https://energyplus.net/

format, which can be edited in the IDF Editor (free available online) or in a text editor. The IDF file contains both the input data that the building model acquires from the HVAC system model, which is developed in Modelica, and the output data that the building model transfers to the HVAC system model. The weather data are included in an .epw file (*EnergyPlus weather file*). The information included in the .idf file and that one included in the .epw file are combined in a file that is readable by Modelica, specifically in a .fmu file. The FMU is exported through a Python script (Nouidui, Wetter and Zuo, 2014).

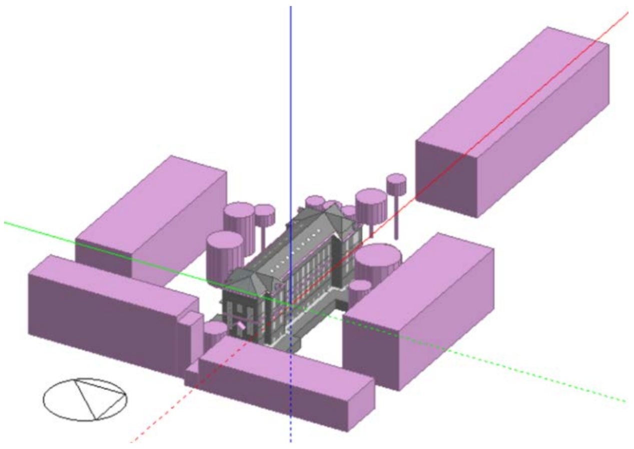


Figure 1. Aztarain model (DesignBuilder)

In the building model, there are thermal zones in which the internal air conditions are considered uniform. The thermal zones were created based on the air conditioning system and of its control logics, to allow an accurate modelling of the HVAC systems, as it is needed by the INDIGO project. As general rule, the zones whose thermal conditions are controlled by the same system (AHU) and based on the same sensors are modelled as part of the same thermal zone.

2.1 Thermal Zones and internal gains

For each building, the zones were created considering the following criteria t:

- Distinction between conditioned, not conditioned and specially conditioned zones (e.g. zones with special conditioning requirements);
- If two parts of the same building are served by different AHUs the two parts will be modelled separately;
- The creation of the zones considers the location of the terminal units, to define if the conditions of some rooms are controlled by a post-heating or by a post-cooling coil.
- The creation of the zones also considers the location of the temperature sensors within the air distribution scheme, to model as close as possible to the reality the control logic and the temperature of the sensors on which the control logic is based.
    - o Rooms whose internal conditions are measured by a specific sensor (inside the room or in the return duct) are modelled separately.
    - o The location of the sensor is important because it defines the conditions that will be used by the control system and therefore affects the goal of the project.
- Internal gains have been modelled based on working schedules and a survey which was used to determine occupancy levels and equipment inside the zones. The occupation of the different zones was modelled considering the number of seats or beds represented in the architectural drawings.

2.2 Weather data

The weather data regarding dry air temperature (°C) and relative humidity (%) are taken on site while all the other data (solar radiation, wind velocity, wind direction, pressure) are taken from the weather station “C039 - Deusto” of the Basque agency of meteorology (“Agencia vasca de meteorología”)[3] . The weather station is in the Bilbao city, 2,5 km away from the demo site presented in this paper.

The global radiance on a horizontal surface expressed in [W/m^2] is information included in the weather file. A method provided by Reindl, D.T. et. al (1990) was used to estimate the diffuse radiation and the direct radiation from the global one. The method was validated for 5 localities in America and in Europe having very different climates (latitudes from 28.4°N to 59.56°N) (Reindl, Beckman and Duffie, 1990).

The sun position is evaluated based on the geographical position of the building.

2.3 Preparation for interfacing with Modelica

To establish the communication between EnergyPlus and Modelica the use of the EnergyPlus object “ExternalInterface” is necessary. This object activates the external interface of EnergyPlus.

Currently, the only valid entries are PtolemyServer, FunctionalMockupUnitImport, and FunctionalMockupUnitExport.

2.3.1 Receiving data from Modelica

For the INDIGO project, the option “FunctionalMockupUnitExport” was selected because the EnergyPlus file is exported as a FMU for co-simulation. The data that Modelica communicates to EnergyPlus are:

[3] http://www.euskalmet.euskadi.net/s07-5853x/es/meteorologia/estacion.apl?e=5&campo=C039

- *Sensible load [Qs]* (W) due to the air supplied by the mechanical ventilation (for every zone of the building)
- *Latent load [Ql]* (W) due to the air supplied by the mechanical ventilation (for every zone of the building)

EnergyPlus considers those loads in the same way as the internal thermal gains.

The use of heat flows instead of typical state variables (temperature, air mass flow rate, relative humidity etc.) in the data exchange from Modelica to the building FMU is motivated by modelling simplifications as exchanging state variables would significantly increase the computational effort without providing advantages over the proposed procedure.

2.3.2 Sending Data to Modelica

The data that EnergyPlus communicates to Modelica are:

- Temperature [T]
  - Site Outdoor Air Dry-Bulb Temperature (°C)
  - Zone Mean Air Temperature (°C) (for every zone of the building)
- Relative humidity [RH]
  - Site Outdoor Air Relative Humidity (%)
  - Zone Mean Air Relative Humidity (%) (for every zone of the building)
- Humidity ratio [X]
  - Site Outdoor Air Humidity Ratio (kg_{Water}/kg_{DryAir})
  - Zone Mean Air Humidity Ratio (kg_{Water}/kg_{DryAir}) (for every zone of the building)

To calculate the heat flows between the HVAC system and the building, Modelica requires knowledge of the temperature and humidity conditions of the zones. Therefore, these are the variables selected to be exchanged from the FMU to Modelica.

3 HVAC Models – Modelica

On the demo site for the INDIGO project, six types of air handling units were identified. However, for reasons of space in this paper, this work focuses on one type which is the most common and the one described in the case study, for further information please refer to (Sterling *et al.*, 2017).

Modelica models for HVAC systems use components based on the Modelica.Fluid library to replicate the schematic of the units. In INDIGO, all AHU will have fresh (port_F) and supply (port_S) port connections. For those units with return air, return (port_R) and exhaust (port_E) port connection are added.

All units will output the heat flow of each active component (e.g. heating coils and cooling coils).

Nominal design conditions have been imposed for the cooling coil models since no information about the input conditions on the water side of the cooling coils is being gathered by the BMS. Such conditions correspond with constant input water temperature and constant maximum mass flow rate achieved when valve is 100% open.

3.1 AHU Model

For this research work, a full-sized air handling unit type is demonstrated. It is composed of:

- Heat recovery (HR): two heat exchangers interconnected via a water circuit;
- Cooling Coil (CC);
- Heating Coil (HC);
- Fans;
- Humidifier (H);

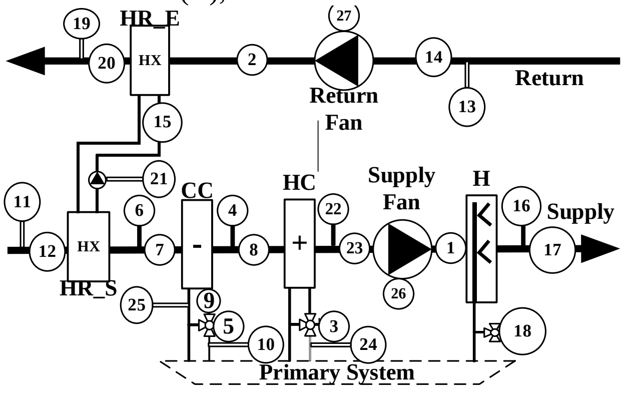


Figure 2. Schematic of the AHU under study

Table 1. AHU Model Components and Variables. Type: M: Measurement, I: Input

Component	*Variable*	*Type*	*#*
Fan Supply	air mass flow rate	M	1
Fan Return	air mass flow rate	M	2
Heating Coil	Valve position	I	3
Cooling Coil	Air output T	M	4
Cooling Coil	Valve position	I	5
Cooling Coil	Air input T	M	6
Cooling Coil	Air input RH	M	7
Cooling Coil	Air output RH	M	8
Cooling Coil	Water mass flow rate	M	9
Cooling Coil	Water input T	M	10
HR Supply	Air input T	M	11
HR Supply	Air input RH	M	12
HR Exhaust	air input T	M	13
HR Exhaust	Air input RH	M	14
HR Exhaust	water mass flow rate	M	15
Humidifier	Air output T	M	16
Humidifier	Air output RH	M	17
Humidifier	Valve position	I	18

HR Exhaust	Air output T	M	19
HR Exhaust	Air output RH	M	20
HR water	Pump pressure	M	21
Heating Coil	Air output T	M	22
Heating Coil	Air output RH	M	23
Heating Coil	Water input T	M	24
Cooling Coil	Water output T	M	25
Fan Supply	Pump pressure	M	26
Fan Return	Pump pressure	M	27

3.2 AHU Controllers

The AHU has an associated control system (named AHU_controller) emulating the behaviour of the real system as closely as possible with the available data (e.g. O&M manuals from the demo site).

The AHU_controller operates in a mode-switching hybrid system, i.e., it is a system that can operate in multiple modes, and can switch between these modes either through continuous- or discrete-valued signals. The AHU can operate in 2 nominal modes for temperature control: (1) when the controlled temperature is above its set-point plus dead-band (T_ASP) and (2) when the controlled temperature is below its set-point minus dead-band (T_BSP):

In the case of T_ASP, the AHU_controller shall:

- Modulate the opening signal valve of the Heating Coil towards the fully closed position.
- Modulate the opening signal valve of the Cooling Coil towards the fully opened position.

In the case of T_BSP, the AHU_controller shall:

- Modulate the opening signal valve of the Cooling Coil towards the fully closed position.
- Modulate the opening signal valve of the Heating Coil towards the fully opened position

All modulations are performed via PID control.

In this controller humidity control operates independently from the heating/cooling operation. For humidity control, the AHU can also operate in two nominal modes: (1) when the controlled relative humidity is above its set-point plus dead-band (RH_ASP) and (2) when the controlled relative humidity is below its set-point minus dead-band (RH_BSP).

In the case of RH_ASP, the AHU_controller shall:

- Modulate the opening signal valve of the Cooling Coil towards the fully opened position.
- Modulate the opening signal valve of the Humidifier towards the fully closed position.

In the case of RH_BSP, the AHU_controller shall:

- Modulate the opening signal valve of the Cooling Coil towards the fully closed position.
- Modulate the opening signal valve of the Humidifier Coil towards the fully opened position

All modulations are performed via PID control.

In this controller, heat recovery control operates independently from the heating/cooling operation. Heat recovery operates in on/off mode as follows:

- If (Cooling Coil valve > 0 and dH > 0) or (Heating Coil valve > 0 and dH < 0) then 1.0 else 0.0.

According to maintenance personnel from the demo site, fans operate at fixed mass flow rate 100%. Hence, in this controller, fan output is always true.

4 Whole Building Model

4.1 FMU Interfacing

EnergyPlus is a well-established, whole building energy simulation tool that considers a broad range of different characteristics of the buildings. It is an optimal tool to simulate the long-term (days, months and years) energy performance of the buildings. However, the implementation of the HVAC systems within EnergyPlus does not account for dynamics of diverse elements of such systems (heat exchangers, ducts, boilers, etc.) making this tool poorly accurate for short-term (minutes and hours) simulations. To overcome this issue, we decided to integrate an EnergyPlus model of the buildings (geometry, materials, weather, internal gains) with HVAC models developed in Modelica via the Functional Mock-up Interface[4]. Figure 3 shows the data exchange, at each time-step, between HVAC model in Modelica and each zone in the EnergyPlus building model.

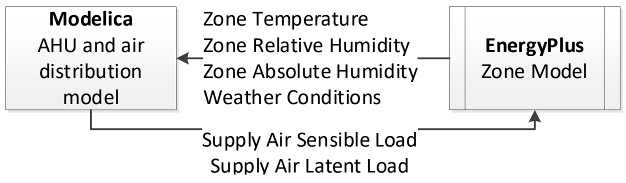


Figure 3. Modelica/EnergyPlus data exchange diagram.

4.2 INDIGO demo site model

To demonstrate the approach INDIGO has taken towards developing the models, part of a building that is supplied by a single air handling unit has been selected. This zone is called “Aislamiento” since it is the section where isolation rooms for immunodepressed patients are hospitalised. Hence, the Aislamiento zones are conditioned by a specific AHU because in those rooms the requested conditions are different. This AHU, which structure is identical to the one in Figure 2, supplied two zones that are kept at a pressure positive state. Figure 4 shows the main blocks of the model with corresponding variable exchange as built in Modelica.

[4] http://fmi-standard.org/

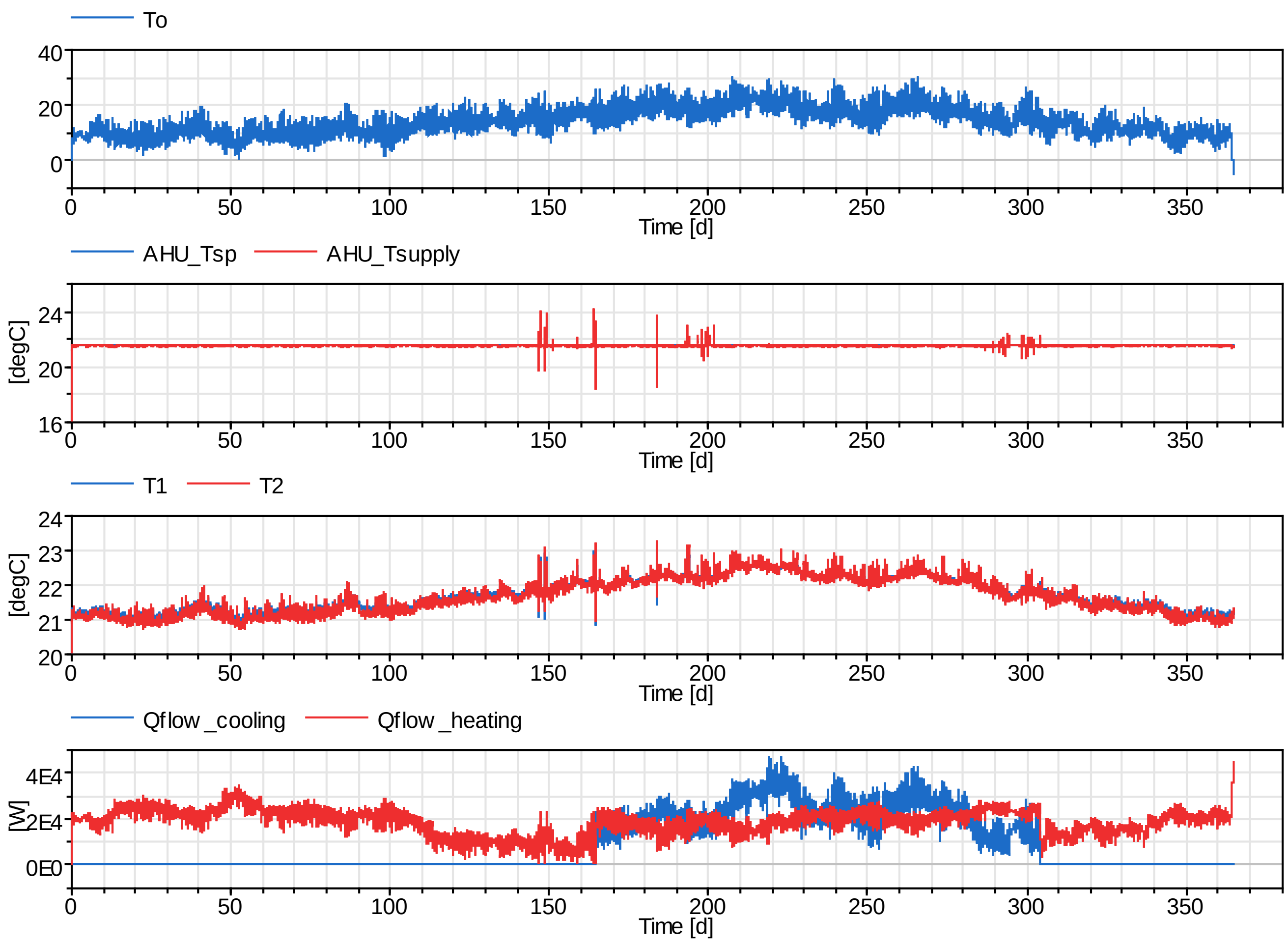


Figure 4. Results from 1-year simulation

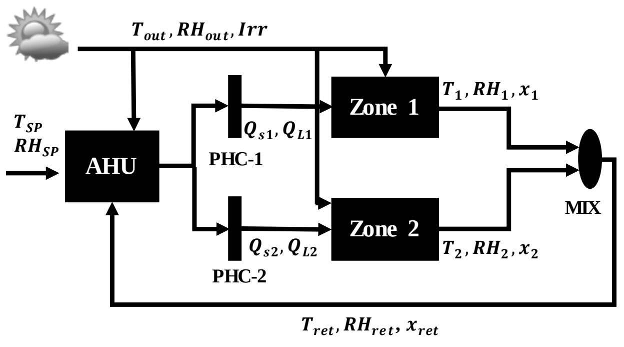


Figure 5. AHU Modelica Schematic

In Figure 4, T stands for Temperature, RH for relative humidity, x for absolute humidity, Qs for sensible heat, Ql for latent heat, PHC for post-heating coil and SP for setpoint. As mentioned, the model takes as inputs weather data and set-points and exchanges with the EnergyPlus FMU sensible and latent loads calculated also using the FMU Zone's information for indoor air conditions.

To use this model in the MPC developments, the whole building (AHU in Modelica and physical model in EnergyPlus-FMU) model was packaged in a FMU as shown in Figure 6.

From the whole building FMU (Figure 6), the model only needs as inputs weather data and outputs all the necessary variables to produce data for developing and training the MPC controllers. This includes not only the environmental conditions of the air in different point of the energy path but also the energy consumption of relevant elements such as coils.

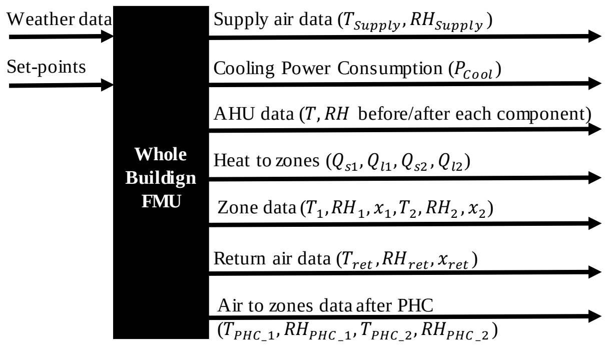

Weather data
Set-points
Whole Buildign FMU
Supply air data (T_{Supply}, RH_{Supply})
Cooling Power Consumption (P_{Cool})
AHU data (T, RH before/after each component)
Heat to zones ($Q_{s1}, Q_{l1}, Q_{s2}, Q_{l2}$)
Zone data ($T_1, RH_1, x_1, T_2, RH_2, x_2$)
Return air data ($T_{ret}, RH_{ret}, x_{ret}$)
Air to zones data after PHC ($T_{PHC\_1}, RH_{PHC\_1}, T_{PHC\_2}, RH_{PHC\_2}$)

Figure 6. Whole Building FMU Schematic

4.3 Model simulation

Results from simulating the model for 1-year are presented in Figure 5. The purpose of performing the simulation was to validate the suitability of the model for use in MPC, to validate the values provided in the results were coherent, to check correct the integration of the different simulation tools via FMU and to check the proper implementation of the low-level controls. The model behaves as expected albeit some spikes appear that show in the middle of the year simulation which are caused by the mode-changing (e.g. state machine) implementation of the control system. However, such spikes do not affect the overall behaviour and result from the model.

5 Interfacing for MPC and results

This section provides details on the development of the Model Predictive Controller for the Air Handling Unit based on the FMU described in the previous section.

5.1 Interfacing scheme

Figure 7 shows the general interfacing scheme between the whole building models and the MPC development framework.

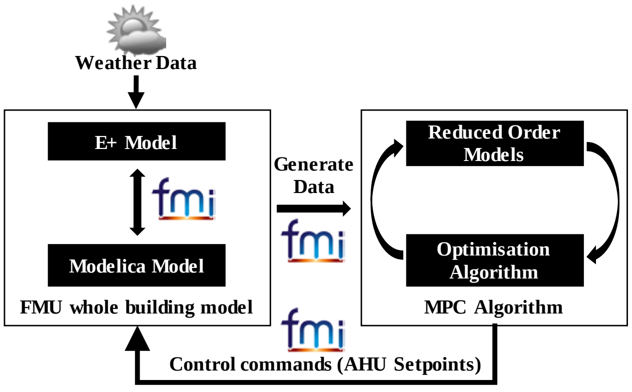


Figure 7 MPC development platform

The main purpose of the whole building models is to generate synthetic data to train the MPC algorithm on one side and on the other side to be used as a test-bed to check that the developed MPC performs as expected. The MPC algorithm development is explained in the following sections.

5.2 Design of the MPC

The developed MPC aims at minimising the energy consumption at building level while maintaining thermal comfort. The MPC is based on iterative, finite-horizon, optimisation of the objective function based on the dynamic model of the plant. The optimization is defined over the interval *[k,k+H]*, where *k* is the current time and *H* is the prediction (optimisation) horizon. Typically, only the first (discrete time) step of the solution is implemented, then the plant state is sampled again, and a new optimization is repeated in a receding horizon fashion (see Figure 8).

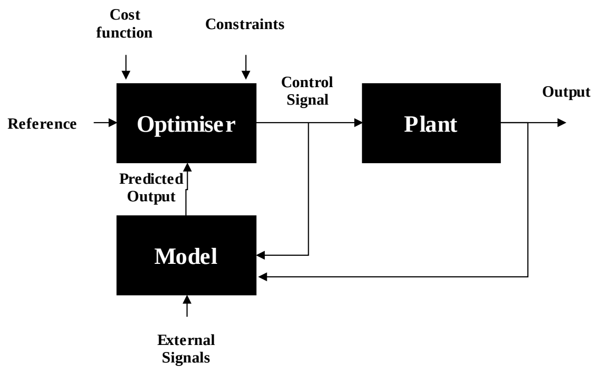


Figure 8. General diagram of a MPC.

The optimisation problem used by the proposed MPC is the following:

$$\min_{x_t, x_{rh}} \sum_{k=1}^{H} [w_c(k)p_c(k) + w_h(k)p_h(k)] + \lambda \sum_{k=1}^{H} \{w_t[t_r(k) - t_i(k)]^2 + w_r[rh_r(k) - rh_i(k)]^2\} \quad 1$$

subject to constraints:

$$(t_r, rh_r, p_c, p_h) = f(x_t, x_{rh}, \Theta) \quad 2$$

$$x_{t,min} \le x_t \le x_{t,max} \quad 3$$

$$x_{rh,min} \le x_{rh} \le x_{rh,max} \quad 4$$

$$|x_t(k+1) - x_t(k)| \le \rho, k = 0, \dots, H-1 \quad 5$$

$$|x_{rh}(k+1) - x_{rh}(k)| \le \gamma, k = 0, \dots, H-1 \quad 6$$

$$t_{min} \le t_r \le t_{max} \quad 7$$

$$rh_{min} \le rh_r \le rh_{max} \quad 8$$

The optimization (control) variables are the supply temperature, $\boldsymbol{x_t}$, and the supply relative humidity (RH), $\boldsymbol{x_{rh}}$, setpoints for the AHU. The function $(\boldsymbol{t_r}, \boldsymbol{rh_r}, \boldsymbol{p_c}, \boldsymbol{p_h}) = f(\boldsymbol{x_t}, \boldsymbol{x_{rh}}, \boldsymbol{\Theta})$ is the building model that predicts the room temperature, room RH and the cooling and heating power as function of the control variables and the external inputs (predicted weather conditions and past room state). The problem computes the optimal setpoints for a prediction horizon of 24 hours. The objective function has two terms: 1) energy consumption and 2) deviation from desired comfort level. The regularization weight λ is a positive constant that balances the trade-off between energy consumption and desired thermal comfort. The constraints impose lower and upper bounds for the supply AHU set-points and for the room temperature and RH. Smoothness constraints are also included to avoid abrupt changes in the setpoints in time.

We tested two types of reduced models for the MPC: 1) First principle based models and 2) long short-term memory recurrent neural network (LSTM-NN). The first-principle based model uses a simplified physical model for the AHU coupled with a linear auto-regressive model for the room envelope. The coefficients of the auto-regressive model can be updated every week, or every season based on observed data.

The LSTM-NN is a purely data-driven model that represents the whole system as an input-output function. The LSTM-NN is trained from simulated data (model from sections 3 and 4) using different weather profiles and setpoint strategies to avoid overfitting and achieve a good approximation of the system dynamics.

5.3 Results from the MPC

We tested the developed MPC in a simulation model of the building demo zone in the test site. The model consists of a room with an AHU. The MPC was tested using both the reduced physical models and NN models in the optimization for a period of 28 days in summer (June-July) with the constraints described in Table 2.

Table 2. Constraints imposed in the MPC problem.

Ideal room temp. (t_i)	21.5 °C	Ideal room RH (rh_i)	50%
Max. room temp. (t_{max})	24 °C	Max. room RH (rh_{max})	55%
Min. room temp. (t_{min})	20 °C	Min. room RH (rh_{min})	45%
Max. supply temp. ($x_{t,max}$)	27 °C	Max. supply RH ($x_{rh,max}$)	65%
Min. supply temp. ($x_{t,min}$)	21 °C	Min. supply RH ($x_{rh,max}$)	45%

The results are summarized in Table 3 also showing the results obtained using a standard PID controller for the room. The MPC is evaluated using two scenarios: a low comfort configuration with λ=1 and a higher comfort configuration with λ=100. The MPC coupled with the NN model and with λ=1 yield savings of approximately 62% in the cooling energy and 26% in the heating energy compared to the PID controller. However, it should be noted that the NN-based MPC achieves an average room temperature close to the ideal temperature but the room RH is far from the ideal. On the other hand, the results with λ=100 achieve room temperatures and RH close to the ideal ones at the expense of having a slightly larger energy consumption than the PID controller. In addition, it should be noted that the NN-based MPC are able to follow better the thermal comfort constraints than the Physical MPC. OIt is worth noticing that the RH has a big impact in the overall energy consumption of the building, thus, better strategies for RH control should be investigated, e.g. wider range for RH.

6 Conclusions

This paper presented the developments of a detailed building energy model aimed at improving cooling control for further coupling with District Cooling.

6.1 Modelica use

In INDIGO, some advantages in the use of Modelica for modelling the energy systems where demonstrated:

Table 3. MPC results for 28 days between June and July using reduced Physical models (Ph-MPC) and NN models (NN-MPC).

		Ph-MPC		*NN-MPC*	
	PID	$\lambda = 1$	$\lambda = 100$	$\lambda = 1$	$\lambda = 100$
Cooling energy (kWh)	14,227	4,648	15,244	5,286	15,063
Heating energy (kWh)	9,234	7,726	13,051	6,759	11,942
MAD temp. (ºC)	0.16	2.27	2.01	1.13	1.16
Mean temp. (ºC)	21.47	23.77	23.51	22.54	22.6
STD temp. (ºC)	0.2	1.11	0.96	0.95	0.89
Min temp. (ºC)	19.49	21.4	21.45	20.02	20.25
Max temp. (ºC)	23.28	26.74	26.08	27.06	26.31
MAD RH (%)	3.47	4.66	5.15	8.19	2.41
Mean RH (%)	52.31	53.98	45.09	57.23	48.57
STD RH (%)	3.7	4.49	2.81	6.92	2.82
Min RH (%)	20.96	0.05	4.69	39.37	0.06
Max RH (%)	93.98	96.86	60.36	99.26	96.45

- The hybrid modelling approach the Modelica enables in a single tool simplifies the modeler work, reduces error and provides an easier to use and understand approach to system's modelling. In Modelica, mechanical, electrical, and thermodynamic modelling can be integrated in the same model, including control algorithms;
- The object-oriented approach enables model reusability on the one side and on the other side, allows for modelling the physical systems following the physical structure as opposed to a signal structure used in other languages. This provides the clear advantage that models are easier to understand;
- The extension capabilities of Modelica via the Functional Mock-up Interface allowed to integrate models from different tools using an independent and standardized API into the MPC development environment, thus providing an integral solution for data analysis, simulation and optimization in one single environment.

6.2 Functional Mock-up Interface use

Given the variety of development tools used in INDIGO, to avoid the imposition of a single tool, which would have limited developments and to allow a seamless integration of the different developments, the use of Functional Mock-up units was agreed since all development tools were found to be compatible with the FMI standard. Embarking in such approach provided several benefits but also some challenges for INDIGO development which are described in the following sections.

6.2.1 Benefits

Amongst the benefit of using the FMI standard we found:

- FMI is a standardised approach. For developers this means there is less effort in the integration between tools. They only need to agree on the variables to be exchanged as opposed to have to develop the integration interface itself.
- The FMI is tool-independent which translates in a seamless model exchange across different tools
- Models can be re-used for different purposes. This is a combined advantage between the use of Modelica and the use of FMI. All public variables can be made accessible by the FMU which means that there is no need to change the model in case a new variable needs to be exported. This results in less effort and better quality of the developments produced with FMI and Modelica based models.

6.2.2 Challenges

In INDIGO, the use of the FMI standard also exposed some challenges:

- Ensuring the efficiency and robustness of the models is fundamental for the FMUs generated in terms of usability, performance and error-handling when running simulations and generating data;
- While the standard for data exchange is certainly excellent, the integration over two platforms presents another challenge in the standardisation of the data to be exchanged.
- Different simulation tools might provide different results mainly due to the use of different solvers. This is something that needs to be acknowledged since it is, for the time being and for a typical modeller, not trivial;
- The parametrization of the models is not necessarily evident when doing model exchange, thus it requires modeller's attention;
- Documentation of the FMU needs to be provided separately from the model. If the modeller is used to Modelica where the documentation is contained within the model, this might be overlooked when exchanging the models

6.3 MPC implementation

MPC is a powerful control strategy that anticipates to future events and takes control actions accordingly. However, in order to achieve real-time control, the optimization problem has to be solved faster than the sampling time of the system. Thus, the reduced models used within the MPC are of great importance. On one hand, the model needs to be sufficiently simple and fast to be used in the optimization loop, and on the other hand, the model needs to be accurate enough to avoid erroneous control strategies due to approximation errors by the reduced models.

In INDIGO, we have explored the use of NN as reduced models for MPC with satisfactory results. The advantages of NN, especially recurrent NN such as the LSTM-NN, are twofold: firstly, fast computation time to allow its use within the MPC, and secondly, high accuracy in the modelling to capture both slow and rapid dynamics of the system. However, in order to capture the correct dynamic behaviour of the system, the NN has to be trained with data that explore a large portion of the data space and model dynamics, which is not often the case with data collected from a real site. Therefore, simulation platforms, such as the one developed in INIDIGO, are a great tool to generate training data for NN models within a MPC.

Acknowledgments

Project INDIGO has received funding from European Union's Horizon 2020 research and innovation programme under grant agreement n° 696098.

References

del Hoyo Arce, I. *et al.* (2018) 'Models for fast modelling of district heating and cooling networks', *Renewable and Sustainable Energy Reviews*. Elsevier Ltd, 82(June), pp. 1863–1873. doi: 10.1016/j.rser.2017.06.109.

Nouidui, T. S., Wetter, M. and Zuo, W. (2014) 'Functional mock-up unit for co-simulation import in {EnergyPlus}', *Journal of Building Performance Simulation*. Taylor & Francis, 7(3), pp. 192–202. doi: 10.1080/19401493.2013.808265.

Reindl, D. T., Beckman, W. A. and Duffie, J. A. (1990) 'Diffuse fraction correlations', *Solar Energy*, 45(1), pp. 1–7. doi: https://doi.org/10.1016/0038-092X(90)90060-P.

Sterling, R. *et al.* (2017) 'Demand side detailed models'. doi: 10.5281/ZENODO.1137755.

Characterization of Linear Reduced Order Building Models Using Bode Plots

Moritz Lauster[1] Dirk Müller[1]

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

Abstract

Simulations of energy supply systems on the urban scale call for dedicated thermal building models with low simulation times and still considering relevant dynamic effects. A common approach for such models are reduced order thermal networks that model heat transfer and storage via thermal resistances and capacitances. To contribute to the open question, how much wall elements should be used in such approaches, this paper characterizes and compares four different model topologies with one, two, three and four wall elements. The characterization using the Linear Analysis toolbox in Modelica and Bode plots in Python reveals a significantly different behavior of the One-Element-Model compared to the higher order models. In consequence, the Two-Elements-Model with comparably low simulation times and a similar behavior as the higher order models qualifies for urban scale simulations.

Keywords: Modelica, Reduced Order Model, Urban Building Energy Model, Bode plot, Linear Analysis Toolbox

1 Introduction

In the context of global warming and anthropogenic greenhouse gas emissions, innovative energy supply systems play an important role to increase energy efficiency. In particular, when supplying entire districts with heat, this calls for sophisticated dynamic building models to consider heat storage effects and compare different system options.

When simulating large numbers of buildings, reasonable simulation times in combination with an appropriate model complexity can be a challenging task. Still, the models need to account for relevant physical effects and details to be able to reflect the buildings' real thermal behavior. In this regard, reduced order models based on thermal networks are an interesting option. They use thermal networks analog to electrical circuits and model heat transfer and storage via thermal resistances and capacitances. The theory of such models is well researched and discussed in Clarke (2001); Davies (2004) and Hensen and Lamberts (2011). The model's complexity and the simulation time is determined by the layout of the network and the number of resistances and capacitances. Furthermore, the number of resistances and in particular the number of capacitances determine the spatial and physical resolution of the model and thus define the accuracy, with which the dynamics of the building are reproduced. In consequence, the structure of the model needs to be aligned to the simulation task, the resolution of dynamic effects and acceptable simulation times.

Reduced order models account for relatively small simulation times by using a small number of state variables, in the case of thermal networks associated solely to thermal capacitances. In this way, they qualify for urban scale simulations, where uncertainties due to unknown boundary conditions and estimated parameters outweigh modeling accuracy. Still, this leads to the question, what the optimal number of capacitances is for the case of urban scale simulations.

This question calls for a detailed analysis and characterization of the dynamic behavior of promising reduced order modelling options. To do so, Bode plots offer the ability to analyse the magnitude and phase shift of a model output compared to a model input for an entire range of excitation frequencies. They allow a dedicated comparison of different reduced order models for a broad range of frequencies as well as for typical frequencies present in the built environment as e.g. done in Akander (2000) and Ramallo-González et al. (2013). In consequence, Bode plots support finding the optimal number of capacitances, from where adding further elements would not substantially increase model accuracy.

This paper aims at contributing to the field of urban scale simulations by characterizing four different reduced order models using Bode plots. It emphasizes on using Modelica and the Linear Analysis toolbox in combination with a Python-based Bode plot analysis. The next chapter gives an introduction to reduced order building modeling to highlight the impact of thermal capacitances and presents the four investigated model topologies. Afterwards, the paper outlines the setup of the characterization and presents the results using Bode plots to answer the question regarding a reduced order model optimized for urban scale simulations.

2 Reduced Order Building Models

As mentioned before, when reducing the order of thermal network models, the question arises, which capacitances resp. states are crucial for the dynamic behavior and which can be omitted. Focusing on one thermal zone, the typical entity in building performance simulations, the number of capacitances depends on the number of wall elements and the discretization of these wall elements. This leads to two options to set up reduced order models:

1. Reducing the number of wall elements, e.g. by merging roof, floor and external walls to one wall element.

2. Reducing the number of capacitances per wall element, which represent the discretization of the wall.

The second aspect is already well investigated (e.g. in Davies 1994 and Rouvel and Zimmermann 1997, 1998) and lead to the theory of the periodic depth of penetration, which is standardized in DIN EN ISO 13786 (Deutsches Institut für Normung, 2008a). In consequence, this aspect was found to be of minor importance, so the number is typically fixed to one capacitance per wall element and the capacity depends on the excitation frequency. Two common standardized reduced order models, described in DIN EN ISO 13790 (Deutsches Institut für Normung, 2008b) and VDI 6007-1 (Verein Deutscher Ingenieure, 2015), follow this approach. However, these two models highlight the differences for the first aspect. While the DIN EN ISO 13790 lumps all walls to one wall element and is explicitly thought for monthly resolution at maximum, the VDI 6007-1 models asymmetrically (external walls) and symmetrically loaded (internal) walls separately. Still, there is no common agreement, how many wall elements are necessary for hourly heat demand calculations and which elements should be lumped.

To contribute to this question, this paper investigates four different model topologies by lumping either all walls to one element (as for DIN EN ISO 13790, Figure 1), distinguish between external and internal walls (as for VDI 6007-1, Figure 2), further divide between walls exposed to solar radiation and floor plates (Figure 3) and finally

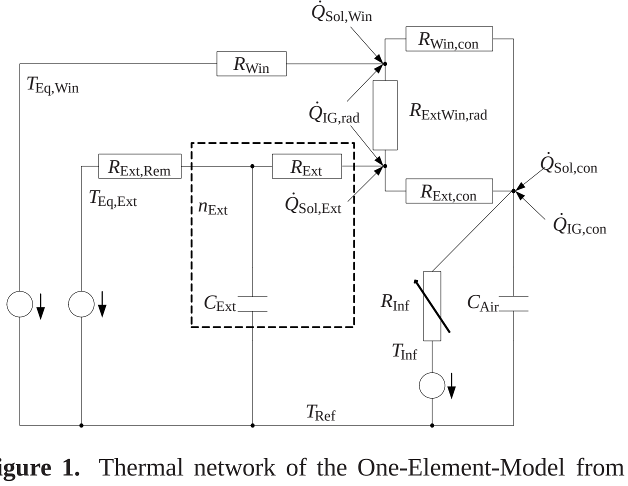


Figure 1. Thermal network of the One-Element-Model from `AixLib`.

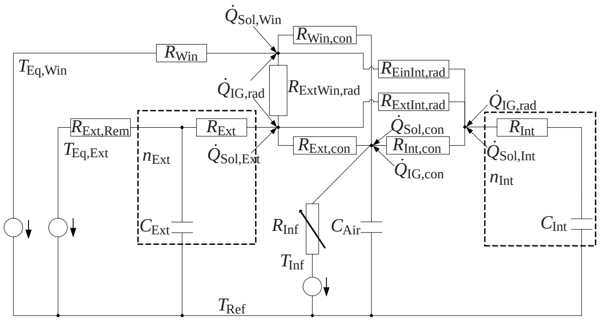


Figure 2. Thermal network of the Two-Elements-Model from `AixLib`.

separate the roof elements as well (Figure 4). To keep the same basic topology for all options, all models are based on the layout and principles of the VDI 6007-1. Figure 2 represents the original VDI 6007-1 model except for a polygon network instead of a star network for the internal radiation circuit to be able to extend the network without loss of accuracy (as described in Davies 1993).

In addition, heat transfer through windows is handled separately to the external walls, since windows commonly do not incorporate thermal mass and merging windows and walls would lead to a delay in the windows' heat transfer. The heat transfer through windows and heat transfer to the ambient and heat storage in the external walls is handled via resistances and capacitances in the left part of Figure 2. The right part takes care of heat storage in internal walls, while the center part deals with convective and radiative heat exchange within the thermal zone. Both effects are represented by one circuit each, the indoor air temperature can be measured at the star point of the convective circuit. This point connects also to a thermal resistance that is used for infiltration of outdoor air (resp. the associated heat flux) through gaps in the thermal zone's envelope. Further explanations and details about the model can be found in Remmen et al. (2017) and VDI 6007-1 Verein Deutscher Ingenieure (2015).

As mentioned, the One-Element-Model in Figure 1 neglects the differing behavior of internal walls and merges them with external elements to one wall element. With the Two-Elements-Model in Figure 2 in between, the Three-Elements-Model in Figure 3 separates walls exposed to solar radiation and exposed directly to the ground. This follows the assumption that ground coupled wall elements such as floor plates behave thermally different due to the excitation with a very low frequency (with a time constant of about one year). Thus, merging them with elements exposed to solar radiation (excitation with a time constant of one day) might lead to smearing the dynamics of the themal zone. The same argument applies for the Four-Elements-Model in Figure 4, where the roof is taken care of seperately to the external walls. However, the time constants of roof elements and external walls should be similar, but the excitation is shifted in time for horizontal and vertical elements. Besides the number of state variables,

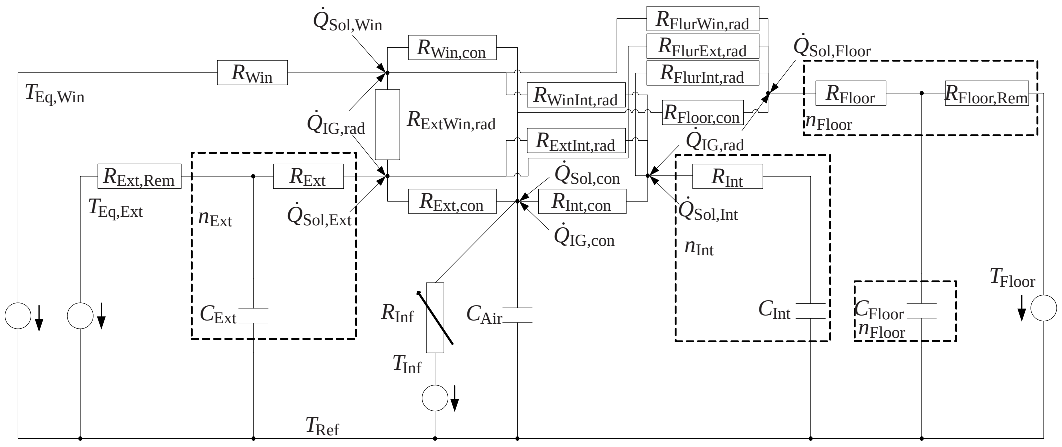


Figure 3. Thermal network of the Three-Elements-Model from `AixLib`.

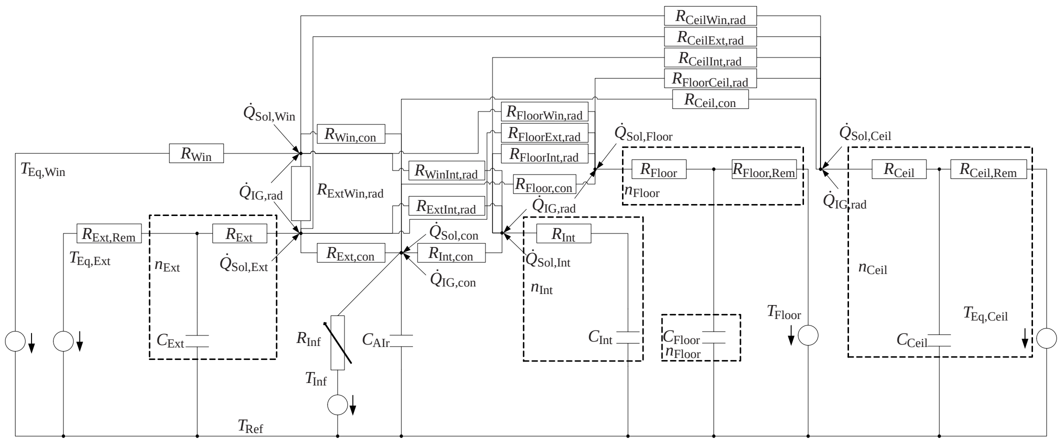


Figure 4. Thermal network of the Four-Elements-Model from `AixLib`.

simulation time is influenced by the way the physical effects are modelled. In the best case, all phenomena can be modelled with a linear approach, which further lowers simulation time. In the case of reduced order models, there are four effects that can be linearized to obtain a fully linear model.

1. The indoor radiative heat exchange follows the Stefan-Boltzmann law including surface temperatures to the power of four. This can be linearized around a given temperature, usually the zone's set temperature.

2. Convective heat exchange depends on free and forced convection and includes nonlinearities. For typical conditions, different standards (e.g. DIN EN ISO 6946 Deutsches Institut für Normung 2015) provide constant values to linearize the effects.

3. The absorption and transmission of solar radiation on inclined surfaces requires calculation of angles that include nonlinear equations. Since these calculations to not depend on the zone's state variables, all values can be precomputed and serve as inputs to the building model itself.

This setup with linear approaches and a reduced number of state variables leads to models with a relatively low complexity, e.g. for the Two-Elements-Model with 28 unknowns, two state variables and one algebraic loop with constant parameters.

The merging or separation of the building elements leads to four different models with the open question, which model considers all dominant dynamics while neglecting all others to have as small simulation times as possible. For this purpose, all four models need to be characterized in a dynamic way as proposed in the next chapters.

3 Characterization

The characterization requires the definition of a use case with a fixed geometry and a known set of physical properties. In this study, the use case is based on a single box-shaped room as the typical layout of a thermal zone. This room follows the geometrical definition of the international validation guideline ASHRAE 140 (ASHRAE, 2007) with a net floor area of 48 m^2, two walls with 21.7 m^2 and two walls with 16.2 m^2 (one internal and one external wall each) as shown in Figure 5. Changes of the geometry do not influence the results, as long as the rela-

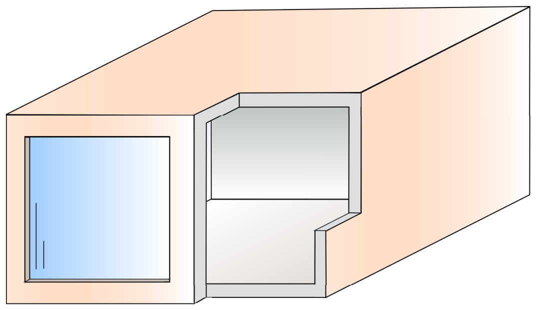


Figure 5. Layout of the test room.

tion between the areas is kept similar. For the given case with similar areas for external walls, roof, floor plate and internal walls, we expect the largest differences between the four models, since all elements have a similar impact.

To cover typical physical properties for the building materials and the wall constructions in the German building stock, the characterization covers 24 setups with insulation levels varying from "EnEV 2009", "EnEV 2002" and "WSchV 1995" to "WSchV 1984", representing German insulation standards of different years. As a second aspect, the building mass is varied from "Light-weight" and "Medium-weight" to "Heavy-weight". Since the cases "EnEV 2009" and "WSchV 1984" turned out to be the extreme cases, the following chapters will focus on these setups.

The use case in all 24 setups and for all four model topologies has been modelled in Modelica using the library AixLib (Müller et al., 2016). AixLib is one of four application libraries based on the same Modelica IBPSA core library, described in Wetter et al. (2015). Both, the AixLib and the IBPSA library, are developed fully open-source and are freely available at `https://github.com/RWTH-EBC/AixLib` and `https://github.com/ibpsa/modelica-ibpsa`. The reduced order models are part of the core library and thus of all four application libraries.

To characterize the models using Bode plots, all four model topologies have been transformed to state-space representation of the form

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{1}$$

$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t) \tag{2}$$

with $\mathbf{x}$ as state vector, $\mathbf{y}$ as output vector and $\mathbf{u}$ as input vector. A stands for the states matrix, B for the input matrix, C for the output matrix and D for the feedthrough matrix. This is a valid approach since we deal with linear time-invariant models. The necessary matrices A,B,C and D can be derived using the ModelicaLinearSystems2.ModelAnalysis.Linearize function of the Linear Analysis toolbox. The created .mat-files containing these matrices can directly be used to create Bode plots in Python with the help of the Python package ModelicaRes, available at `https://github.com/kdavies4/ModelicaRes`. In this way, a dedicated analysis of the dynamic behavior of Modelica models is convenient to perform in a semi-automated process.

4 Results

This chapter presents the results for the given use case, generated with the process sketched above. As mentioned, it focuses on the setups "EnEV 2009" and "WSchV 1984" as they represent the extreme cases of all setups. The Bode plots show the dynamic behavior for all four model topologies over a frequency range from 10^{-6} to 10^{-3} Hz, in particular highlighting typical excitation frequencies in buildings at 1.15^{-5} Hz (daily) and 2.77^{-4} Hz (hourly). Since buildings are typically excited by external (outdoor air temperature variations) and internal sources (internal gains, convective and radiative), the Bode analysis is performed twice for each setup. The observed output is the indoor air temperature as the target value. It would be even more benefitial to observe the heat flow through the constructions, which would require strategies to compare the overall heat flows of the different models or observing multiple outputs. These topics are marked as future work and not covered in this paper. The aim is to identify the model with the lowest order that shows no major differences to higher order models for indoor air temperature as output.

Figure 6 shows the Bode plot for external excitation of the heavy-weight "EnEV 2009" setup. The upper part focuses on the magnitude of the output, while the lower part concentrates on the phase shift of the output compared to the input. The pattern of all four models is similar and follows the PT_1-behavior of a low-pass filter. The low-pass filter originates from the given RC-element for external wall elements, e.g. shown in Figure 1. In comparison to the other models, only the One-Element-Model reveals some deviations regarding the damping as well as for the phase shift. In particular for the typical daily and hourly excitation frequencies, all models except the One-

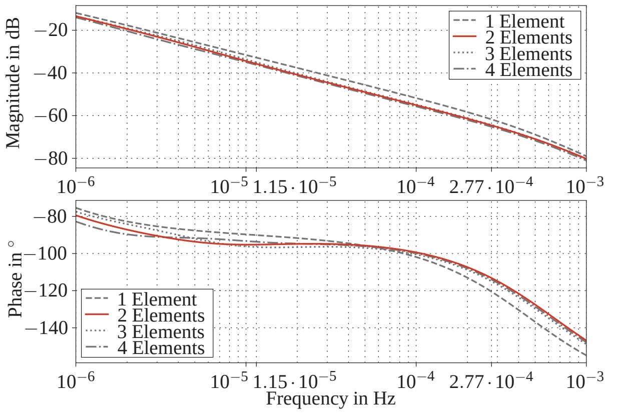


Figure 6. Bode plot for external excitation with heavy-weight setup (EnEV 2009) and indoor temperature as observed output.

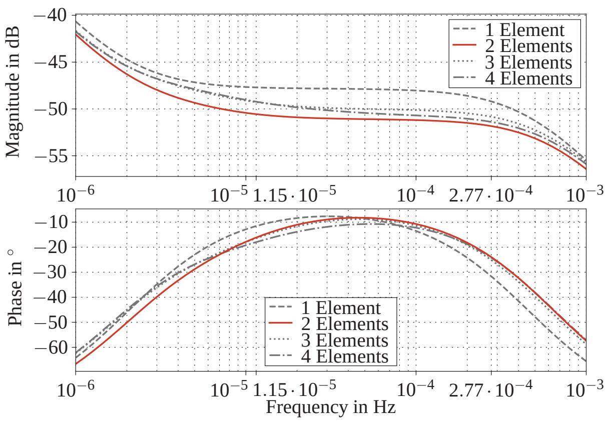


Figure 7. Bode plot for internal excitation with heavy-weight setup (EnEV 2009) and indoor temperature as observed output.

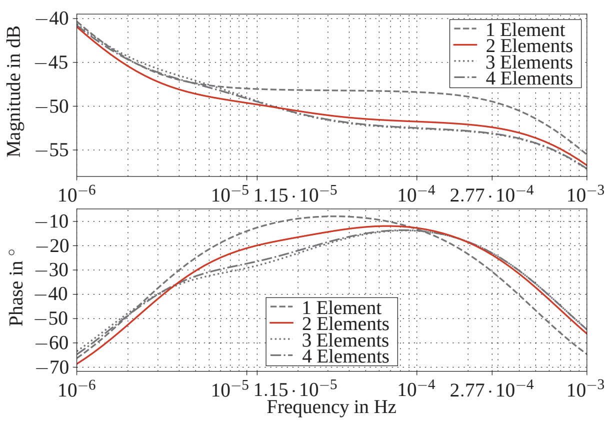


Figure 9. Bode plot for internal excitation with light-weight setup (WSchV 1984) and indoor temperature as observed output.

Element-Model behave almost identical.

Figure 7 shows the Bode plot for the same heavy-weight setup in the case of internal excitation. All models show a PIT_1-behavior that is related to the interference of the wall elements' PI-behavior and a PT_1-behavior of the air volume. In particular the One-Element-Model deviates from the other model topologies and reveals a phase shift in the direction of lower excitation frequencies. This leads to significant differences, especially for an hourly excitation, and can be explained by the missing consideration of the internal walls. In this way, the One-Element-Model on the one hand neglects parts of the zones' thermal mass and other hand does not consider heat transfer between external and internal masses. This is mainly visible for internal excitation, since external excitation is highly damped by the external wall elements. Though, the other three model topologies deviate as well from each other regarding the magnitude as well as the phase shift for low excitation frequencies.

In addition to the heavy-weight setup, Figure 8 shows the Bode plot for external excitation of the light-weight "WSchV 1984" setup. The patterns of all four models are comparable to the behavior in Figure 6, although the Three- and Four-Elements-Model show significant deviations compared to the One- and Two-Elements-Model. The impact of these deviations is hard to assess at this point and will be further investigated in Figure 10.

To complete the set of Bode plots, Figure 9 shows the plot for internal excitation of the light-weight setup. The discussed deviations of the One-Element-Model occur in this plot in an amplified manner. For the magnitude as well as for the phase shift, the One-Element-Model clearly deviates from the other topologies. Two-, Three- and Four-Element-Model show similar deviations as in Figure 7.

Resulting from the Bode plots, the Two-, Three- and Four-Elements-Model predominantly show a similar behavior, in particular for typical excitations with daily and hourly time constants. All models have a similar DC-gain of approximately 1, varying between the models at the 10th decimal place. Solely the One-Element-Model reveals major deviations from the other models, in partic-

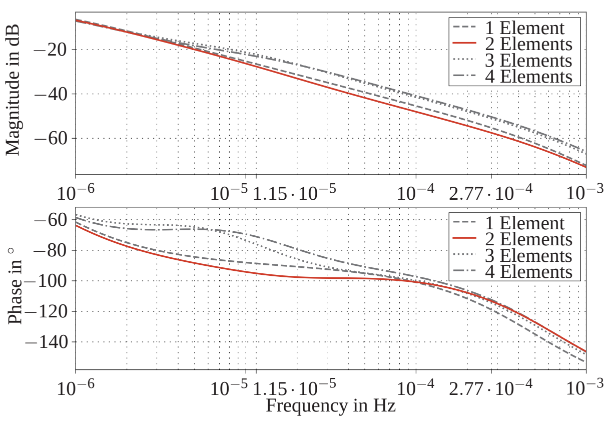


Figure 8. Bode plot for external excitation with light-weight setup (WSchV 1984) and indoor temperature as observed output.

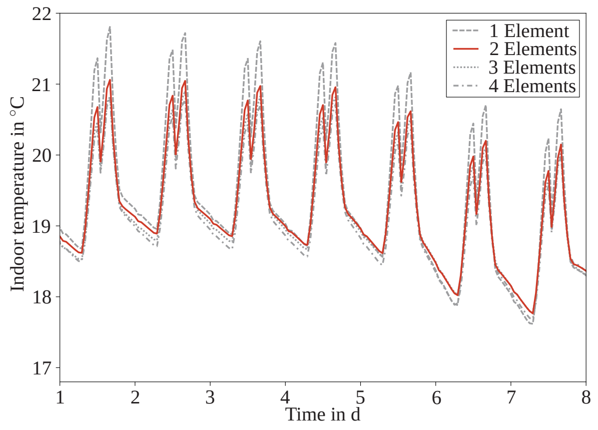


Figure 10. Comparison of all four model topologies in the time domain for the medium-weight setup (EnEV 2002).

Table 1. Maximum and mean deviation of the indoor temperature in K between two- and four-elements-model for setups "EnEV 2009 Heavy", "EnEV 2002 Medium" and "WSchV 1984 Light".

	Heavy	Medium	Light
Max	0.3	0.4	0.7
Mean	0.1	0.1	0.1

ular for internal excitation. The deviations in general tend to rise from heavy-weight to light-weight setups and majorly occur for internal excitation. To evaluate the impact of these deviations, it is necessary to complement the analyses in the frequency domain by investigations in the time domain.

For this, yearly simulations of all four model topologies are performed with time-dependent weather data (TRY 2010 data for Mannheim, Germany), varying internal gains (generic profiles for persons and machines, convective and radiative) and free-floating indoor air temperature. Figure 10 shows seven days in spring, where typical indoor air temperatures between 17 and 21°C occur and the typical excitation frequencies of one hour and one day can be observed. For reasons of clarity, the figure focuses on the medium-weight setup "EnEV2002". The figure shows a significant overshoot of the One-Element-Model in times of local maximum temperatures. This correlates with the observations in Figure 7 and Figure 9, where the One-Element-Model shows a lower damping of internal excitations. This behavior leads to maximal differences of the indoor air temperature of 1.2 K compared to the other model topologies. In comparison, the difference between Two- and Four-Elements-Model for the same setup is 0.4 K, as given in Table 1. The differences between the higher-order models tend to rise from heavy-weight to light-weight setups.

Given the difference in the frequency as well as in the time domain between the One-Element-Model and the other topologies, this approach with only one state variable for the wall elements seems inappropriate for dynamic builing performance simulations. Following the approach to keep the number of states as small as possible, the Two-Elements-Model comes into focus and shows a significantly better behavior. With these results at hand, the Two-Elements-Model qualifies for dynamic heat demand calculations on urban scale, where simulation time plays a major role and modelling simplifications are outweighed by uncertainties of the boundary conditions.

5 Conclusions

Urban scale simulations of large building stocks for energy efficient supply systems call for dynamic building models with low simulation times while accepting modelling simplifications. Such simplifications are typically outweighed by uncertainties in boundary conditions on urban scale. A common approach for such building models are reduced order models based on thermal networks, which represent heat transfer and storage via thermal resistances and capacitances. The models' simulation times, complexity and order is correlated to the number of state variables resp. thermal capacitances. Although this topic has been well researched and standardized approaches exist on single wall level, the questions remains, how many wall elements are necessary to model heat storage effects on urban scale in a sufficient way.

This paper contributes to this question by characterizing four model topologies with one, two, three and four wall elements. The thermal masses of a given use case are therefore merged all together, separated into internal and external masses, additionally with a separate element for the floor plate or further splitted up to a separate roof element. The use case is a simple, box-shaped room, which is commonly used for model validation. To take into account different insulation and building mass levels, the use case contains 24 setups from light-weight, well insulated scenarios up to heavy-weight, hardly insulated versions. As extreme cases, the characterization focuses on an heavy-weight "EnEV 2009" and on a light-weight "WSchV 1984" setup.

All model topologies and setups are modelled using the Modelica library AixLib, `https://github.com/RWTH-EBC/AixLib`, an application library of the IBPSA core library, `https://github.com/ibpsa/modelica-ibpsa`.

The characterization makes use of Bode plots and comparisons in the time domain to analyse the behavior of all four topologies with regard to magnitude and phase shift of the indoor air temperature compared to a chosen input over a range of excitation frequencies. Typical excitation frequencies in the field of building performance simulation with time constants of one day and one hour can in this way clearly be highlighted. The necessary matrices to describe the model's transfer function can be obtained with the ModelicaLinearSystems2.ModelAnalysis.Linearize function out of the Linear Analysis toolbox. The resulting files can directly be used to create dedicated Bode plots in Python with the help of the Python package ModelicaRes, `https://github.com/kdavies4/ModelicaRes`.

The results show that the behavior of the One-Element-Model significantly differs from the higher order models, for the magnitude as well as the phase shift, when observing the indoor air temperature while exciting outdoor air temperature or internal gains. This originates in neglecting internal masses, what leads to a significantly different transfer function.The Two-, Three- and Four-Elements-Model show slight differences in the Bode plots, what requires further analyses in the time domain. The simulation of one year reveals maximal differences in the free-floating indoor air temperature between Two- and Four-Elements-Model of 0.4 K. The same case shows differences between One-Element-Model and higher order models of 1.2 K.

Based on these results, the Two-Elements-Model qualifies for urban scale simulations with low simulation times while keeping a similar behavior compared to higher order models. As the differences partly depend on the insulation and thermal mass level, further research should result in an adaptive method to automatically choose a reduced order modelling approach based on these properties.

Acknowledgements

We gratefully acknowledge the financial support by BMWi (German Federal Ministry of Economic Affairs and Energy) under promotional reference 03ET1562A.

References

Jan Akander. *The ORC method: Effective modelling of thermal performance of multilayer building components*. Dissertation, The Royal Institute of Technology, Stockholm, 2000.

ASHRAE. Standard Method of Test for the Evaluation of Building Energy Analysis Computer Programs, 2007.

J. A. Clarke. *Energy simulation in building design*. Butterworth-Heinemann, Oxford, 2. ed edition, 2001. ISBN 0750650826.

M. G. Davies. Definitions of Room Temperature. *Building and Environment*, 28(4):383–398, 1993. doi:10.1016/0360-1323(93)90015-U.

M. G. Davies. The Thermal Response of an Enclosure to Periodic Excitation: The CIBSE Approach. *Building and Environment*, 29(2):217–235, 1994.

Morris G. Davies. *Building Heat Transfer*. John Wiley & Sons, Hoboken, NJ, 2004. ISBN 978-0-470-84731-2.

Deutsches Institut für Normung. Thermal performance of building components - Dynamic thermal characteristics - Calculation methods, 2008a.

Deutsches Institut für Normung. Energy performance of buildings - Calculation of energy use for space heating and cooling, 2008b.

Deutsches Institut für Normung. Building components and building elements - Thermal resistance and thermal transmittance - Calculation method, 2015.

Jan Hensen and Roberto Lamberts, editors. *Building performance simulation for design and operation*. Spon Press, London and New York, NY, 2011. ISBN 978-0-415-47414-6.

Dirk Müller, Moritz Lauster, Ana Constantin, Marcus Fuchs, and Peter Remmen. AixLib - An Open-Source Library within the IEA-EBC Annex60 Framework. In *BauSIM 2016: Sixth German-Austrian IBPSA Conference*, pages 3–9, 2016.

Alfonso P. Ramallo-González, Matthew E. Eames, and David A. Coley. Lumped parameter models for building thermal modelling: An analytic approach to simplifying complex multilayered constructions. *Energy and Buildings*, 60:174–184, 2013. ISSN 03787788. doi:10.1016/j.enbuild.2013.01.014.

Peter Remmen, Moritz Lauster, Michael Mans, Marcus Fuchs, Tanja Osterhage, and Dirk Müller. TEASER: An open tool for urban energy modelling of building stocks. *Journal of Building Performance Simulation*, pages 1–15, 2017. doi:10.1080/19401493.2017.1283539.

Lothar Rouvel and Frank Zimmermann. Ein regelungstechnisches Modell zur Beschreibung des thermisch dynamischen Raumverhaltens. *HLH Lüftung/Klima - Heizung/Sanitär - Gebäudetechnik*, 48,49(10, 12, 1):66–75, 24–31,18–29, 1997, 1998.

Verein Deutscher Ingenieure. Berechnung des instationären thermischen Verhaltens von Räumen und Gebäuden - Raummodell, 2015.

Michael Wetter, Marcus Fuchs, Pavel Grozman, Lieve Helsen, Filip Jorissen, Moritz Lauster, Dirk Müller, Christoph Nytsch-Geusen, Damien Picard, Per Sahlin, and Matthis Thorade. IEA EBC Annex 60 Modelica Library - An International Collaboration to Develop a Free Open-Source Modelica Library for Buildings and Community Energy Systems. In *Building Simulation 2015: 14th Conference of International Building Performance Simulation Association*, pages 395–402, 2015.

BIM2Modelica – An open source toolchain for generating and simulating thermal multi-zone building models by using structured data from BIM models

Christoph Nytsch-Geusen[1] Jörg Rädler[1] Matthis Thorade[2] Carles Ribas Tugores[3]

[1]Institut für Architektur und Städtebau, Berlin University of the Arts, Germany, `nytsch@udk-berlin.de`
[2]Modelon, Germany, `matthis.thorade@modelon.com`
[3]AEE INTEC, Austria, `c.ribastugores@aee.at`

Abstract

This contribution describes an open source toolchain which can transfer BIM models of 3D building constructions from CAAD programs into executable thermal multi-zone buildings models based on Modelica building energy simulation libraries. For this purpose, different open source libraries and tools were integrated into a Python-based software architecture of the toolchain: the IfcOpenShell/OCC libraries as the foundation for the import, analysis, and preparation of the BIM models; CoTeTo as the tool for the template-based code generation of the Modelica building models; the BuildingSystems library as the base for the thermal multi-zone building models; and JModelica as the simulation tool to perform the simulation analyses.

While the first part of the paper describes the general approach and the software architecture of the toolchain, the second part illustrates its application with an example of a real building.

Keywords: Building Information Modeling, IFC, Modelica code generation, Multi-zone building models

1 Introduction

The graphical modelling approach of Modelica, based on visualized components, connectors, and connections fits well for 2-dimensional topologies of energy plant systems, but not for 3-dimensional shapes of buildings and the topology of their constructions. On the one hand, the manual configuration of a thermal multi-zone building in Modelica in a graphical editor, based on components of a predefined library is an error-prone process. For example, the definition of a thirteen-zone building model in Modelica leads to a mo-file with more than 1,500 lines of code and a huge number of connect statements (Nytsch-Geusen, 2017). On the other hand, architects are using modelling tools such as ArchiCAD or Revit for their 3D building designs and often also Rhinoceros for prototypical designs. All these tools are able to export the geometry and the topology of a building design as a structured BIM model, normally in the IFC format.

For this reason, different research activities during the last years have been focused on the automatic generation of Modelica building models using IFC building models as the input (e.g. Thorade et al., 2015 and Reynders et al., 2017).

The toolchain described in Thorade et al. (2015) is based on the SimModel data model (O'Donnell et al. 2011). This data structure is able to store all relevant information for building energy simulation (the building construction and the related HVAC system), which is present in the BIM model itself (the IFC file) and which is optionally added by further data sources (e.g. data for missing material properties of building constructions). All these data are gained and combined by the use of the simulation tool Simergy (https://d-alchemy.com/products/simergy): with Simergy the user imports the architectural model as an IFC file, performs a space boundary analysis to obtain the topology information for the multi-zone building model, adds additional data with the Simergy GUI, and finally exports the entire data set as a SimModel file in the xml format. In the next step, a mapping tool takes the SimModel file, which instantiates and parameterizes the component and system models from present Modelica libraries, which reflect the problem of the BIM model. Because Simergy is a commercial simulation tool and the simulation tool used here is Dymola, not all of the toolchain is open source.

The approach of Reynders et al. (2017) describes a toolchain *Ifc2Modelica v0.2*, which is based on a Python framework. It can read IFC-files, determine the building topology for multi-zone building models, and generate Modelica building models in four different levels of complexity (LOC) for the Modelica IDEAS library (Jorissen et al., 2018). The model complexity reaches from a detailed thermal multi-zone building model, where each IFC entity is 1:1 mapped to a correspondent Modelica component model (LOC1) over some intermediate steps (LOC2, LOC3) down to a maximum simplified thermal single-zone building model, where all IFC entities are mapped to a small number of Modelica wall, window and door models (LOC4). The simplification from LOC1 to LOC4 takes place by merging constructions of the same type and orientation and zones with similar conditions of use with the objective of a minimum loss of precision in the results and a maximum acceleration of the computation speed. The simulation analyses mentioned above were performed with the commercial *Dymola* tool, and the *Ifc2Modelica v0.2* toolchain is not released as an open source project.

The approach described in this paper demonstrates a Python-based complete open source toolchain, which reaches from the BIM modelling analysis up to the Modelica code generation and also supports an executable building simulation experiment, based on an open source Modelica simulation tool.

2 Toolchain

The BIM2Modelica toolchain from the IFC file up to the generated Modelica model includes three serial working Python modules (compare with Figure 1): a module for the BIM data import and analysis, a building data model for storing the analyzed and prepared information for building energy simulation, and the CoTeTo tool for generating thermal multi-zone building models based on the BuildingSystems library (http://www.modelica-buildingsystems.de).

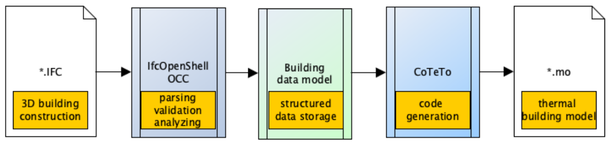


Figure 1. Software architecture of the BIM2Modelica toolchain.

2.1 BIM data analysis and preparation

The basis for the IFC data import and the subsequent data preparation is the IfcOpenShell library (IfcOpenShell, 2019) in combination with the OpenCascade library (pythonOCC, 2019). Based on these two Python libraries, a new library was implemented which enables the analysis and preparation of the imported IFC files in the following steps:

1. Filtering and sorting of all IFC types relevant for a thermal building model (types IfcSite, IfcSpace, IfcWall, IfcSlab, IfcDoor, IfCColumn, IfcWindow, etc.)
2. Extraction of all employed building constructions (type IfcMaterialLayerSet) from the imported IFC building model
3. Identification of the contact surfaces between the spaces (type IfcSpace) which represent the thermal zones and the adjacent building elements (types IfcWall, IfcSlab etc.) by means of a space boundary analysis (1st level space boundaries)
4. Determination of potential available openings in the building elements and the correspondent elements which fill them out (e.g. the relations between an IfcWall and an IfcWindow or IfcDoor)
5. Cutting of continuous building constructions (e.g. IfcWall or IfcSlab) which belong to more than one IfcSpace into sub components. Each of them can represent an individual thermal building element in the thermal building model with a potentially different thermal boundary condition (2nd level space boundaries).

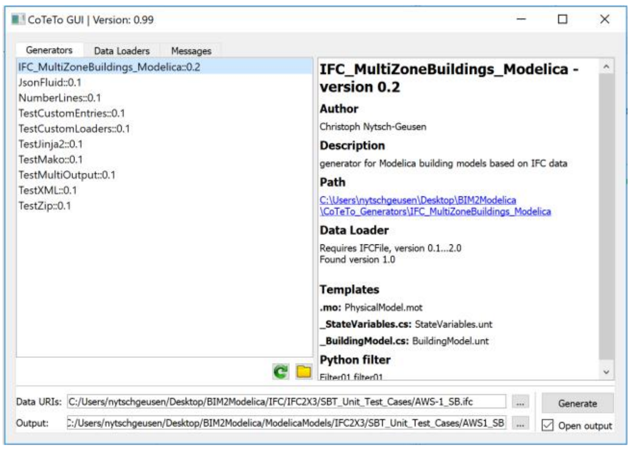
2.2 Building data model

The building data model consists of a data structure which stores all of the information in an intermediate step, before it is used for the code generation of the thermal building model, expressed in Modelica. The building data model is realized by a couple of Python classes which are able to store all of the required geometry and topology information of each thermal zone and individual building element. It also includes a list of all of the construction types used. Further, information regarding the employed building materials, the type of use for each thermal zone (ventilation rates, internal heat sources, set temperatures for heating and cooling), the building orientation and the building location can be added, if not already present in the IFC file.

The information collection of the building data model covers the typical amount of data for the parametrization of multi-zone thermal building models. Up to now, it has exclusively been used as a database for Modelica code generation, but in principle, it could also be applied to the creation of multi-zone building models for other simulation tools such as EnergyPlus or TRNSYS.

2.3 Modelica code generation

In the next step of the toolchain, the Modelica building models are generated using the information stored in the building data model. For this purpose, the Python based module CoTeTo (**Co**de **Te**mplating **To**ol) is used, which was developed in the EnEff-BIM project (see Thorade et al., 2015).



IFC_MultiZoneBuildings_Modelica::0.2
JsonFluid::0.1
NumberLines::0.1
TestCustomEntries::0.1
TestCustomLoaders::0.1
TestJinja2::0.1
TestMako::0.1
TestMultiOutput::0.1
TestXML::0.1
TestZip::0.1

IFC_MultiZoneBuildings_Modelica - version 0.2

Author
Christoph Nytsch-Geusen

Description
generator for Modelica building models based on IFC data

Path
C:\Users\nytschgeusen\Desktop\BIM2Modelica \CoTeTo_Generators\IFC_MultiZoneBuildings_Modelica

Data Loader
Requires IFCFile, version 0.1...2.0
Found version 1.0

Templates
.mo: PhysicalModel.mot
_StateVariables.cs: StateVariables.unt
_BuildingModel.cs: BuildingModel.unt

Python filter
Filter01.filter01

Data URIs: C:/Users/nytschgeusen/Desktop/BIM2Modelica/IFC/IFC2X3/SBT_Unit_Test_Cases/AWS-1_SB.ifc
Output: :/Users/nytschgeusen/Desktop/BIM2Modelica/ModelicaModels/IFC2X3/SBT_Unit_Test_Cases/AWS1_SB

Figure 2. GUI of the CoTeTo code generation tool.

CoTeTo can be flexibly configured with pluggable input, filter, and output components which support the single steps of data acquisition, preprocessing and code generation using a template system. It can be used standalone with a GUI (compare with Figure 2) or as an imported module in Python applications. The code generation step in CoTeTo is based on the Mako template engine (Mako, 2019).

2.4 BuildingSystems Library

The CoTeTo code generator for thermal multi-zone models code was designed for the predefined model classes (thermal zones, walls, windows, doors etc.) of the Modelica BuildingSystems library (Nytsch-Geusen et al., 2016). As other Modelica libraries for building energy simulation such as IDEAS, AIXLib and Buildings, the BuildingSystems library uses as its core the same Modelica IBPSA library (Modelica IBPSA library, 2019), which is the successor of the former Annex 60 library (Wetter et a. 2015).

The following code excerpt demonstrates the principle, upon which the model classes of the BuildingSystems library are instantiated and parameterized during the code generation process, based on a Mako template. Access to the required building information stored in the building data model takes place in the example over the Python dictionary `data`. Outgoing from a generalized template definition in Mako

```
% for ele in data['elementsOpaque']:
BuildingSystems.Buildings.Constructions.Wa
lls.WallThermal1DNodes ${ele.name}(
% if
generatorCfg['MODELICA_SWITCHES'].getboole
an('surTemOut'):
  show_TSur = true,
% endif
  redeclare ${ele.constructionData}
constructionData,
  angleDegAzi = ${ele.angleDegAzi},
  angleDegTil = ${ele.angleDegTil},
  AInnSur = ${ele.AInnSur},
  height = ${ele.height},
  width = ${ele.width});
% endfor
```

the Modelica code for a flexible number of wall models of a thermal building model can be generated:

```
BuildingSystems.Buildings.Constructions.Wa
lls.WallThermal1DNodes wall_4(
 redeclare Construction1 constructionData,
    angleDegAzi = 90.0,
    angleDegTil = 90.0,
    AInnSur = 0.0,
    height = 7.8,
    width = 10.000000000000002);

BuildingSystems.Buildings.Constructions.Wa
lls.WallThermal1DNodes wall_5(
 redeclare Construction1 constructionData,
 angleDegAzi = 0.0,
 angleDegTil = 90.0,
    AInnSur = 0.0,
    height = 7.8,
    width = 9.849999999999998);
 …
```

The syntax of the Mako language is similar to the Python language, but it works with the `%` sign before control statements and without indents. Therefore, a for-loop or a conditional statement in Mako needs an **% endfor** and an **% endif** in addition to the `% for` and the `% if`. Expressions within curly braces, e.g. `${ele.name}`, are evaluated, and the result is used for the code generation process. The example of the Mako template also illustrates the flexibility of the code generation process. If the Boolean expression

```
generatorCfg['MODELICA_SWITCHES'].getboole
an('surTemOut'):
```

becomes true, the Modelica parameter `show_TSur = true` is generated in the Modelica code; otherwise it is not.

2.5 Toolchain validation

The toolchain was tested with a set of 33 BIM building models in the IFC 2x3 format, which cover a broad spectrum of possible geometrical and topological structures of building constructions (Bazjanac, 2017). Figure 3 shows a subset of these building models.

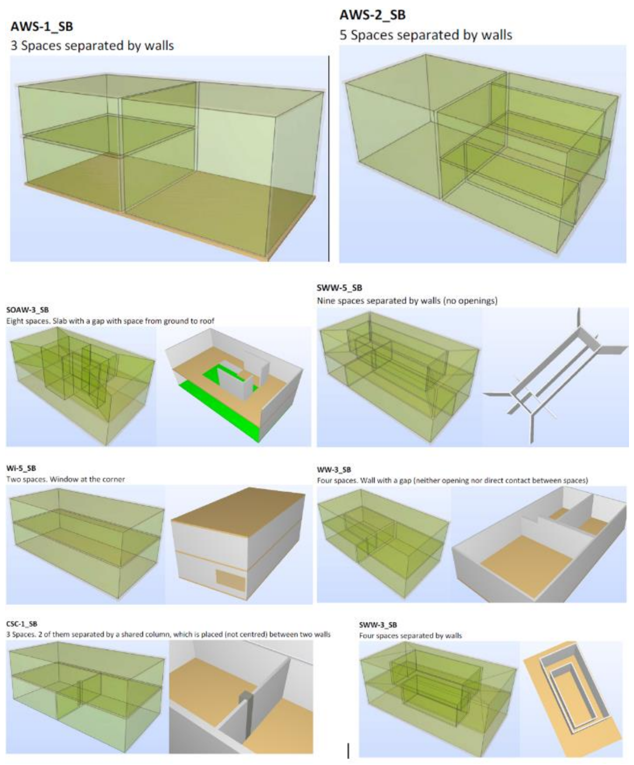

AWS-1_SB
3 Spaces separated by walls
AWS-2_SB
5 Spaces separated by walls
SOAW-3_SB
Eight spaces. Slab with a gap with space from ground to roof
SWW-5_SB
Nine spaces separated by walls (no openings)
Wi-5_SB
Two spaces. Window at the corner
WW-3_SB
Four spaces. Wall with a gap (neither opening nor direct contact between spaces)
CSC-1_SB
3 Spaces. 2 of them separated by a shared column, which is placed (not centred) between two walls
SWW-3_SB
Four spaces separated by walls

Figure 3. Exemplary IFC test cases for validating the entire toolchain from the BIM data import through the data preparation to the Modelica code generation.

In addition, three further IFC2X3 models with one, two and thirteen thermal zones were used for the validation procedure. All of the building models were constructed in ArchiCAD and afterwards exported as (IFC) BIM-models.

The validation process was executed for all of the models in three serial steps:

1. Correct translation of the BIM model into the building data model.
2. Correct generation of the Modelica building model with the Mako template.
3. Successful simulation of the generated building models with JModelica and Dymola and achieving the same simulation results with both tools.

With the help of these test cases, many potential failures and weak spots in the algorithms of the IFC import and data preparation module (e.g. incorrectly calculated geometries and topologies), the building data model (e.g. missing attributes) and the code generation template used in CoTeTo (e.g. required additional features for a more flexible code generation) could be detected, fixed, and improved.

2.6 Simulation experiment with JModelica

After the multi-zone building model code was generated with CoTeTo, a simulation experiment could be performed with a Modelica tool. Because the objective of the development of the toolchain was a pure open source solution, JModelica (http://JModelica.org) was used for this purpose. The definition of a Modelica simulation experiment in JModelica takes place in Python script, in which the three steps model translation, model simulation and, result visualization have to be defined. The JModelica compiler (Python module pymodelica) obtains the Modelica model over the method `compile_fmu()` and generates an executable FMU in version 1.0 or 2.0.

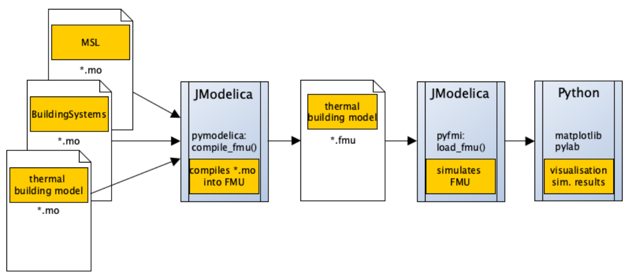


Figure 4. Compilation and simulation of the thermal building models with JModelica.

In the following step this FMU is taken by the JModelica run time system (Python module pyfmi) by using `mymodel`, an instance of the Python class `FMUModelBase`. This instance is generated as the return value of the function call `load_fmu()`. The two methods calls `myModel.simulate_options()` and `myModel.simulate()` configure the numerical options and start the simulation experiment. The simulation results are stored in a Python dictionary and visualized with a suitable graphical Python library such as matplotlib or pylab after the simulation experiment is performed (compare with Figure 4 and the following excerpt of a Python script, which defines the simulation experiment):

```
# compile model to fmu
from pymodelica import compile_fmu
fmu = compile_fmu('MultiZoneBuilding',…)

# load the fmu
from pyfmi import load_fmu
myModel = load_fmu(fmu)

# simulate the fmu and store results
opts = myModel.simulate_options()
opts['solver'] = "CVode"
opts['ncp'] = 240
res = myModel.simulate(start_time=0.0,
final_time=864000, options=opts)

# plotting of the results
import pylab as P
fig = P.figure(1)
y1 = res['ambient.TAirRef']
y2 = res['building.TAir[1]’]
y3 = res['building.TAir[5]’]
y4 = res['building.TAir[12]’]
t = res['time']
P.subplot(2,1,1)
P.plot(t,y1,t,y2,t,y3,t,y4)
P.legend(['ambient.TAirRef','building.TAir
[1]'],…)
P.ylabel('Temperature (K)')
P.xlabel('Time (s)')
P.show()
```

3 Case study

The described approach of the toolchain was evaluated by the example of a small residential living unit, the Rooftop building, which was developed for the Solar Decathlon Europe 2014 (SDE 2014) in Versailles, France (http://www.solardecathlon2014.fr/en/) by a student team from UdK Berlin and TU Berlin (see Figure 5).

Figure 5. The realized prototype of the Rooftop building on the SDE 2014 competition site in Versailles, France.

This rooftop construction was designed as a solar plus energy living unit, which can be placed on top of the building stock (compare with Figure 6) and can be air-conditioned and supplied by its own gained energy all the year around. A detailed description of the Rooftop building incl. the technologies used (reversible heat pump, adaptable photovoltaic facades, thermal and electrical storage management etc.) can be found in Team Rooftop (2014).

Figure 6. The concept of the Rooftop building as a solar living unit for the building stock for dense city districts.

The Rooftop building was modelled for the case study as a 3D BIM model in ArchiCAD. Starting from this information base, an IFC2X3 model was exported. Figure 7 shows the visualization of this IFC model with all construction elements, and Figure 8 shows only the part of the building model which is relevant to a building energy simulation.

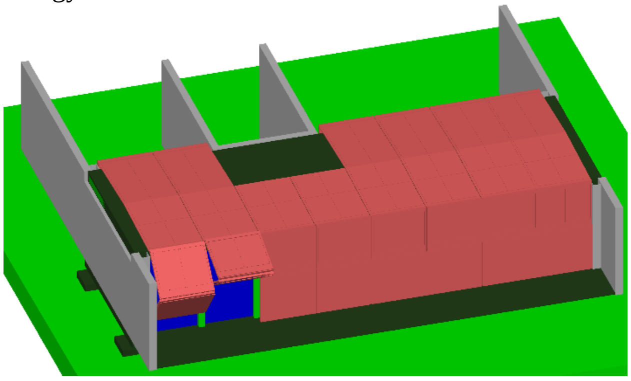


Figure 7. BIM model Rooftop building with all building elements.

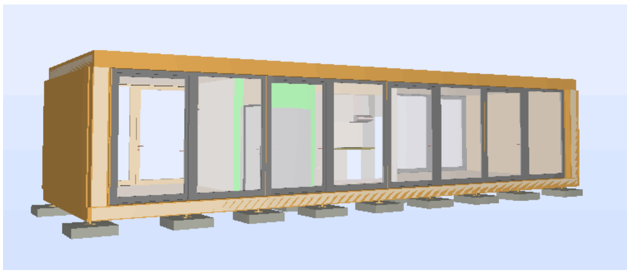


Figure 8. BIM model of the Rooftop building, reduced to the relevant building elements for building energy simulation.

The inner building structure includes four thermal zones, two of which can be air-conditioned by a floor heating and a cooling ceiling system (zones lab and seminar). The other two are only thermal buffer zones with free floating temperatures (zones toilet and core). The building construction consists of wooden lightweight building elements in combination with large glass facades.

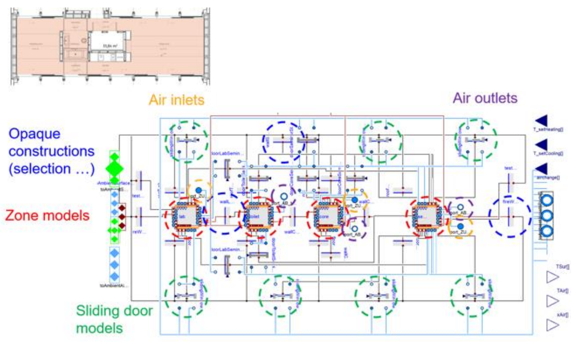


Figure 9. Generated Modelica building model.

At present, the Modelica code generation is restricted to the thermal building model (see Figure 9); the HVAC system of the building energy system still has to be configured by hand.

The generated Modelica model of the Rooftop building was simulated with JModelica for a period of four hot summer days for the location Berlin. In Figure 10, the outside air temperature and the free-floating air temperatures of the four thermal zones are illustrated. Because the air change of the generated building model is suppressed and the large transparent facades are unshaded in the configuration of the simulation experiment, the air temperatures in the seminar zone and the lab zone show the typical increasing overheating behavior of a "glass house" over the time.

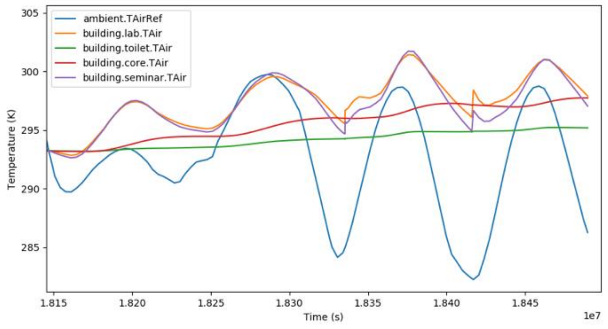


Figure 10. Simulated indoor climate of the Rooftop building during four warm summer days (location Berlin).

4 Summary and Outlook

A Python-based open source toolchain for generating thermal multi-zone building models from BIM models for the Modelica BuildingSystems library was successfully implemented, validated, and evaluated by means of the case study of the Rooftop building. Up to now, the code generation process is limited to the building construction; the HVAC system of the building has to be manually added.

The source code of the BIM2Modelica toolchain incl. the code generation tool CoTeTo, the Modelica BuildingSystems library, and a set of BIM test cases is available for free and can be downloaded as one software package from GitHub (https://github.com/UdK-VPT/BIM2Modelica).

The future development of the toolchain will take place in collaboration with research partners within the IBPSA project 1 (https://ibpsa.github.io/project1) in work package 2.2 "Building Information Modeling".

Future developments of the BIM2Modelica toolchain will focus on automatic reduction of the building model complexity dependent on the given boundary conditions (orientation of façade elements, conditions of use for the zones), similar as described in Reynders et al. (2017).

Further, additional specialized CoTeTo templates for C# code generation that supports a building model visualization for Unity (https://unity3d.com/de) are under development (see also Nytsch-Geusen et al., 2017).

Acknowledgements

The research described in this paper was conducted within the research project "EnEff BIM: Planung, Auslegung und Betriebsoptimierung von energieeffizienten Neu- und Bestandsbauten durch Modellierung und Simulation auf Basis von Bauwerkinformationsmodellen" and funded by the Federal Ministry for Economic Affairs and Energy in Germany (reference: 03ET1177D).

References

Bazjanac, V. (2017). Testing space boundaries that transcribe complex CAD building geometry into surface geometry usable by EnergyPlus and similar building energy performance simulation engines. Internal report, UdK Berlin, Germany.

IfcOpenShell (2019). The open source IFC toolkit and geometry engine - http://ifcopenshell.org/python.html (last access 2019 Jan 21)

Jorissen, F, Reynders, R.; Baetens, R.; Picard, D.; Saelens, D. and Helsen, L. (2018). Implementation and Verification of the IDEAS Building Energy Simulation Library. Journal of Building Performance Simulation, 11 (6), 669-688, doi: 10.1080/19401493.2018.1428361.

Mako (2019). Mako templates for Python - http://www.makotemplates.org (last access 2019 Jan 21)

Modelica IBPSA library (2019) - https://github.com/ibpsa/modelica-ibpsa (last access 2019 Jan 21).

Nytsch-Geusen, C.; Banhardt, C.; Inderfurth. A.; Mucha, K.Möckel, Jens; R., Jörg; Thorade, M.; Tugores, C. (2016). BuildingSystems – Eine modular hierarchische Modell-Bibliothek zur energetischen Gebäude- und Anlagensimulation. BAUSIM 2016 IBPSA. Conference Proceedings, Dresden, Germany.

Nytsch-Geusen, C.; Inderfurth, A.; Kaul, W.; Mucha, K.; Rädler, J.; Thorade, M. and Tugores, C.R. (2017). Template based code generation of Modelica building energy simulation models. 12th International Modelica Conference, Conference Proceedings, Prag, Czechia.

O'Donnell, J.; See, R.; Rose, C.; Maile, T., Bazjanac, V. and Haves, P. (2011). SimModel: A domain data model for whole building energy simulation. In Proceedings of the 12th IBPSA Building Simulation Conference, Sydney, Australia.

pythonOCC (2019). pythonOCC – 3D CAD for python - http://www.pythonocc.org (last access 2019 Jan 21).

Reynders, G.; Andriamamonjy, A.; Klein, R; Saelens, D. 2017 Towards an IFC-Modelica tool facilitating model complexity selection for building energy simulation (2017). 15th IBPSA Building Simulation Conference, Conference Proceedings, San Francisco, USA.

Team Rooftop (2014), Deliverable 6 & 7 of the Solar Decathlon Europe 2014. Official documentation of the Rooftop project. UdK Berlin and TU Berlin, Germany.

Thorade, M.; Rädler, J.; Remmen, P.; Maile, T.; Wimmer, R.; Cao, J; Lauster, M.; Nytsch-Geusen, C.; Müller, D. and van Treeck, C. (2015) An open toolchain for generating Modelica code from Building Information Models. 11th International Modelica Conference, Conference Proceedings, Versailles, France.

Wetter, M.; Fuchs, M.; Grozman, P.; Helsen, L., Jorissen, F.; Lauster, M.; Müller, D.; Nytsch-Geusen, C.; Picard, D.; Sahlin, P.; and Thorade, M. (2015). IEA EBC Annex 60 Modelica Library - An international collaboration to develop a free open-source model library for buildings and community energy systems. 14th IBPSA Building Simulation Conference, Conference Proceedings, Hyderabad, India.

SESSION 1B: POWER & ENERGY 1

Open Source PhotoVoltaics Library for Systemic Investigations

Jovan Brkic[1] Muaz Ceran[1] Mohamed Elmoghazy[1] Ramazan Kavlak[1]
Anton Haumer[2] Christian Kral[1]
[1]TGM Wien XX, College of Engineering, Austria, dr.christian.kral@gmail.com
[2]OTH Regensburg, Germany, anton.haumer@oth-regensburg.de

Abstract

For the planning of photovoltaic power plants standard software tools are used. Most of these software tools use statistical solar data to determine the overall energy harvest of a photovoltaic plant over one year. The calculations rely on stationary location and ideal boundary conditions, e.g., constant ambient temperature. Even though, for example, shadowing may be considered by standard software, the investigation of untypical configurations and problems cannot be performed by such software, as most configurations cannot be changed by the user.

The presented PhotoVoltaics library was developed with the intention to provide a flexible framework for standard and non-standard problems. Particularly, the PhotoVoltaics library can be coupled with other Modelica libraries to perform systemic investigations. An application library, PhotoVoltaics_TGM, is provided as add-on, where measured data of two photovoltaic pants of the TGM in Vienna can be compared with simulation results. This add-on library serves as validation of the PhotoVoltaics library.

Keywords: Photovoltaics, cell, module, plant, data sheet, converter, maximum power tracking, irradiance, terrestrial solar model

1 Introduction

For academic and scientific investigations and calculations in the engineering field of photovoltaics an open source library is advantageous. Currently, some Modelica libraries exist which provide photovoltaic plans on different levels of abstractions.

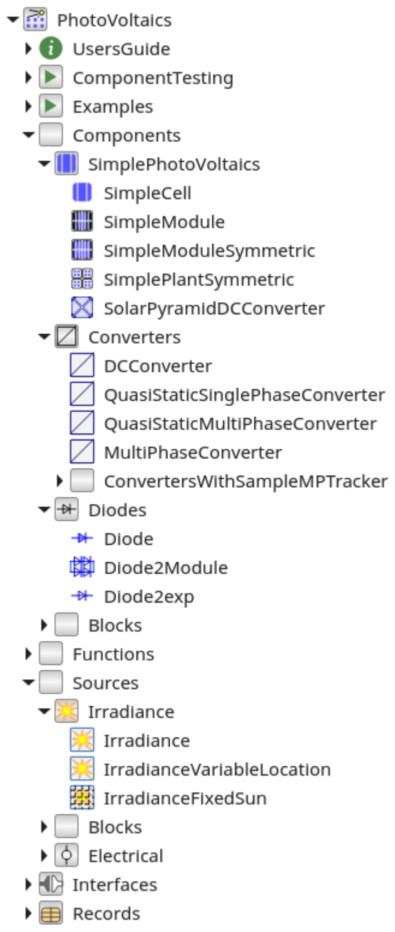
The Buildings library (Wetter, 2017) includes photovoltaic plant models based area of cross section and efficiency parameters. The plant models evaluate the irradiance input and calculate the harvested power by means of a non-standard electrical connector. Additionally, the Buildings library provides blocks for the processing of public irradiance data available from EnergyPlus.

In the BuildSysPro library a photovoltaic model is provided based on physical material and geometry parameters, but not electrical parameters (BuildSysPro, 2017). Electric power is represented by a signal output, so the actual interaction with maximum power control and the power grid cannot be modeled in a physical way.

The PVSystems library relies on manufacturer data and uses electrical connectors from the Modelica Standard library (PVSystems, 2017). The photovoltaics model includes a series and parallel resistor but unfortunately no parameterization aid is provided to determine the resistance parameters. Temperature dependency is considered by means of a temperature signal input. Even though the PVSystems library relies on a roughly similar physical modeling approach as the PhotoVoltaics library, the model behavior is not consistent. The investigation of the open circuit voltage at standard conditions and a slightly higher temperature shows simulations results which clearly deviate from the data sheet parameters.

The PhotoVoltaics library was developed during a Diploma project at the College of Engineering, TGM, in 2016--17. It is available on GitHub (Kral, 2017). The main target of this library was to provide physical models of photovoltaic components that show consistent behavior and can be parameterized solely on data sheet values. So the open circuit voltage and short circuit current including their temperature dependencies were intended to be modeled to fully match the given data sheet values.

The paper presents the structure of the provided libraries in Section 2. In Section 3 the data sheet parameters are presented and explained. Based on these parameters, the included cell, module and plant model are elaborated in Section 4. Different converters and the maximum power tracking are explained in Sections 5 and 6. The irradiance models of Section 7 are needed for the system investigations of complex photovoltaic systems. The library validation and further application examples are presented in Sections 8 and 9.



- PhotoVoltaics
  - UsersGuide
  - ComponentTesting
  - Examples
  - Components
    - SimplePhotoVoltaics
      - SimpleCell
      - SimpleModule
      - SimpleModuleSymmetric
      - SimplePlantSymmetric
      - SolarPyramidDCConverter
    - Converters
      - DCConverter
      - QuasiStaticSinglePhaseConverter
      - QuasiStaticMultiPhaseConverter
      - MultiPhaseConverter
      - ConvertersWithSampleMPTracker
    - Diodes
      - Diode
      - Diode2Module
      - Diode2exp
    - Blocks
  - Functions
  - Sources
    - Irradiance
      - Irradiance
      - IrradianceVariableLocation
      - IrradianceFixedSun
    - Blocks
    - Electrical
  - Interfaces
  - Records

Figure 1. Structure of the `PhotoVoltaics` library

2 Library Structure

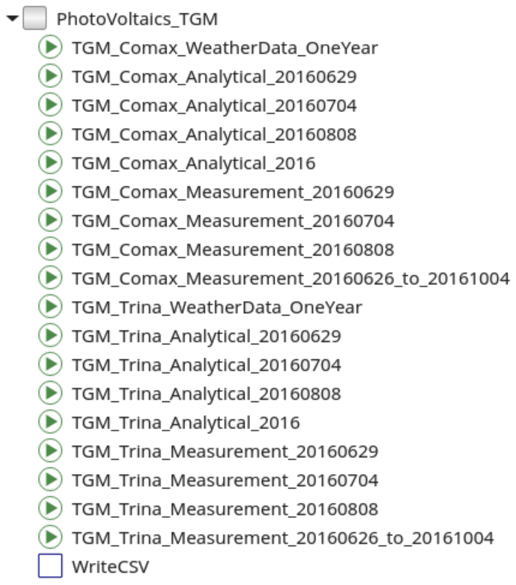
The `PhotoVoltaics` library includes:

- Photovoltaic (PV) components (cells, modules and plants)
- Converters (DC/DC, quasi static single and three phase, transient three phase)
- Diodes
- Analytic irradiance models (terrestrial, arbitrary sun location)
- Records of selected industrial module data sheets

The structure of the library is depicted in Figure 1. General configuration examples are included in the package `Examples`. Additional application examples which allow the validation of the proposed models are provided in the external package `PhotoVoltaics_TGM` shown in Figure 2.

3 Data Sheet Parameters

Most data sheet parameters refer to standard conditions (STC) of a module. These conditions are characterized by



- PhotoVoltaics_TGM
  - TGM_Comax_WeatherData_OneYear
  - TGM_Comax_Analytical_20160629
  - TGM_Comax_Analytical_20160704
  - TGM_Comax_Analytical_20160808
  - TGM_Comax_Analytical_2016
  - TGM_Comax_Measurement_20160629
  - TGM_Comax_Measurement_20160704
  - TGM_Comax_Measurement_20160808
  - TGM_Comax_Measurement_20160626_to_20161004
  - TGM_Trina_WeatherData_OneYear
  - TGM_Trina_Analytical_20160629
  - TGM_Trina_Analytical_20160704
  - TGM_Trina_Analytical_20160808
  - TGM_Trina_Analytical_2016
  - TGM_Trina_Measurement_20160629
  - TGM_Trina_Measurement_20160704
  - TGM_Trina_Measurement_20160808
  - TGM_Trina_Measurement_20160626_to_20161004
  - WriteCSV

Figure 2. Structure of the `PhotoVoltaics_TGM` library, including application and validation examples

the reference temperature $T_{\mathrm{ref}} = 25\,^{\circ}\mathrm{C}$ and the reference irradiance $\texttt{irradiance}_{\mathrm{ref}} = 1000\,\mathrm{W/m^2}$. Under these reference conditions the following quantities are listed:

- Open circuit voltage $V_{\mathrm{oc,ref}}$ under reference conditions
- Short circuit current $I_{\mathrm{sc,ref}}$ under reference conditions
- Maximum power voltage $V_{\mathrm{mp,ref}}$ under reference conditions
- Maximum power current $I_{\mathrm{mp,ref}}$ under reference conditions
- Linear temperature coefficient of open circuit voltage $\alpha_{V\mathrm{oc,ref}}$ at reference conditions
- Linear temperature coefficient of short circuit current $\alpha_{I\mathrm{sc,ref}}$ at reference conditions

Typically, the temperature coefficient of maximum power is also listed in the data sheet of a photovoltaic module. This parameter is, however, not evaluated in the proposed model, since the model inherently considers the maximum power temperature coefficient from the open and short circuit temperature coefficients.

Each module is usually equipped with `nb` bypass diodes. These diodes are usually connected anti parallel to two or more strings of the photovoltaic cells. The diodes are used to overcome reverse operating conditions caused by partial shading of the module. In order to consider the bypass diodes properly in the model, the parameters `BvCell`, `Ibv` and `Nbv` listed in Listing 1 are taken into account.

Photovoltaic modules usually consist of `ns` series connected cells, but have no parallel connected cells. The cell parameters are determined from the module parameters as shown in Listing. 1. Additionally, the temperature voltage at reference temperature,

$$V_{t,\mathrm{ref}} = \frac{k \cdot T_{\mathrm{ref}}}{Q}, \tag{1}$$

is calculated as final parameter. In this equation k is the Boltzmann constant and Q is the elementary charge (of an electron). The module parameters are organized as record in Modelica.

Listing 1. Record of module data

```
record ModuleData "Data of PV module"
  extends Modelica.Icons.Record;
  import SI=Modelica.SIunits;
  parameter String moduleName = "Generic";
  parameter SI.Temperature TRef = 298.15
    "Reference temperature";
  parameter SI.Irradiance irradianceRef
    = 1000 "Reference solar irradiance";
  parameter SI.Voltage VocRef = 30.2
    "Reference open circuit module voltage
    > 0 at TRref";
  final parameter SI.Voltage
    VocCellRef = VocRef / ns "Reference open
    circuit cell voltage > 0 at TRref";
  parameter SI.Current IscRef= 8.54
   "Reference short circuit current
    > 0 at TRref and irradianceRef";
  parameter SI.Voltage VmpRef = 24.0
    "Reference maximum power module
    voltage > 0 at TRref";
  final parameter SI.Voltage VmpCellRef
    = VmpRef / ns "Reference maximum power
    cell voltage > 0 at TRref";
  parameter SI.Current ImpRef = 7.71
    "Reference maximum power current
    > 0 at TRref and irradianceRef";
  parameter SI.LinearTemperatureCoefficient
    alphaIsc = +0.00053 "Temperature
    coefficient of reference short circuit
    current at TRref";
  parameter SI.LinearTemperatureCoefficient
    alphaVoc = -0.00340 "Temperature
    coefficient of reference open circuit
    module voltage at TRref";
  parameter Integer ns = 1
    "Number of series connected cells";
  parameter Integer nb = 1
    "Number of bypass diodes per module";
  parameter SI.Voltage BvCell = 18
    "Breakthrough cell voltage";
  parameter SI.Current Ibv = 1
    "Breakthrough knee current";
  parameter Real Nbv = 0.74
    "Breakthrough emission coefficient";
  final parameter SI.Voltage VtCellRef
    = Modelica.Constants.k * TRef / Q
    "Reference temperature voltage of cell";
  constant SI.Charge Q = 1.6021766208E-19
    "Elementary charge of electron";
end ModuleData;
```

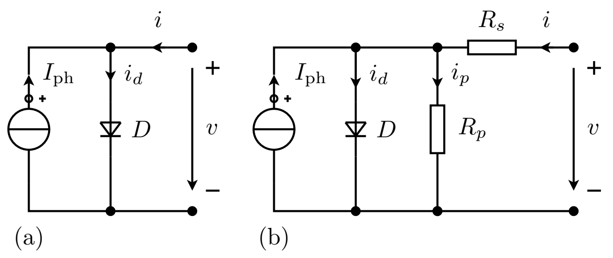


Figure 3. (a) Basic and (b) extended equivalent circuit diagram of PV cell

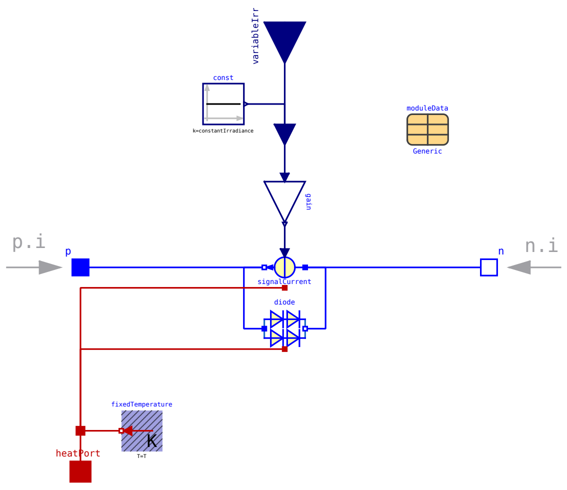


Figure 4. Modelica implementation of the cell model `SimpleCell`

4 PhotoVoltaics Components

4.1 Photo Voltaic Cells

The basic photovoltaic component is the cell. The implemented model consists of a diode and current source as shown in Figure 3 (Mahmoud et al., 2012). The current source represents the solar power source and the diode includes the semiconductor properties of a cell. The basic equivalent circuit diagram of Figure 3(a) could be extended by a series resistor R_s and a parallel resistor R_p. The extended model may be more accurate than the basic model, but from data sheet values the extended model can usually not be parameterized as the slopes of the voltage versus current characteristics are not provided accurately enough by the manufacturers. Therefore, the `PhotoVoltaics` library includes only basic models, so far. The actual implementation of the cell model `SimpleCell` is depicted in Figure 4.

In the basic photovoltaic cell model the source current I_{ph} is modeled directly proportional to the actual irradiance the cell is exposed to, including temperature dependence

$$\frac{I_{\mathrm{ph}}}{I_{\mathrm{ph,ref}}} = \frac{\texttt{irradiance}}{\texttt{irradiance}_{\mathrm{ref}}} + \alpha_{I\mathrm{sc}}(T - T_{\mathrm{ref}}), \tag{2}$$

where T is the actual operating temperature of the current source. The actual irradiance can be considered in two

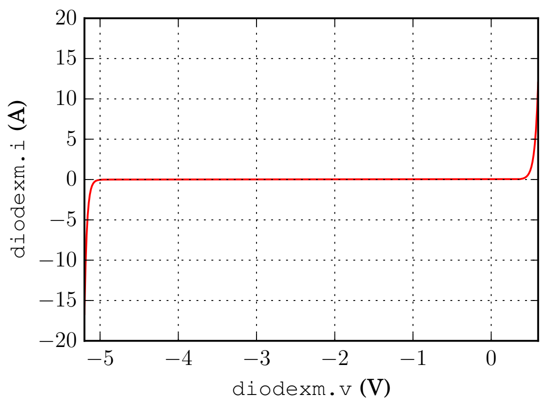


Figure 5. Diode model with two exponential regions and one linear region

ways:

- A constant quantity `constantIrradiance` is provided as parameter if the boolean parameter `useConstantIrradiance = true`
- A signal input `variableIrradiance` is enabled if the boolean parameter `useConstantIrradiance = false`

The variable `irradiance` in (2) is thus assigned to either `constantIrradiance` or `variableIrradiance` depending on the boolean parameter `useConstantIrradiance`.

The diode models is especially designed for the purpose of photovoltaic applications. It covers the forwards and backwards breakthrough region. For numerical reasons the current versus voltage characteristic of the used diode model considers three different regions:

- Forward exponential range for positive cell voltages,

$$i = I_{ds}\left(\exp\left(\frac{v}{mV_t}\right) - 1\right) + \frac{v}{R}, \qquad (3)$$

  where $R = 10^8\,\Omega$ is a parallel resistance used to stabilize the model numerically; I_{ds} represents the saturation current m is the ideality factor of the diode
- Backward linear range in the reverse direction starting from zero voltage
- Backward exponential range of negative voltages in the breakthrough region

The current versus voltage characteristic of the implemented diode model depicted in Figure 5 is determined by the experiment `DiodeCompare` using `ns = 1`, `nsModule = 1` and `npModule = 1`. The variables I_{ds} and m of the diode forward exponential region (3) are determined through the operating conditions of the photovoltaic cell. The first condition is derived from the open circuit case of Figure 3(a) substituted in (3),

$$I_{\mathrm{sc}} = I_{ds}\left(\exp\left(\frac{V_{\mathrm{oc}}}{mV_t}\right) - 1\right). \qquad (4)$$

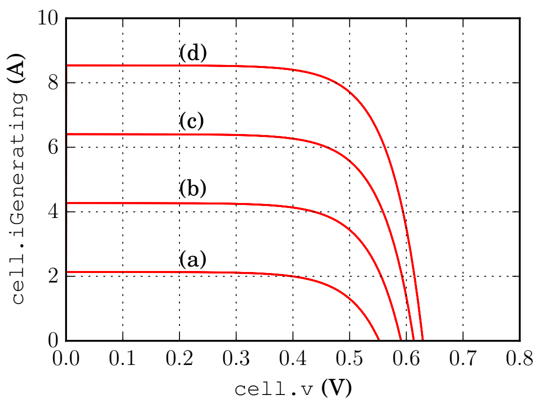


Figure 6. Cell current versus cell voltage for (a) `irradiance` $= 250\,\mathrm{W/m^2}$, (b) `irradiance` $= 500\,\mathrm{W/m^2}$, (c) `irradiance` $= 750\,\mathrm{W/m^2}$, and (d) `irradiance` $= 1000\,\mathrm{W/m^2}$

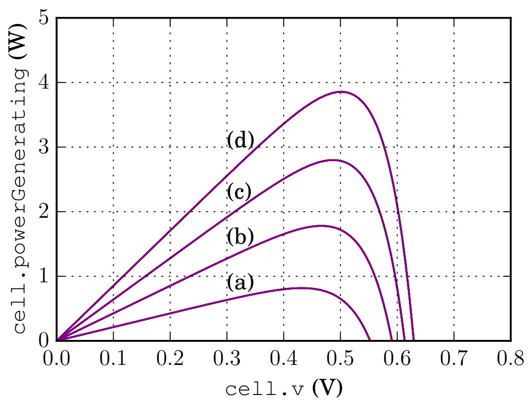


Figure 7. Cell power versus cell voltage for (a) `irradiance` $= 250\,\mathrm{W/m^2}$, (b) `irradiance` $= 500\,\mathrm{W/m^2}$, (c) `irradiance` $= 750\,\mathrm{W/m^2}$, and (d) `irradiance` $= 1000\,\mathrm{W/m^2}$

In this equation the photo current is equal to the short circuit current. The impact of the parallel resistance on this equation is neglected. The second conditions is determined by the temperature voltage

$$V_t = \frac{k \cdot T}{Q}. \qquad (5)$$

Additionally the temperature dependencies of the open circuit and short circuit voltage of the photovoltaic cell have to be taken into account:

$$V_{\mathrm{oc}} = V_{\mathrm{oc,ref}}\left(1 + \alpha_{V\mathrm{oc}}(T - T_{\mathrm{ref}})\right) \qquad (6)$$
$$I_{\mathrm{sc}} = I_{\mathrm{sc,ref}}\left(1 + \alpha_{I\mathrm{oc}}(T - T_{\mathrm{ref}})\right) \qquad (7)$$

This implementation causes the saturation current I_{ds} and the ideality factor m to be temperature dependent. The variability of these quantities is a result of consistent operating conditions of the proposed model based on manufacturer data.

The linear scaling of the short circuit current according to (2) for constant temperature is demonstrated in Figure 6. Figure 7 shows the dependence of the maximum

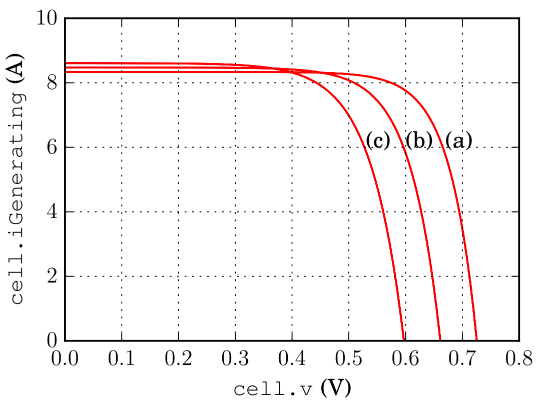


Figure 8. Cell current versus cell voltage for (a) $T = -20\,^\circ\mathrm{C}$, (b) $T = 10\,^\circ\mathrm{C}$, (c) $T = 40\,^\circ\mathrm{C}$

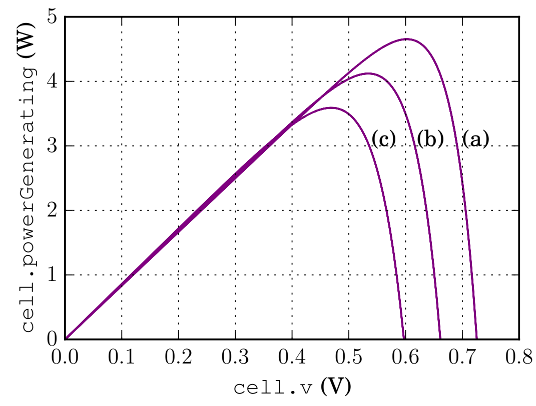


Figure 9. Cell power versus cell voltage for (a) $T = -20\,^\circ\mathrm{C}$, (b) $T = 10\,^\circ\mathrm{C}$, (c) $T = 40\,^\circ\mathrm{C}$

power point on irradiance. The impact of the operating temperature on the short circuit current and the open circuit voltage is shown in Figure 8. In this figure a typical case is evaluated, considering a positive temperature coefficient of the short circuit current and a negative temperature coefficient of the open circuit voltage. Since the absolute value of the temperature coefficient of the open circuit is greater than the temperature coefficient of the short circuit current, the temperature coefficient of the maximum power is negative. Consequently, the maximum power harvest of a photovoltaic cell decreases with increasing temperature, see Figure 9.

The photovoltaic cell model also considers shading of a cell. In this implementation `shadow = 0` represents the case of full exposure to solar irradiance, whereas `shadow = 1` considers zero irradiance. In addition to conventional shading caused by visible obstacles, this approach also allows the consideration of the dimming of the cell over time due the impact of pollution (Häberlin and Renken, 1999; Renken and Häberlin, 1999).

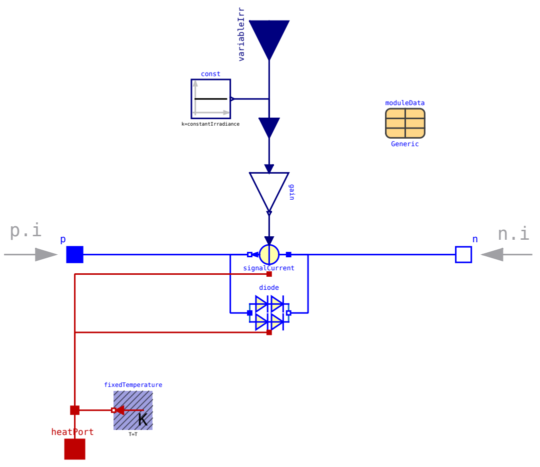


Figure 10. Modelica implementation of the symmetric module model `SimpleModuleSymmetric`

4.2 Symmetric Photovoltaic Modules

In the `PhotoVoltaics` library two different module classes are provided. The symmetric module assumes uniform shading of all cells of a module. In this case the bypass diodes are not taken into account.

The cell currents `iCell` and the cell voltages `vCell` correspond with the module current `i` and the module voltage `v` by

$$\mathtt{iCell} = \mathtt{i}, \tag{8}$$

$$\mathtt{ns} * \mathtt{vCell} = \mathtt{v}. \tag{9}$$

In (8) it is considered that a module has no parallel connections. However, both the current source and the diode model are designed such way that parallel and series connections may be considered by scaling the photo current and the diode voltage and current, respectively. Therefore, the `SimpleModuleSymmetric` model of Figure 10 looks similar to the cell model of Figure 4.

4.3 Asymmetric Photo Voltaic Module

The asymmetric photovoltaic module consists of a physical series connection of cells (Figure 11). For each cell the corresponding shading can be adjusted by the module array parameter `shadow`. The division of the number of series connected cells by the number of bypass diodes, `ns/nb`, has to have zero remainder in order to model the bypass diodes symmetrically.

4.4 Symmetric Plant

The symmetric plant model is designed in the spirit of a symmetric module model as shown in Figure 10. For the plant model the number of series and parallel connected modules, `nsModule` and `npModule` are considered. The plant current `i` and the plant voltage `v` corre-

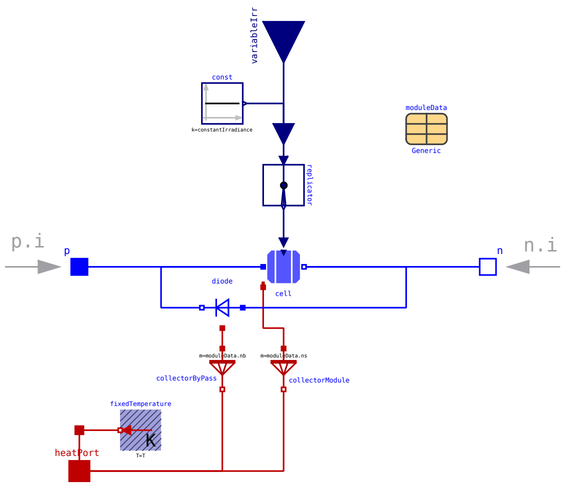
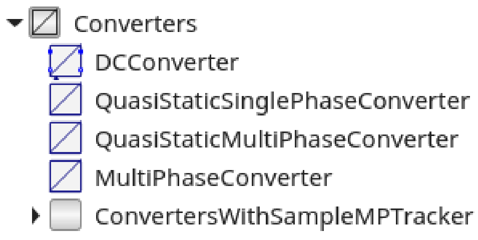


Figure 11. Modelica implementation of the module model `SimpleModule`



- Converters
  - DCConverter
  - QuasiStaticSinglePhaseConverter
  - QuasiStaticMultiPhaseConverter
  - MultiPhaseConverter
  - ConvertersWithSampleMPTracker

Figure 12. Converter models of the PhotoVoltaics Library

spond with module and cell currents and voltages by:

$$\texttt{npPlant} * \texttt{iModule} = \texttt{i} \tag{10}$$

$$\texttt{iCell} = \texttt{iModule} \tag{11}$$

$$\texttt{nsPlant} * \texttt{vModule} = \texttt{v} \tag{12}$$

$$\texttt{ns} * \texttt{vCell} = \texttt{vModule} \tag{13}$$

5 Converters

The power conversion from a photovoltaic cell, module or plant to a DC, single or three phase AC grid is performed by means of converter models, see Figure 12. The converter models are all designed the same way considering the following characteristic behavior:

- Neither conduction nor switching losses are taken into account
- The converter models rely on ideal power conversion
- The voltage of the photovoltaic DC side can be adjusted (controlled) by a signal input

For the DC/DC conversion only, a converter model with integrated maximum power tracker is provided.

6 Maximum Power Tracking

In order to harvest the maximum energy, a photovoltaic plant has to be operated in the point of maximum power, see Figures 7 and 9. The photovoltaic DC side voltage has to be controlled such way that the maximum power point is reached. There are various maximum power tracking methods available in the literature. For the sake of simplicity there is only one discrete maximum power tracker implemented so far.

The implemented maximum power tracker is a block which evaluates the sensed power through a signal input. The output is the controlled photovoltaic DC voltage. The tracker samples the input power with a fixed sampling period. The output voltage is permanently changing in order to always follow changes of the maximum power point.

The initial setting is the voltage of the maximum power point according to the data sheet of the used modules. The output voltage gets increase by a voltage increment $\Delta v \cdot s$, where $s = -1$, so the output voltage actually gets decreased. If the sensed power is greater than the previously sensed power, the voltage will be decreased again. This procedure is performed until the actually sampled power gets smaller than the previously sampled power. In that case the sign s will be altered. Then the output voltage starts increasing, again until the maximum power point is exceeded. This way, under steady state thermal and solar conditions, the output voltages is continuously changed between three different stages.

The experiment `SimpleModuleMP` investigates the maximum power tracking under varying irradiance conditions and a sampling time of one second. The initial irradiance is equal to $200\,\mathrm{W/m^2}$. From 100 to 200 seconds the irradiance increases linearly up to $1000\,\mathrm{W/m^2}$ and remains constant until 300 seconds. Figure 13 and 14 show the power and reference voltage versus time, respectively. The reference voltage starts with the voltage of the maximum power point $V_{\mathrm{mp,ref}} = 24\,\mathrm{V}$, which refers to $1000\,\mathrm{W/m^2}$. However, since the experiment starts with $200\,\mathrm{W/m^2}$, the reference voltage is decreased to roughly $20\,\mathrm{V}$ in order to reach the actual maximum power point of approximately $30\,\mathrm{W}$ under these conditions. After increasing the irradiance to $1000\,\mathrm{W/m^2}$ the reference voltage is controlled up to $24\,\mathrm{V}$ which is equal to the expected $V_{\mathrm{mp,ref}} = 24\,\mathrm{V}$. From the voltage waveform the operating behavior of the controller can be observed. In quasi static operation the reference voltage is controlled upwards and downwards by Δv, having the maximum power point in between. The nonlinear increase of the reference voltage between 200 and 300 seconds is due to the fact that the irradiance and the maximum power of the module are not related linearly. Even though the power curve appears smooth in Figure 13, is also reveals discrete power changes when zooming into the curve.

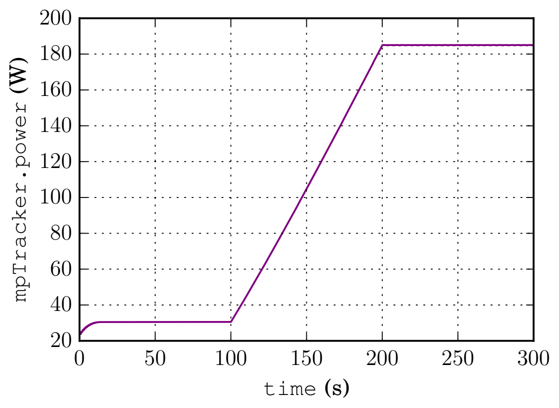


Figure 13. Example `SimpleModuleMP`, power versus time

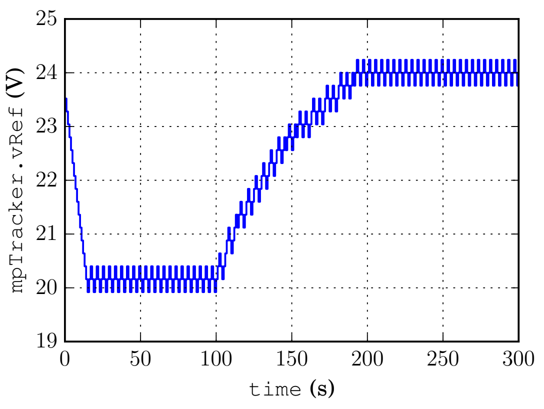


Figure 14. Example `SimpleModuleMP`, reference voltage versus time

7 Irradiance Models

In the `PhotoVoltaics` library terrestrial irradiance models are provided. The calculations are based on (Quaschning, 2011). The used equations are not discussed in this paper. However, the basic idea was to provide an analytic solar model that calculates the irradiance from the following parameters:

- Start day
- Start month
- Start year
- Time zone
- Longitude
- Latitude
- Reference irradiance (default value is $1000\,\mathrm{W/m^2}$)
- Angle of inclination, γ, of the photovoltaic module with respect to the horizontal plane
- Azimuth of the photovoltaic module orientation (N = $0\,°$, E = $90\,°$, S = $180\,°$, W = $270\,°$, see Figure 21(c))

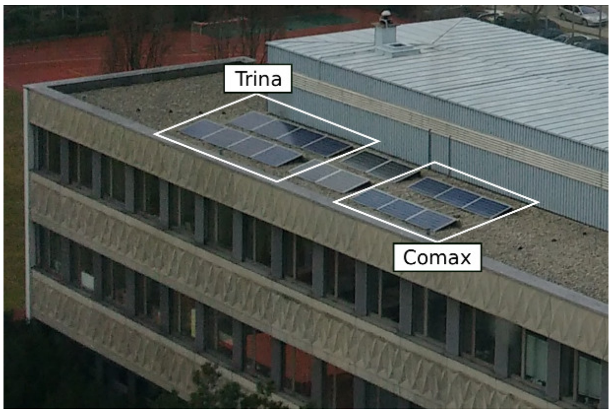


Figure 15. Photovoltaic plants of the TGM in Vienna

The start time of a simulation experiment refers to local time 00:00 of the indicated start day (and month and year). In this model, the following effects are not considered:

- Reflection
- Diffusion
- Visible obstacles in the vicinity of the photovoltaic plant

The purpose of the analytic solar model is to provide a basis for systemic investigations without having real weather conditions distorting the virtual experiments.

Since the `Buildings` library (Wetter, 2017) provides models to determine the effective irradiance based on statistical weather data, there are no extra models included in the `PhotoVoltaics` library to serve this purpose. Open access weather data of Vienna, Austria are included in the `PhotoVoltaics` library (EnergyPlus, 2017).

8 Validation

The `PhotoVoltaics_TGM` library is an additional library dedicated to the comparison of the simulation and measurement data of two photovoltaic plants located at the College of Engineering, TGM, in Vienna, Austria (Figure 15). The library relies on the `PhotoVoltaics` and the `Buildings` library (Wetter, 2017). The investigated plants are named after the manufacturer of the modules, Trina and Comax; see (Trina, 2017) and (Comax, 2017). The two plants are equipped with one irradiance sensor which allows the validation of the proposes cell and module models of the PhotoVoltaics library. For the two different plants one validation example, each, will be presented in this paper. The modules are aligned with the building, the direction is roughly south. The angle of inclination equals $10\,°$.

For the validation of the `PhotoVoltaics_TGM` library a constant module temperature of $25\,°\mathrm{C}$ is used. If necessary, measured or modeled temperature data could be fed to the thermal connector of the module.

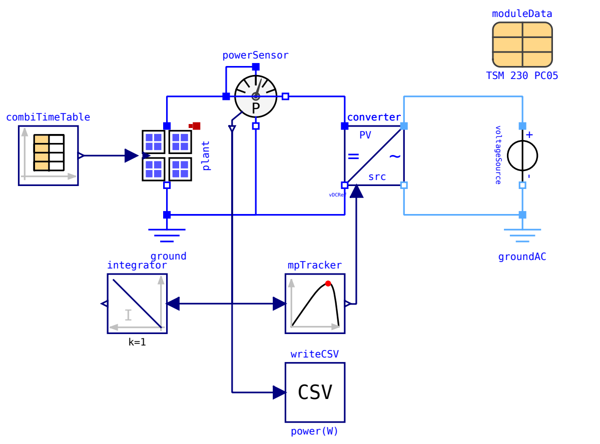


Figure 16. Model of the Trina plant of TGM in Vienna, reading irradiance data from a file

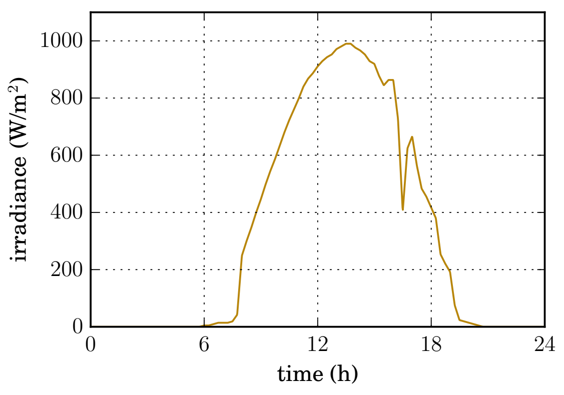


Figure 17. Measured irradiance at TGM in Vienna on 2016-06-29

The first example, shown in Figure 16, refers to the Trina plant on 2016-06-29. The measured irradiance data are depicted in Figure 17 are fed to the irradiance input connector. A shadow of 0.1 is taken into account in the model, to consider the degradation due to pollution (Häberlin and Renken, 1999). The generated DC power is fed to the AC power grid by means a quasi static DC/AC converter including power tracking. The simulated power is also integrated to determine the energy harvest of this day. An additional interface block is used to write time versus power data directly to a CSV file to simplify the evaluation with a spreadsheet processing software.

The irradiance waveform of Figure 17 shows one significant drop at 16:30 caused by the shadow of the highriser school building, which is located next to the photovoltaic plants. The maximum irradiance reaches about $1000\,\mathrm{W/m^2}$ and occurs at 13:30.

Since the power generated by the plant is directly proportional to the irradiance, the simulated DC power in Figure 18 shows the same drop at 16:30 as the irradiance. This figure also shows the measured DC and AC power of the converter. The measured DC power shows a high congruence with the simulated DC power. The measured AC power is smaller than the DC power due to the loss of the DC/AC converter. In the simulation, the converter loss is not taken into account. Consequently, there is no equivalent quantity in the simulation to be compared to the measured AC power.

The second example investigates the Comax plant at the TGM in Vienna on 2016-06.29. This plant has a smaller peak power than the Trina plant. The simulated DC power and the measured AC power are shown in Figure 19. Unfortunately, the converter of the Comax plant does not measure the DC power. Therefore, only the measured AC power can be compared to the simulated DC power. The difference between the two curves is again caused by the converter loss. However, by comparing the Figures 18 and 19 it can be roughly estimated that measurement and simulation again show a good agreement, if similar converter losses for both cases are presumed.

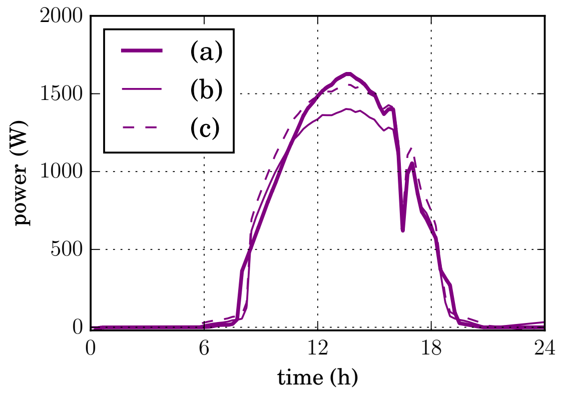


Figure 18. Power of the Trina plant at TGM in Vienna on 2016-06-29, (a) simulated DC power, (b) measured AC power and (c) measured DC power

The comparison of the measurement and simulation results of the two plants validates the presented `PhotoVoltaics` library.

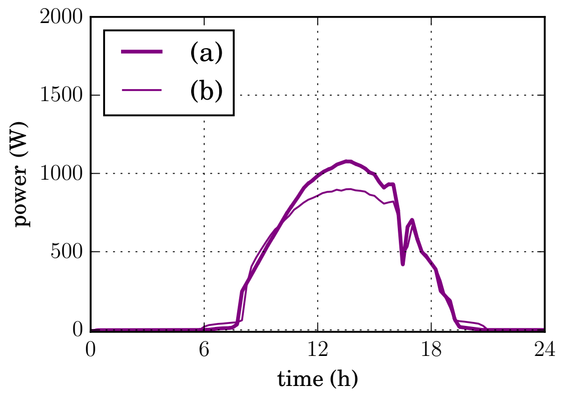


Figure 19. Power of the Comax plant at TGM in Vienna on 2016-06-29, (a) simulated DC power, (b) measured AC power and (c) measured DC power

9 Applications

Typical applications of the PhotoVoltaics library are systemic investigations which include photovoltaics. Since all photovoltaic components are equipped with a thermal heat port, the influence of temperature on the operational behavior may be investigated. Particularly, the library is capable of investigating of the total energy consumption and generation of alternative building concepts including interaction with the power grid.

One special application of the `PhotoVoltaics` library is the Phileas rover of the Austrian Space Forum (Austrian Space Forum, 2017). This rover is equipped with four triangularly shaped solar panels. The four panels are equipped with photovoltaic cells and shape a pyramid in the upright position as shown in Figure 20(a). At the top of the pyramid all four panels are mechanically connected. This top point can be moved vertically only. The remaining two bottom points of each face can only move in the horizontal plane as sketched in Figure 20(b). So, by vertically adjusting the top point of the four panels, the total energy harvest of the panel configuration can be changed. The actual panel configuration is characterized by the inclination angle γ of a panel as shown in Figure 20(a) and (b).

The Modelica model of the Phileas rover is depicted in Figure 21. If the position of one solar panel is known the position of other second panels is displaced by $90\,°$, and so on. Therefore, the input connectors of the model are the inclination angle γ and the azimuth of panel number 1. The inclination angles of the four panels are equal. The location of the sun is fixed in this model in order to allow systemic investigations. The irradiance of each of the four models is calculated and fed to the respective signal input of the panel. Each panel is connected with a DC/DC converter including maximum power tracker. The output sides of the four DC/DC converters are connected in parallel.

In the experiment `SolarPyramidBatteryCharge` a simplified investigation is made. The azimuths of the panels are kept constant and the inclination angle γ is varied between 0 and $60\,°$. The output connectors of the four parallel DC/DC converters are supplying a battery with constant voltage. The panel parameters are taken from a standard module. The actual geometric size of the four panels is not taken into account, since it cause a scaling of power only. An additional simplification of the experiment is done by using terrestrial solar irradiances instead of the irradiances on the Mars. The sun height is set to $22\,°$ and the sun azimuth is equal to $260\,°$ (see Figure 20(c)), i.e., the orientation of the sun is close to west. The four panels are oriented towards the main directions (panel 1 = north, panel 2 = east, panel 3 = south, panel 4 = west). The calculated powers of the four panels (1)–(4) and the sum power (Σ) are depicted in Figure 22. When increasing the inclination angle γ up to approximately $22\,°$, the power of solar panel number 2 drops to zero. This is a consequence of the inclination angle being greater than the sun height. The power of panel number 4 becomes the greatest, since it is oriented to west, close to the azimuth of the sun. All four panel powers start at the same starting power at $\gamma = 0\,°$, since they are then equally located in the plane. The total power shows a local maximum at $\gamma = 0\,°$ and a global maximum at $\gamma = 42\,°$.

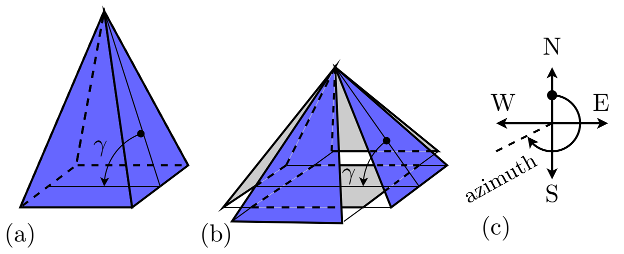


Figure 20. Solar panels of the Phileas rover (a) in most upright position and (b) in inclined position; (c) definition of the azimuth

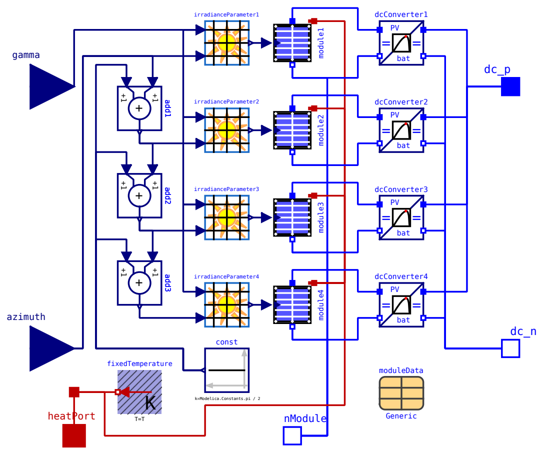


Figure 21. Model of the Phileas rover of the Austrian Space Forum (Austrian Space Forum, 2017)

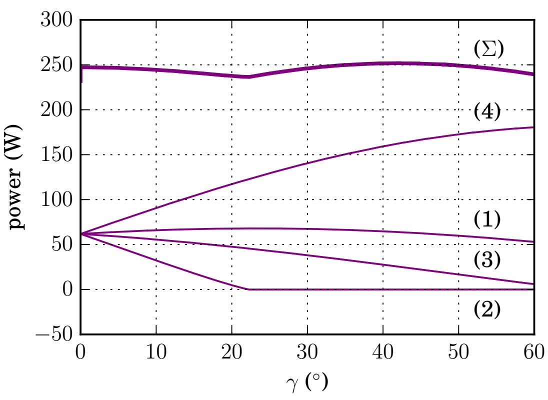


Figure 22. Power versus inclination angle γ of panels (1)–(4) and sum (Σ)

10 Conclusions

In this paper the open source `PhotoVoltaics` library was presented. Various photovoltaic components are explained based on typical data sheet parameters provided by manufacturers. Different models of cells, modules and plants are explained. Additional models of converters including maximum power tracking are described. In order to make systemic investigations, different analytic solar model are introduced.

Based on measurement data of two small photovoltaic plant at the TGM in Vienna, a validation of the library is performed. An application example of the Phileas rover of the Austrian Space Forum is investigated to demonstrate the potential of systemic investigations enabled by the `PhotoVoltaics` library.

References

OeWF Austrian Space Forum. The Phileas Rover, June 2017. URL http://oewf.org/en/polares-science/phileas-rover/.

BuildSysPro. Edf's modelica library for buildings, districts and energy systems modelling, June 2017. URL https://github.com/edf-enerbat/BuildSysPro.

Comax. Photo voltaic module data sheet Comax TSM 200 DC 01A, June 2017. URL http://www.elektra.si/uploads/datoteke/trina_tsm-195-200-205-210dc80.08_mono.pdf.

EnergyPlus. Weather data by location, europe wmo region 6 - austria, June 2017. URL https://energyplus.net/weather-location/europe_wmo_region_6/AUT//AUT_Vienna.Schwechat.110360_IWEC.

Heinrich Häberlin and Christian Renken. Allmähliche Reduktion des Energieertrags von Photovoltaikanlagen durch permanente Verschmutzung und Degradation. *Bulletin SEVIVSE 10/99*, 1999.

Christian Kral. Modelica PhotoVoltaics library, June 2017. URL https://github.com/christiankral/PhotoVoltaics.

Yousef Mahmoud, W. Xiao, and H. H. Zeineldin. A simple approach to modeling and simulation of photovoltaic modules. *IEEE Transactions on Sustainable Energy*, 3(1):185–186, January 2012.

PVSystems. Library toolbox for photovoltaic systems analysis, June 2017. URL https://github.com/mmanana/pvsystems.

Volker Quaschning. *Regenerative Energiesysteme: Technologie - Berechnung - Simulation*. Hanser, 7 edition, 2011.

C. Renken and H. Häberlin. Langzeitverhalten von netzgekoppelten Photovoltaikanlagen; Schlussbericht. Technical report, Berner Fachhochschule, Hochschule für Technik und Architektur (HTA) Burgdorf, 1999.

Trina. Photo voltaic module data sheet Trina TSM 230 PC05, June 2017. URL https://http://www.franklin-electric.com.au/media/52578/Trina%20TSM190-200DC01A_WW.pdf.

Michael Wetter. Modelica Buildings library, June 2017. URL https://github.com/lbl-srg/modelica-buildings.

Python-Modelica Framework for Automated Simulation and Optimization

Mareike Leimeister[1,2]

[1]Naval Architecture, Ocean and Marine Engineering, University of Strathclyde, United Kingdom
[2]Fraunhofer IWES, Fraunhofer Institute for Wind Energy Systems, Germany,
mareike.leimeister@iwes.fraunhofer.de

Abstract

Modeling and simulation are essential for the development of complex engineering systems, such as wind turbines. Thus, Fraunhofer IWES (Fraunhofer Institute for Wind Energy Systems) has developed the MoWiT (Modelica for Wind Turbines) library for fully-coupled aero-hydro-servo-elastic simulations of wind turbine systems. To meet the needs for detailed assessment and design development of such sophisticated engineering systems, which imply iterative steps for design optimization, a Python-Modelica framework is set up and presented in this paper. By means of this, the simulation of MoWiT models can easily be managed, including redefinition of model parameters, specification of output sensors and simulation settings, integration of optimization algorithms, post-processing of simulation results, as well as parallel execution of several simulations. The application of this Python-Modelica framework is shown based on the example of a design optimization task of a floating wind turbine support structure.

Keywords: Modelica, OneWind, MoWiT, Python, wind turbines, automated design optimization

1 Introduction

The development process of engineering systems is very complex, labor-intensive, and extensive. System simulation, analysis, and of course optimization are of high importance in, for example, power, control, automotive, aerospace, marine, material, or building engineering. The focus of interest could range from general design optimization, through performance or efficiency enhancement, including for example flow properties or comfort aspects, to a commonly envisaged cost reduction. Regardless of objectives, constraints, criteria, and engineering system, design processes always implicate several iterations, in which the evolving designs are tested, analyzed, and modified accordingly until an optimized design is achieved. Thus, an automated simulation framework is essential to cope with the large number of simulations, required to assess and develop such an engineering system design in detail, but also to support design optimization processes, in which iterative simulations have to be executed.

Good examples for such intricate engineering systems and their extensive development process are wind turbines. These power plants have to comply with requirements from standards, such as IEC 61400-3 (International Electrotechnical Commission, 2009) or DNVGL-ST-0437 (DNV GL AS, 2016), and need to be tested on their performance in various environmental conditions, including loads and system responses. However, the complexity of wind turbine systems, with their non-linear system behavior and couplings between aerodynamics, hydrodynamics (if offshore), control system, and structural dynamics, makes modeling and simulation indispensable.

Thus, at Fraunhofer IWES (Fraunhofer Institute for Wind Energy Systems) a computational model for wind turbine load calculations has been developed in the open-source object-oriented and equation-based modeling language Modelica. This modeling language has the power to deal with multi-physics problems and, hence, can be used for simulation of various engineering systems. The MoWiT (Modelica for Wind Turbines) library[1] is capable of fully-coupled aero-hydro-servo-elastic simulations of wind turbine systems - onshore, bottom-fixed offshore, or even floating offshore. The hierarchical programming and the multibody approach in Modelica allow representation of the entire wind turbine system through models for single components. This component-based structure of the MoWiT library simplifies the adaption and modification of a wind turbine model because single components can easily be exchanged or customized. (Leimeister and Thomas, 2017; Thomas et al., 2014; Strobel et al., 2011)

Even if MoWiT can model the non-linear system behavior, a large number of simulations are required for the design of an optimized wind turbine system. For this purpose a Python-Modelica framework is developed for automated execution of simulations and optimization tasks.

[1]formerly OneWind Modelica library

In this paper, first the simulation framework in Python, interfacing with models created in MoWiT, is presented in detail in Section 2, followed by the extension of the framework for automated optimization applications, covered in Section 3. Afterwards, the application of this Python-Modelica framework is shown exemplarily on the design optimization of a floating wind turbine system (Section 4). Conclusions are given at the end in Section 5.

2 Simulation Framework in Python

The framework for automated simulation of wind turbine models requires

1. a modeling environment, which is the MoWiT library building upon the Modelica modeling language;
2. a tool for executing the time-domain simulations (Dymola[2]);
3. and a programming interface (Python[3]) for external and automated control of the simulations.

The tools, which are selected to be incorporated in one framework for automated simulation, stand in perfect mutual complement. The Modelica based modeling environment in combination with the Dymola simulation engine is very suitable for time-domain simulations of complex multi-physics engineering problems. Programming in Python, on the other hand, facilitates the management and handling of simulations, controls the entire simulation process, and creates a set framework for automated application to engineering systems models and problems.

A schematic representation of the simulation framework in Python is presented in Figure 1. In Modelica, using the MoWiT library, the considered wind turbine system is specified and all parameters are set, so that the model can be simulated in Dymola. With setting up the MoWiT model, a Modelica package is created. This is the main input to the Python-Modelica framework as it contains all necessary information about the simulated model (structure, components, parameters, equations, states, ...). Due to the fact that the Python-Modelica framework should also be used to set and modify parameter values according to specific simulation requirements, it is important to add `annotation(Evaluate=false)` to these parameters, when defining them in the MoWiT model.

The simulation framework in Python itself works on different levels, as shown in Figure 1. It contains a `Model Wrapper` for processing the Modelica package and establishing the interface to Modelica based on the Python package `BuildingsPy`[4]. On the next level, the `Simulation Manager` handles the instances from the `Model Wrapper` and manages the simulations. Finally, a main script is required to execute the simulation task and define additional commands, for example for writing result files or for post-processing.

2.1 Processing the Modelica Package

The Modelica package of the created MoWiT wind turbine system model is given as input to the `Model Wrapper`. The Python-Modelica interface is defined based on the available interface between Python, Modelica, and Dymola, provided by the Python package `BuildingsPy`[4]. Within this package, the main class, which is finally required to simulate a Modelica model, is the `Simulator`.

2.1.1 The Simulator

The Python script for the class `Simulator` is taken from the Python package `BuildingsPy` and slightly modified to make it compatible with the used Python 3.x version. The `Simulator` provides the interface between Python and Modelica to run simulations with Dymola. Based on the inputs for model name and the path to the Modelica package of the MoWiT model, the used simulation engine (Dymola) and the path to the executable, as well as optional inputs for working and

[2]https://www.3ds.com/products-services/catia/products/dymola/ (Accessed: 5 October 2018)

[3]https://www.python.org/ (Accessed: 26 October 2018)

[4]http://simulationresearch.lbl.gov/modelica/buildingspy/ (Accessed: 9 October 2018)

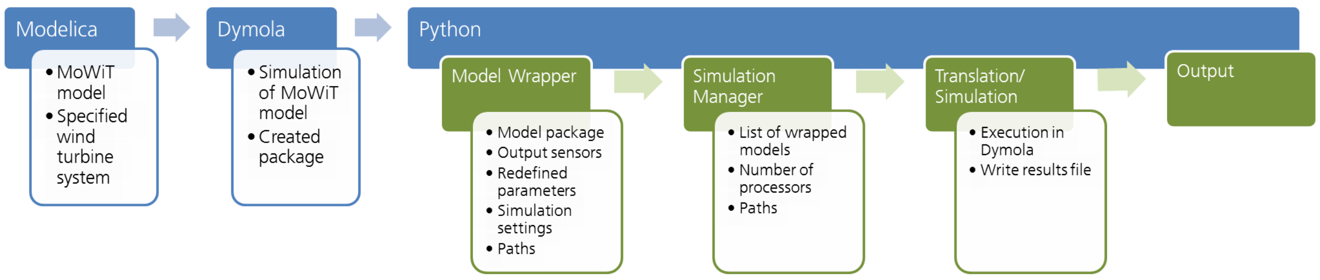


Figure 1. Simulation framework in Python.

output directories, the methods for setting paths, directories, but also simulation parameters and commands are defined. Furthermore, methods for adding pre-processing statements when translating or simulating the model, as well as post-processing statements before writing the log-file are specified. Finally, the methods to simulate a model, translate a model, or simulate a translated model are declared.

2.1.2 The Model Wrapper

The `Simulator` is called within the `Model Wrapper`, which specifies parameters, paths, and simulation settings for Dymola to be used in the `Simulator`. Besides this, also parameters to be set for translation or to be redefined before simulation, pre-processing statements to be added ahead of translation or simulation, as well as a list of output sensor names are defined.

The basic `Model Wrapper` class directly processes a Modelica package of a MoWiT model and modifies, if required, specified parameters and settings. Additionally, a method is defined to write Dymola commands for generating a `*csv` output file after completion of the simulation. Furthermore, the total number of simulations to be executed is specified. This is especially relevant when running several simulations, which could be processed in parallel or successively. This is managed by the `Simulation Manager`, which is introduced in the next Subsection 2.2.

2.2 Managing the Simulation

Wrapped models are then further processed in the `Simulation Manager`. The input list of wrapped models could contain

- one instance of a `Model Wrapper` class, corresponding to just one MoWiT model;
- one instance of a `Model Wrapper` class, based on one and the same model, however, comprising several simulations with different parameter settings;
- or several different instances of a `Model Wrapper` class for working with various MoWiT models.

These models in the list of wrapped models can be handled either successively or in parallel, while for the latter the number of processors used for multi-processing in a pool can be specified as additional input to the `Simulation Manager`. Both forms of management are available for different processing methods:

- translating a wrapped model;
- simulating a translated wrapped model;
- creating a turbulent wind file.

The first two methods are calling functions in the `Simulator`. The latter method is only relevant for simulations with turbulent wind. This is defined through the turbulence intensity, as well as the wind spectrum type (Kaimal, von Karman, or Mann). In this Python-Modelica framework application, TurbSim (Jonkman, 2009) is used for generating a turbulent wind field. A file containing the time series of the wind speed could

- either already exist and the path to this file has to be specified directly in the MoWiT model or is given as input to the `Simulation Manager`;
- or still has to be generated, which requires the path to the TurbSim executable, as well as the wind turbine and simulation case specific TurbSim input file.

2.3 Executing the Task

Finally, a main script is required to execute the simulation task and define additional commands, for example for writing result files or post-processing calculations. This script highly depends on the application case. Hence, for running a large number of simulations, such as in the case of design load case simulations, only the simulations to be executed, as well as simulation settings, paths, and input parameters are to be specified. However, for applying the Python-Modelica framework to automated optimization tasks, additional code, in which design variables and objective functions are defined and linked to existing Python packages for optimization, has to be written in the main script. More detailed information on the Python-Modelica framework extension for the use for automated optimization is given in the following Section 3.

3 Extension for Automated Optimization

The Python-Modelica framework, as presented in Section 2, serves as basis for further applications, apart from automated simulation, such as the realization of optimizations. The extension of the Python-Modelica framework for automated optimization is of significant importance, as optimization tasks are highly iterative. This finds profitable use in the design and optimization of wind turbine systems. Due to the complexity of an (offshore) wind turbine model, comprising a huge number of parameters, and the non-linear system behavior, optimization problems cannot directly be solved and a large number of iterations has to be gone through.

The wind turbine system model, which should be used for optimization purposes, has to be wrapped and processed with the `Model Wrapper` and `Simulation Manager`, respectively, according to the explanations in Subsections 2.1 and 2.2. This, together with further definitions regarding the optimization process, is passed to the main script, by which means finally the execution of the optimization algorithm is started (see Subsection 2.3). Additional information on the optimization itself is provided by separate classes, clustered into the optimization problem, the optimizer, and the optimization algorithm, which are introduced in the following Subsections 3.1 to 3.3.

3.1 The Optimization Problem

Based on the model from the `Simulation Manager`, the optimization problem has to be described. This comprises definitions of design (also called optimization) variables, objective functions, as well as additional constraints. As the Python-Modelica framework works without a GUI, all input has to be provided in the programming scripts. These, however, are coded in such a way, that they all internally use variables, which are only once defined in the main script and assigned their values by means of the user input.

The optimization variables are the design parameters of the wind turbine model, which are to be modified during the optimization iterations. The parameter names must be provided according to the Modelica dot-notation and following the structure within the MoWiT model. Since these parameters are assigned new values during the optimization, it is important that they are still existing in the compiled model (see the remark at the beginning of Section 2).

As important as optimization variables for an optimization procedure are objective functions. These describe the goals, which are to be obtained by means of the optimization. Mostly, optimization routines are defined to minimize the objective functions, thus, these have to be provided accordingly. Depending on the optimization routine type, only one or several objective functions can be processed. For multi-objective optimizers, each goal can be defined separately. However, if the optimizer can handle only one objective function, all goals have to be combined in one expression, in which weight can be incorporated to rank the importance of the single objectives.

The two key elements of the optimization problem are already specified by means of the optimization variables and the objective functions; however, further input can be given in form of constraints. These apply either for the goals and specify if only certain values are allowed or if dependencies or relations exist, or for the design parameters and define the allowable ranges of values which they can take on.

3.2 The Optimizer

The optimization problem is given to an optimizer, which then executes the optimization task and algorithm. There are several open-source optimizers available for the use in a Python environment, such as optimization routines from OpenMDAO (Multi-disciplinary Design, Analysis, and Optimization), an open-source framework for efficient multi-disciplinary optimization, (openmdao.org, 2016); PyGMO (Python Parallel Global Multi-objective Optimizer), focussing on multi-objective (MO) optimization, (Izzo and Biscani, 2015); or Platypus with a special focus on MOEAs (MO Evolutionary Algorithms) (Hadka, 2015) - just to name a few examples.

In the presented Python-Modelica framework, optimization routines from Platypus (Hadka, 2015) and OpenMDAO (openmdao.org, 2016) are implemented. Only gradient-free optimizers can be used for the application to wind turbine models in MoWiT, as these models represent too complex systems, which cannot be reduced to one single equation by means of minimization techniques. Furthermore, the high complexity also attributes greater importance to multi-objective optimizers.

Apart from optimizer-specific inputs, a criterion has to be specified for limiting the number of iterations within the optimization process. This could be defined for instance through the number of optimization cycles to be performed or a convergence tolerance for the results.

3.3 The Optimization Algorithm

Using the defined optimization problem and the specified optimizer, as described in Subsections 3.1 and 3.2, respectively, the optimization algorithm is executed. In each run, the design variables are modified, based on the objective results from previous simulations, complying with the defined value ranges of the optimization variables, and following the optimizer-specific routine. The iterative optimization simulations are terminated as soon as the specified stop criterion is fulfilled. Figure 2 visualizes this process schematically. Furthermore, depending on the specified processing method, as set in the `Simulation Manager` (see Subsection 2.2), several simulations within the optimization routine may be executed in parallel.

Due to the fact that - especially at the beginning of the optimization routine - also suboptimal settings might be selected by the optimizer, it could happen that simulations of individual models are aborted before the specified simulation duration. To handle these or similar failures a query condition can be incorporated when analysing the results for evaluating the objective functions. One possible approach is to check if the simulation was successful by evaluating the last entry in the time output. In case of aborted simulations, the goals might not be

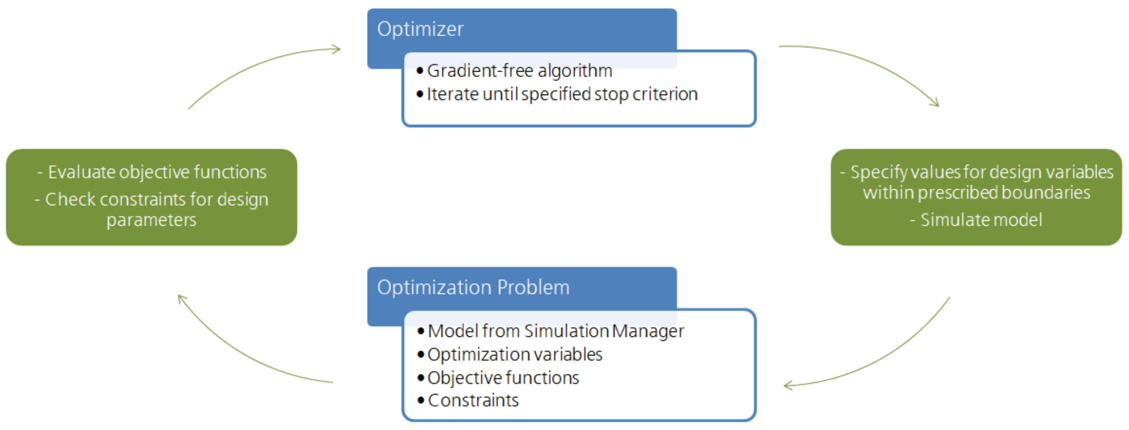

Optimizer
• Gradient-free algorithm
• Iterate until specified stop criterion
- Specify values for design variables within prescribed boundaries
- Simulate model
Optimization Problem
• Model from Simulation Manager
• Optimization variables
• Objective functions
• Constraints
- Evaluate objective functions
- Check constraints for design parameters

Figure 2. Automated optimization algorithm in Python.

derived as defined, but set to undesireable values to ensure that these unsuccessful and thus suboptimal individuals are excluded and not considered further by the optimizer.

During the execution of the optimization algorithm, the simulation results are written in *csv output files according to the defined method in the `Model Wrapper`, as outlined in Paragraph 2.1.2. By means of supplementary code, additional outputs, such as the objectives of each solution, can be written and exported subsequent to the optimization.

4 Application Example: Design Optimization of Wind Turbine Systems

Optimization tasks in the development of wind turbine systems are wide-ranging. Mostly costs, and thus indirectly also performance and material demand, are the main drivers, but optimization problems can for instance as well be related to noise emissions, dimensions, and lifetime. In the following the Python-Modelica framework is exemplarily applied to automated design optimization of a floating offshore wind turbine system.

In this optimization task, the floating spar-buoy wind turbine system from phase IV of the Offshore Code Comparison Collaboration project OC3 (Jonkman, 2010) is used. The floating wind turbine system consists of a spar-buoy platform, which supports the NREL 5 MW reference wind turbine (Jonkman et al., 2009). The visualization of the MoWiT model in Dymola is presented in Figure 3. There, also the coordinate system of the wind turbine, as well as the nomenclature of the six degrees of freedom of movement are introduced.

The optimization algorithm is defined based on the following problem and settings:

- Three parameters of the spar-buoy floating platform are selected as design variables with their corresponding allowable value ranges: the diameter (between 6.5 m and 10.0 m) and the height (between 68.0 m and 108.0 m) of the spar-buoy column, as well as the density of the ballast (between 1281.0 kg/m^3 and 2600.0 kg/m^3). A fourth indirect variable, the amount of ballast (filling height in the column), is internally determined and adjusted, based on the design parameters and to ensure floatation of the system.

- Three objective functions and corresponding constraints are defined to limit the maximum system inclination (pitch) to 10°, limit the maximum nacelle or tower-top acceleration to 1.962 m/s^2, and to minimize the floater translational motion (combined surge, sway, and heave).

- The optimizer NSGAII[5] from Platypus (Hadka, 2015) is used due to the multi-objective optimization task and the optimization algorithm is executed for 25 generations with 36 individuals each.

The variation of the floater design within the optimization algorithm is shown in Figure 4. The black shape represents the original geometry and corresponding ballast height (indicated by the dashed line). A few exemplary geometries of individuals obtained during the optimization are presented in different green tones and reveal the ranges of the design variables.

A more detailed analysis of the simulation results shows that both the spread of the design parameters and the spread of the optimization objectives converge, with having a minimum spread in generation number 13, as indicated in Figure 5. From this, the final optimum geometry is selected, which is displayed in red in Figure 4. With this design, the objectives are achieved, while still fulfilling the prescribed boundaries and constraints for the design parameters.

[5]Non-dominated Sorting Genetic Algorithm II

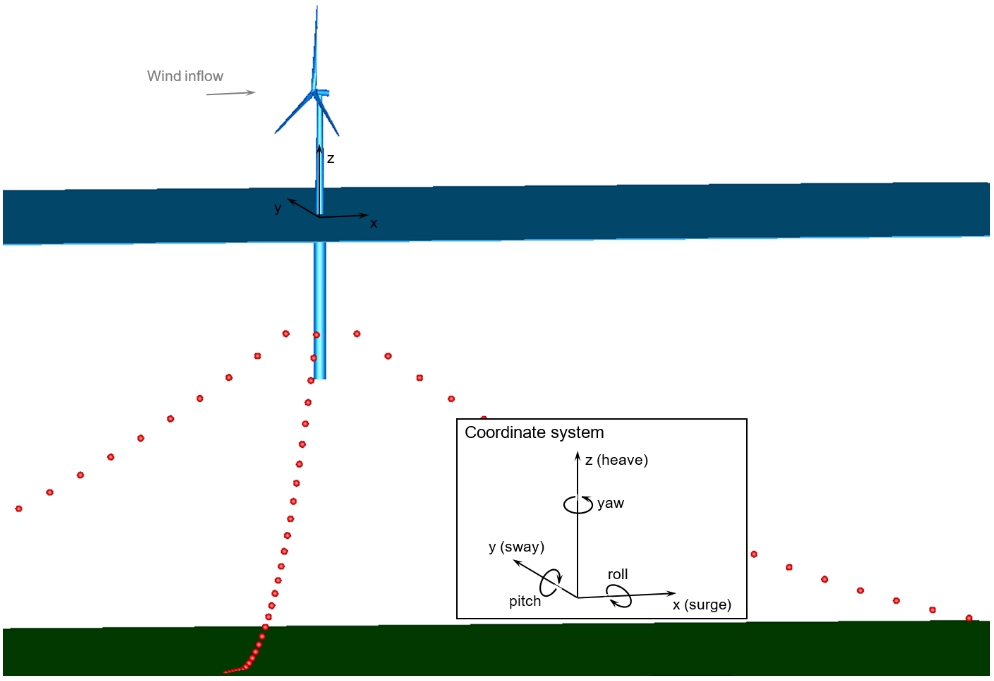


Figure 3. Floating spar-buoy wind turbine system in MoWiT, including coordinate system and system degrees of freedom, as well as wind inflow direction.

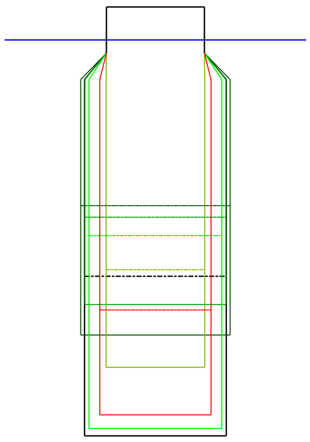
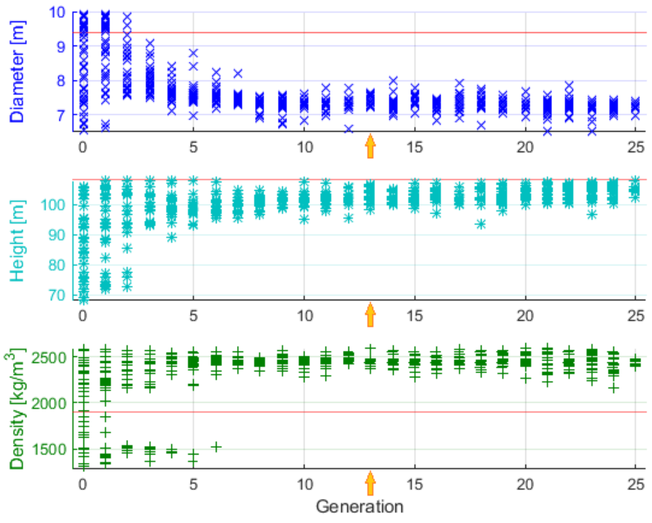
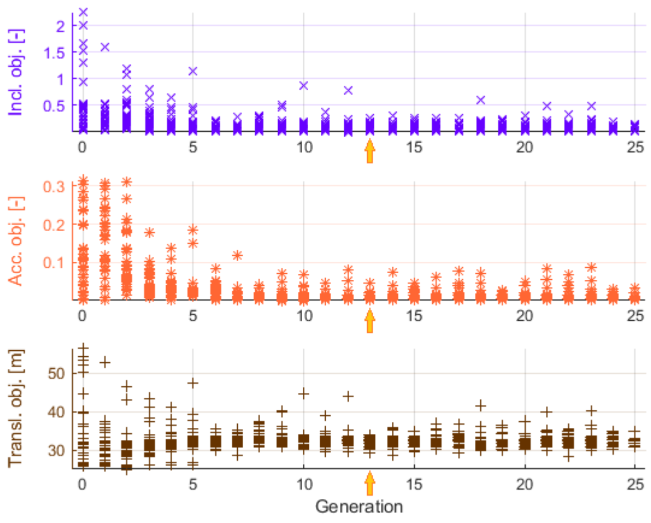


Figure 4. Interim (green tones) and final (red) results from the floater design optimization procedure, in comparison with the original design (black).



(a) Development of the design variables column diameter (blue), column height (cyan), and ballast density (green), together with the original values (red).



(b) Development of the objectives for system inclination (violet), nacelle acceleration (orange), and floater translation (brown).

Figure 5. Results from the floater design optimization procedure, arrows indicate the generation from which the final optimum design is selected.

5 Conclusion

In this paper, a Python-Modelica framework is presented, by which means Modelica models can be managed and simulations executed automatically, using scripts programmed in Python. Models for entire wind turbine systems (onshore, bottom-fixed offshore, or even floating offshore) are created in the MoWiT library, which are then simulated in Dymola. The external and automated control of the simulations is taken over by various Python scripts. These are split up into methods for processing the Modelica package of the MoWiT model, methods for managing the simulation, and the main script for executing the task and performing further (post-)processing. By means of this Python-Modelica framework iterative simulations can automatically be performed, which is very relevant for the assessment, design, and optimization of wind turbine systems. For the latter application, the framework is extended to cover also definitions for optimization algorithms, including optimization problem and optimizer.

An exemplary optimization task for design optimization of a floating wind turbine support structure demonstrates that the presented Python-Modelica framework automates the execution of a large number of simulations, is capable of handling non-linear system behaviors, and thus is a valuable tool for detailed assessment of wind turbine system designs.

Acknowledgements

This work was partially supported by grant EP/L016303/1 for Cranfield University, University of Oxford and University of Strathclyde, Centre for Doctoral Training in Renewable Energy Marine Structures - REMS (http://www.rems-cdt.ac.uk/) from the UK Engineering and Physical Sciences Research Council (EPSRC). The author also wants to thank Philipp Thomas and Niklas Requate from Fraunhofer IWES, as well as Athanasios Kolios and Maurizio Collu from University of Strathclyde for their contribution.

References

DNV GL AS. *Loads and site conditions for wind turbines: Standard DNVGL-ST-0437*. November 2016 edition, 2016. URL `https://www.dnvgl.com/`.

David Hadka. *Platypus Documentation, Release*. 2015. URL `https://platypus.readthedocs.io/en/latest/`.

International Electrotechnical Commission. *Wind turbines – Part 3: Design requirements for offshore wind turbines: International standard IEC 61400-3*. 1.0 edition, 2009.

Dario Izzo and Francesco Biscani. Welcome to PyGMO, 2015. URL `https://esa.github.io/pygmo/index.html`.

Bonnie J. Jonkman. *TurbSim User's Guide: Version 1.50: Technical Report NREL/TP-500-46198*. National Renewable Energy Laboratory, 2009.

Jason Jonkman. *Definition of the Floating System for Phase IV of OC3: Technical Report NREL/TP-500-47535*. National Renewable Energy Laboratory, 2010.

Jason Jonkman, Sandy Butterfield, Walt Musial, and George Scott. *Definition of a 5-MW Reference Wind Turbine for Offshore System Development: Technical Report NREL/TP-500-38060*. National Renewable Energy Laboratory, 2009.

Mareike Leimeister and Philipp Thomas. The OneWind Modelica Library for Floating Offshore Wind Turbine Simulations with Flexible Structures. In *Proceedings of the 12th International Modelica Conference*, Linköping Electronic Conference Proceedings, pages 633–642. Linköping University Electronic Press, 2017. doi:10.3384/ecp17132633.

openmdao.org. OpenMDAO 2.4.0 Beta documentation: Optimizer, 2016. URL `http://openmdao.org/twodocs/versions/latest/tags/Optimizer.html#optimizer`.

Michael Strobel, Fabian Vorpahl, Claudio Hillmann, Xin Gu, Adam Zuga, and Urs Wihlfahrt. The OnWind Modelica Library for Offshore Wind Turbines - Implementation and First Results. In *Proceedings of the 8th International Modelica Conference*, Linköping Electronic Conference Proceedings, pages 603–609. Linköping University Electronic Press, 2011. doi:10.3384/ecp11063603.

Philipp Thomas, Xin Gu, Roland Samlaus, Claudio Hillmann, and Urs Wihlfahrt. The OneWind Modelica Library for Wind Turbine Simulation with Flexible Structure - Modal Reduction Method in Modelica. In *Proceedings of the 10th International Modelica Conference*, Linköping Electronic Conference Proceedings, pages 939–948. Linköping University Electronic Press, 2014. doi:10.3384/ECP14096939.

Demand oriented Modelling of coupled Energy Grids

Jörn Benthin[1] Annika Heyer[1] Philipp Huismann[1] Michael Djukow[1] Anne Hagemeier[2] Klaus Görner[1]

[1]Gas- und Wärme-Institut Essen e.V., Germany,
{benthin, heyer, huismann, k.goerner}@gwi-essen.de

[2]Fraunhofer UMSICHT, Germany, anne.hagemeier@umsicht.fraunhofer.de

Abstract

This paper describes the development of a modular approach for modelling and simulation of coupled energy grids within different kinds of settlement structures. One presented thesis is that the spatial distribution of the demand structure is given by the urban framework and should be the origin of the modelling of coupled energy grids on the distribution level. Thus, the logic is that the grid is developing towards the given and developing demand structure and not vice versa. The defined spatial distribution delivers the loss relevant lengths between the consumers and producers, which are needed for pipes and cables.

Following these assumptions, a modular approach was realised by creating a so-called *GridConstructor*. This constructor allows it to easily build user defined urban frameworks and combine them with a single grid or multiple grids (electricity, gas, heat). These grids can be coupled via different systems. In conclusion, first results of coupled grid simulations are presented.

Keywords:

Thermodynamic and energy systems applications, Large-scale system modelling

1 Introduction

The ongoing integration of renewable energy sources into the different energy grids is one of the major tasks for the next decades. The overall goal behind this integration is the decarbonisation of the energy consumption in the different structural sectors (industrial, service, residential, mobility). Due to the highly volatile and increasing power input of the renewable energy sources, the need for coupled energy grids and flexibilities is rising (Behnert, 2018).

Especially on the distribution level, the question arises how the different grid designs (electricity, gas, heat) will look like and how these networks are going to connect and interact. To find the ecological and technological optimum, different coupled design options have to be analysed.

To address these questions, different libraries and software tools are available. In general, these tools share one common approach. This approach is defined by the modelling hierarchy, which sets the grid structure as a fixed boundary and not the urban framework.

The natural process in the development of cities and districts is that the demand structure is given by the urban framework and its developing demand and decentralised production. The energy grids have to develop towards these needs and therefore the grid design, connections and interactions are the variables of this adjustment process. The given or designed spatial structure of the settlement delivers the loss relevant lengths between the consumers, producers and the demand density, which are needed for the sizing of pipes, cables and the design of the resulting networks.

These assumptions were used in the research project *IntegraNet* (IntegraNet, 2018) to develop a modular approach for modelling urban frameworks at the distribution grid level.

The *TransiEnt* (TransiEnt, 2018) library developed within the project Transient.EE (Andresen, 2017) by the Hamburg University of Technology was used for the work presented in this paper.

2 GridConstructor

2.1 Modelling Philosophy

The future energy grid will undergo considerable changes through decentralisation, an increasing share of renewable energy supply and sector coupling technologies. In order to investigate the potentials of these technologies and other feasible innovations in grid design, the modelling of energy grids needs to be able to map the existing structure as well as future options. Thus, a modelling philosophy for energy grids should be based upon flexibility to allow the research of multiple configurations of technologies and types of energy supply.

In case of energy grids, the network structure, as in routing and connections of pipes and cables to consumers, is not fixed and can change depending on the energy supply strategy and time. For example, it might happen that an energy concept for a district based entirely on decentralised oil heating systems is converted to an energy supply via district heating, PtH or gas heating. In this case the overall grid structures are changed and extended, but the demand structure, as in the distances between consumers and their location, stays constant. Based upon this, the modelling of energy grids only from a standpoint of grid structure with fixed types of pipes and cables, that are derived from the given technologies, is no optimal solution to target research projects regarding the development of future energy grids. The use of such models would make simulations with changing technologies difficult since changes in energy generation can involve a change in grid structure. Instead, modelling based on demand structure is more feasible, since it will usually not undergo modifications with changed technologies or grid structure.

2.2 Structure

Based on a demand structure oriented modelling, the *GridConstructor* (GC) has been conceptualised for Modelica. The GC uses reoccurring patterns in the demand structure to model the energy grid in a flexible and user-friendly way. Based upon settlement types defined in (Blesl, 2001), energy grids can be described using several grid segments with changing numbers of consumers. These grid segments contain variable number of grid elements with one or two consumers each (see Figure 1).

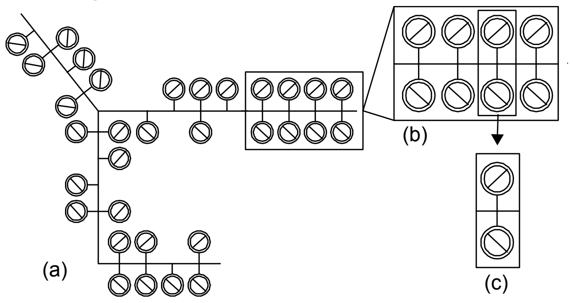


Figure 1: Demand structure (a - distribution grid, b – grid segment, c – grid element)

In the presented modelling approach, one GC represents one grid segment. The GC enables the modelling of a grid segment by creating a series of grid elements using arrays of models. For the GC, the used technologies, number of consumers as well as grid structure parameters such as length or type of cables are freely exchangeable. By connecting several GC, each representing a grid segment, a distribution grid can be modelled. Each of the GC used in the modelling of the distribution grid can be individually configured regarding technology as well as properties for the grid structure like cable type or length. Due to the interchangeability of parameters and technologies inside the GC, case studies of changing configurations of grid structures and used technologies of existing energy grids as well as future scenarios are possible.

Each GC consists of an array of grid element models (*GridElement* (GE)). Depending on user input, one or multiple GE are strung together to create a grid segment. The basic idea of the GE model is that each grid element ultimately consists of an energy demand (electricity, space heating and hot water) and technologies to meet that demand and/or to generate additional energy (see Figure 2). Each of the consumers inside the GE is represented by its own energy demand and technologies. For each GE, either one or two consumers can be activated. In this way, an exact representation of the demand and grid structure is possible as well as a simplified representation.

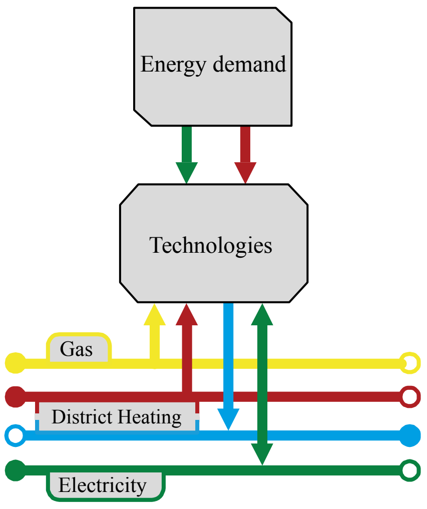


Figure 2: Basic idea behind the *GridElement* structure and connection scheme for one consumer inside a *GridElement*.

The GE model consists of sub-models providing the mentioned demand, representing used technologies and grid structures inside the grid element (see Figure 3). This includes:

- Gas and district heating pipe models
- Electric cable models
- Models representing used technologies
- Models providing time series of energy demand

Simulations of gas distribution networks are enabled by the use of gas pipe models taken from the Modelica library *TransiEnt* and adjusted to calculate the pressure drop not from nominal values, but via the Darcy-Weißbach equation with the function *StraightPipe.dp_overall_MFLOW* provided by the *FluidDissipation* library inside Modelica. The gas pipe is parametrised regarding length and diameter. Changing composition of the gas, for example through hydrogen injection, is accounted for.

The electric grid is modelled by using an electric cable model from *TransiEnt* library. The type of cable (diameter, material, resistance) is freely exchangeable via use of replaceable models.

For simulations of district heating networks, three models representing district heating pipe pairs are placed inside the GE as replaceables. The central pipe model represents the distribution pipe of the heat carrier in the grid element. For the connection of consumers to the distribution pipe, two pipe models are placed perpendicular to the distribution pipe. If faster simulations are desired or physical effects in the house pipes are neglected, the models of the house connections may be deactivated. The use of parallel pipe models increases usability by parametrising the pipes homogeneously for length and geometry. Alternatively, the user is able to specify the nominal diameter based upon manufacturer specifications saved within records inside Modelica from which the pipe geometry is set. Initialization parameters for temperature and pressure for supply and return pipes are provided on the top level of the modelled energy grid.

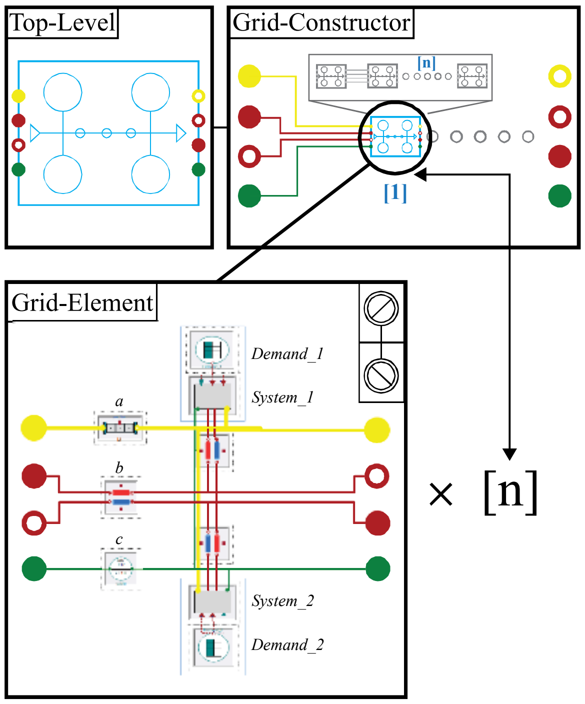


Figure 3: Structure of the *GridConstructor* and *GridElement*. a – gas pipe model, b – district heating pipe model, c – electric cable model.

The district heating pipes used inside the GE are modelled after the PlugFlow approach described in (Heijde, 2017; Hägg, 2016). Contrary to pipe models with spatial discretisation, the PlugFlow approach determines the fluid properties only at the inlet and outlet of the pipe by means of the residence time. Temperature wave propagation and heat loss are both handled by use of the spatialDistribution() function included in Modelica. Validation work carried out for the PlugFlow approach in (Heijde, 2017) and as part of simulations within the *IntegraNet* shows good accuracy to measurements.
Pressure drop calculations inside the pipe model are carried out with the *Darcy-Weißbach* equation using the function *StraightPipe.dp_overall_MFLOW* provided by the *FluidDissipation* library. Fluid properties such as density or specific heat capacity are calculated with models from the *TILMedia* suite developed by the TLK-Thermo GmbH.

Electric cables, gas pipes and district heating pipes can be deactivated or activated as required. Corresponding connectors are deactivated automatically. Multiple GE inside a GC and sub-models inside of a GE are connected automatically.

The technologies used inside the GE are integrated in models called *Systems_1* and *Systems_2* representing technologies used for the top consumer (*1*) and bottom consumer (*2*) of the grid element. Consumers inside a grid segment are divided into a top row and bottom row representing the two sides of a road. Inside each of the *System* models multiple models for technologies like gas boilers, photovoltaics or heat pumps are activated or deactivated based upon the settings defined during parametrisation of the GC.

For the energy demand, time series of heat and/or cooling and electricity demand with arbitrary resolution (s, min, h) are used and provided by the models *Demand_1* and *Demand_2* for both consumers. These time series are saved inside comma separated value (csv) files and may be provided for the model outside Modelica based upon real life measurements, computer generated data or standard load profiles. Technology models defined inside *System* use these profiles to meet the associated demand.

2.3 Programming & Modelling details

To enable flexible modelling of grid segments, the necessary programming effort can be divided into the following problem statements:

- Enabling the creation of the demand and grid structure
- Exchangeability of technologies
- Enabling a user-friendly parametrisation

The demand and grid structure inside the grid segment are achieved by use of Boolean arrays. For this, the user has to define the number of grid elements *n* inside the grid segment. From this, *n* GE models are initialized inside a GC model (see Figure 4).

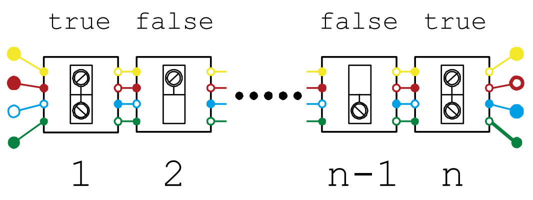


Figure 4: Arrangement of GE models inside a GC

By use of for-loops, automatic connections of GE inside the GC are achieved:

```
for i in 1:n-1 loop

      connect(Grid_Element[i].PortOut,
      Grid_Elements[i+1].PortIn);

end for;
```

The inlet connectors for the first GE (n=1) and the outlet connectors for the last GE (n) are connected to the connectors of the GC by regular connect statements. All connectors and their interconnections are dependent on booleans, which are set depending on the technologies used. Also, the models for gas pipes, district heating pipes and cables are conditional models and are activated/deactivated depending on parametrisation. Therefore, unused models are removed from the equations during compilation.

In each GE, one or two consumers can be connected to the energy grid. To achieve this property, an array of Booleans *second_Consumer*[:] is created with which all models belonging to the second consumer are activated if the boolean is *true* (see Figure 4). This Boolean is always set *false* if only one consumer is supposed to exist in the related GE, no matter if it's the top or bottom consumer.

By use of records, properties of the GE models inside the GC are set. The records are defined to have all the properties associated to a given type of parametrisation objective. For example, length and cable type of the electric cable models inside each GE are set as array parameters inside the *record CableParameters.* The length of the cable model in each GE inside a GC is assigned as follows:

```
parameter Records.CableParameters
pipeparameters[:] =
fill(Records.CableParameters(),30);

protected

parameter SI.Length len[:]=
CableParameters.len_cable;

      [ . . . ]

Grid_Element[n](
      redeclare model cable =
      Components.CableModel(length = len),
      [ . . . ]  );
```

Records are not only used for the parametrisation of models representing grid structure, but also for the exchangeability of technologies in the models *System_1* and *System_2*. A record called *TechnologyMatrix* with parameters as integers for all available technologies is created. This record is used for each consumer (top and bottom) as a matrix with columns representing technologies and rows representing each GE. The set parameters inside this matrix are propagated from the GC to the corresponding GE and corresponding technology. With these propagated integers (zero or one), technology models like gas boilers or photovoltaic are activated or deactivated with if-Statements. Integers are used instead of booleans to reduce the time the user needs to enter the parameters. Technologies get activated with one and deactivated with zero. For this approach to work, all possible technology models must be present as default in the *System* model, but each of the models can be deactivated freely. Connectors inside the model *System (1* and *2)* are activated and deactivated based upon used technologies without user-input. For example, district heating connectors are only activated if technologies corresponding to district heating are used. The values inside the *TechnologyMatrix* can be freely changed from simulation to simulation.

Parametrisation of technologies in *System_1* and *System_2* for properties like efficiency is realised by use of further records containing the parameters used in the individual models. Set parameters are propagated to activated technologies and used as input for the models with the *redeclare model* statement.

The demand time series data is provided to *System* with a model *CombiTimeTable* each inside the model *Demand_1* and *Demand_2*. In order for each of the consumers to use their own energy demand as a time series, the data for space heating, domestic hot water heating and electricity must be stored inside three csv tables with each column representing another consumer. Alternatively, the demand time series can be provided in a single csv-file, which contains groups of columns with each column group specifying the demand data for one consumer. Each group consists of three columns that specify the data for the electricity, space heating and domestic hot water demand respectively. The time series for the energy demands are assigned to the *Demand* models for each GE by accessing the corresponding column in the provided time table. Columns are accessed and assigned in series starting from a user defined integer *start_c1* and *start_c2* representing the top and bottom consumer:

```
Grid_Element[n](

      [ . . . ]

  redeclare model Demand_1 = Demand_1
  (row={i for i in start_c1:(n+start_c1 -
  1))},

  redeclare model Demand_2 = Demand_2
  (row={i for i in start_c2:(n+start_c1 -
  1))},
      [ . . . ] );
```

2.4 Parametrisation

The parametrisation of one GC can be divided into the following steps:

1. Define number of grid elements (*n*) as well as the arrangement of consumers (*secondConsumer)*
2. Deactivate/activate inlet and outlet connectors
3. Set properties of energy distribution models
4. Assign load-profiles
5. Define and set properties of technologies used

Parametrisation of the GC starts with the definition of grid elements in the given grid segment. As an example, Figure 5 shows a grid segment consisting of six grid elements with one or two consumers per grid element. Based on this example, the user passes n=6 to the GC and arranges the consumers on the grid element by setting the booleans for *secondConsumer*. The booleans *secondConsumer* are stored in an array with the alignment orientation source to sink. The orientation of the consumer to the street for *secondConsumer* is irrelevant and does not influence the simulation (see Figure 5).

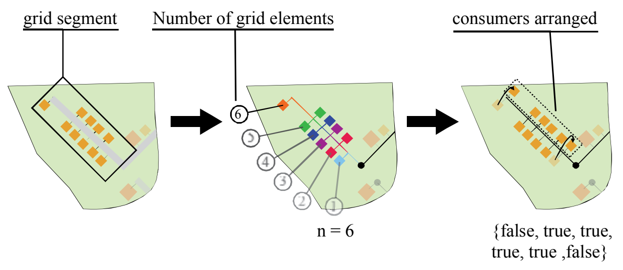
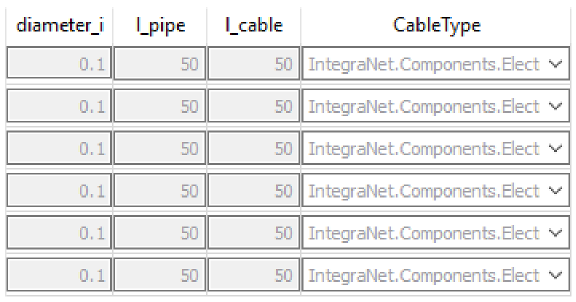


Figure 5: Graphical representation of the first steps for parameterizing the GC

As a next step, the user deactivates unused connectors of the GC with checkboxes. For example, the gas inlet and outlet connectors are not necessary if a district heating network and electricity network without gas network is to be simulated. As mentioned, this also automatically deactivates any connector and energy distribution model associated to the deactivated connectors. If previous GC models have already been placed and parameterised, the GC can be connected to the corresponding GC via the respective connectors.

Subsequently the properties of energy distribution models (gas pipes, cables and district heating pipes) are set by editing the corresponding records. Each row inside the array of records represents one of the grid elements (see Figure 6).

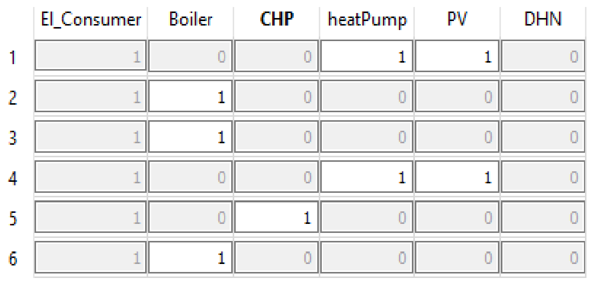


	diameter_i	l_pipe	l_cable	CableType
1	0.1	50	50	IntegraNet.Components.Elect
2	0.1	50	50	IntegraNet.Components.Elect
3	0.1	50	50	IntegraNet.Components.Elect
4	0.1	50	50	IntegraNet.Components.Elect
5	0.1	50	50	IntegraNet.Components.Elect
6	0.1	50	50	IntegraNet.Components.Elect

Figure 6: Parametrisation of gas pipes and electric cables models inside a GC

With the basic structure of the grid segment set, the user is able to pass the load profiles for heat and electricity demand to the GC. For this, the path to the csv-tables is set for the top consumers and bottom consumers. Consumers virtually transformed to exist in the top row during the arrangement of *secondConsumer* have to have their corresponding load profile inside the csv-table for the top row even though they, in reality, are placed on the other side. If the demand profile tables are used in more than one GC, the start column of the load profiles has to be passed as the integer *start_c1* and *start_c2*.

Finally, the technologies used in each network element of the GC are defined by using two arrays of the record *TechnologyMatrix* for the upper and lower rows of consumers. Technologies for the consumers are deactivated/activated with ones (activated) and zeros (deactivated). Analogue to the parametrisation of the pipes and cables, each row inside the parameter array represents a grid element and consumer (see Figure 7). Parameters of the activated technologies are set with further records similar to the parameterisation of gas pipe, cable and district heating pipe models.



	El_Consumer	Boiler	CHP	heatPump	PV	DHN
1	1	0	0	1	1	0
2	1	1	0	0	0	0
3	1	1	0	0	0	0
4	1	0	0	1	1	0
5	1	0	1	0	0	0
6	1	1	0	0	0	0

Figure 7: Defining technologies used in the GC for each GE with use of a record

With the GC fully parametrised and connected to other corresponding GC models, the simulation of the desired energy grid is possible.

Currently, the parametrisation is the most time-consuming part of modelling an energy grid, which is considerably simplified by the GC approach.

3 Use Cases

In the following section, utilizations of the GC for simulation of energy grids are presented. At first, a coupled gas and electric grid simulation of a rural district is described. Subsequently, the simulation of a low temperature district heating network (DHN) is described with the aim of investigating heat losses and potentials for Power-to-Heat (PtH).

3.1 Simulation of a coupled gas and electricity grid

In the following the first use case and the general functionality and usability of the provided models are presented. General challenges of modelling a coupled electric power and gas distribution grid with different technologies such as photovoltaic (PV) or Power-to-Gas (PtG) are shown.

In many research projects, no real measurement data of the demands per building is available. To anyhow introduce high fluctuations into the modelled district, profiles for the heat demand according to the VDI 4655 (Verein Deutscher Ingenieure, 2008) can be used. Using the same profile for each building would lead to unrealistically high gradients and maxima of the total demand. Thus, to also depict needs to balance between the consumers, it is necessary to shift the VDI-profiles individually for each building over time. For shifting a normal distribution is used.

One challenge when simulating large networks, is to secure stable initialisation. Especially when using parallel and serial networks of gas pipes, the initialisation of the pressure loss for the given heat demands at the first-time step is fairly difficult. To avoid having to calculate the complete network in a static cycle beforehand, initialisation using zero heat demand, resulting in zero gas flow, can be used. This increases the usability of the model, especially when changing the grid structure frequently due to different scenarios.

The described modelling approach has successfully been used to simulate and analyse a rural example district containing 314 buildings - all of them connected to the electric grid and 166 buildings also supplied with gas.

Figure 8 shows the resulting electricity surplus of this grid with PV for one day each of different seasons of the year.

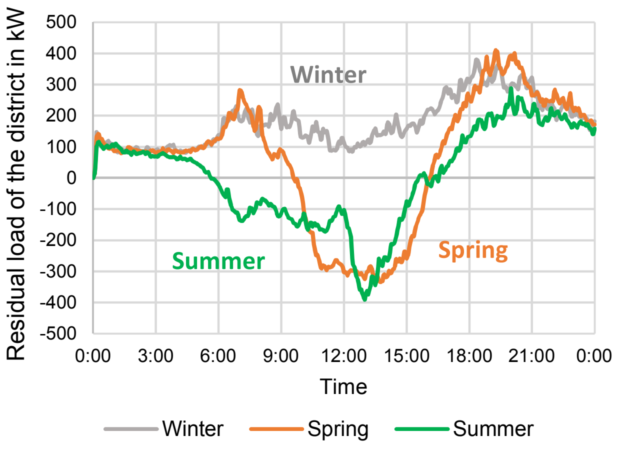


Figure 8: Total electricity load for the simulated example district with PV for three different days of the year

The simultaneous simulation of the coupled electric power and gas grid enables the analysis of sector coupling technologies, such as PtG. Converting all surplus electric power into hydrogen and feeding it into the gas grid leads for the observed grid for the spring day to the curves as depicted in Figure 9. The resulting volume fraction of hydrogen of up to 64 % exceeds the admissible range clearly. Implementing a monitoring of the gas properties and corresponding control strategies, enables the analysis of the resulting residual load with different limit values for the gas properties.

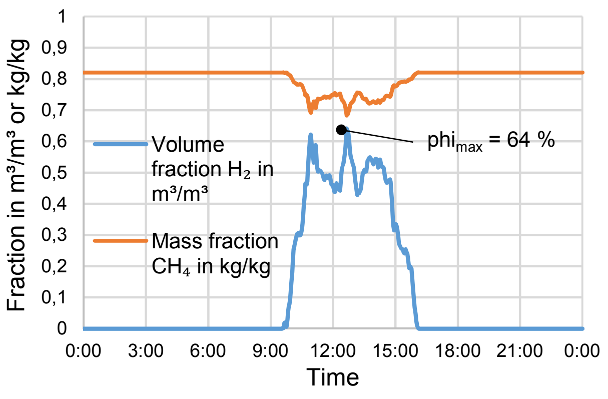


Figure 9: Hydrogen feed-in into the gas grid from PtG without control of the gas properties at a spring day

Using the same basic grid, different scenarios with various combinations of technologies (conventional heating systems, PV, CHP-systems and PtG) have been simulated including variations of the corresponding automation and control strategies.

Thus, the presented modelling approach enables a straightforward comparison of the different supply and control strategies.

Further details on the modelling of this specific grid and simulation results can be found in (Garzon-Real *et al*, 2018).

3.2 Simulation of a district heating network

Besides the coupled electrical and gas grid, the GC was used to simulate and examine a planned low temperature DHN regarding heat losses and potentials for PtH. The examined DHN is based upon the use of a heat source providing heat to achieve a supply temperature of around 25 °C all year round. This supply temperature is elevated to the necessary temperature levels for domestic hot water and room floor heating by use of heat pumps at high coefficients of power. Pipes planned for the network are regular uninsulated polyethylene pipes embedded in a filling of sand. The DHN is supposed to provide heat to a district with around 200 consumers made up of residential and commercial energy-efficient buildings.

Modelling of the heating network was carried out according to the specifications of the network and demand structure given by the project. PtH was considered in the combination of heat pumps and PV systems. For this, two scenarios of the DHN with 100 and 50 % utilization of the available roof areas for PV are investigated. Standard load profiles are used for the heat demand and a combination of measurement data and computer-generated load profiles for the electricity demand. Real-life weather data for solar radiation as well as ground and air temperature provided by the German weather service (Deutscher Wetterdienst, 2017) were taken from weather stations closest to the planned area of the DHN.

The DHN has been simulated for one year with an hourly resolution, resulting in computing times of around 6 h for a single simulation and 13 h for three parallel simulations of different scenarios. Deactivation of house pipes can reduce the simulation run time further.

Simulation of the heating network resulted in a heat demand of 6700 MWh/a with a peak load of 2500 kW (see Figure 10). The heat loss of the DHN was determined to be 25 % of the annual heat supplied to the DHN (1675 MWh/a). Analysis of the DHN simulation revealed low heat carrier velocities due to oversized pipes and thus increased heat losses.

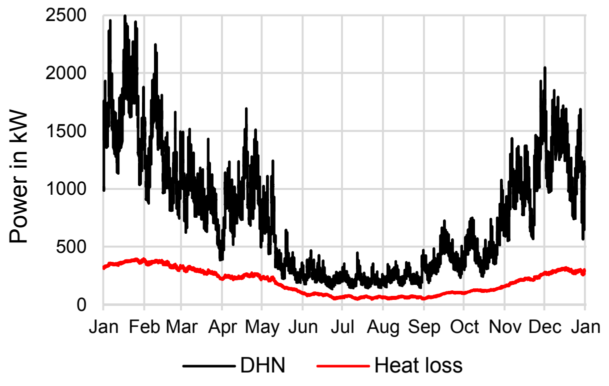


Figure 10: Simulated heat load of the DHN and corresponding heat loss profile

Complementary, the results of the simulation made it possible to determine the transition period from winter to summer as the critical time for operation of the DHN. Due to sudden increases in outside temperature during this transitional period, the heat demand of the DHN drops significantly. With lowered heat demand the mass flow through the DHN is decreased resulting in a significant cooling of the heat carrier inside the pipes. In this context, periods of high temperatures during the transition period, followed by sudden drops in temperature can lead to a DHN operation where it is not possible to immediately provide heat to the consumer until the heat carrier temperature is increased again.

As mentioned, the DHN is simulated coupled with the electrical grid to examine the potentials for PtH. The electricity demand of the district is 6600 MWh/a. At 100 % utilization of the available roof areas with PV, 27 % of the electricity demand is met. For the 50 % scenario, 13 % of the electricity demand is met by the PV systems. Due to the contradictory nature of heat demand and electricity generation of PV systems, the potential of heat pumps to lower the negative residual load is low (see Figure 11).

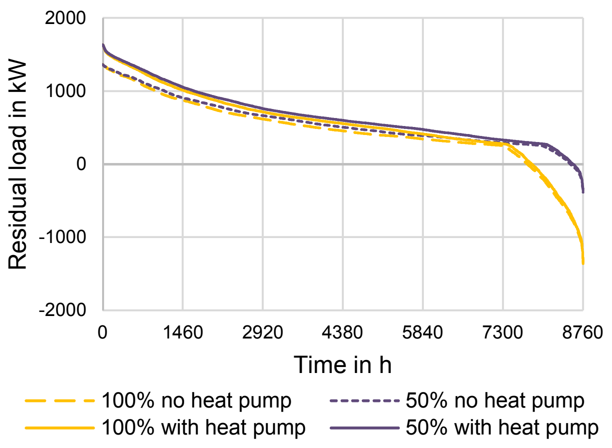


Figure 11: Sorted residual load for both roof utilization scenarios with and without heat pump electricity demand

4 Summary and Outlook

The approach of demand oriented modelling of urban frameworks and the corresponding energy grids was presented. The general modelling philosophy was explained and described in its realisation and the fundamental development ideas were shown in detail. In conclusion first examples are presented and can give a first view on the potentials of the work.

In the future several additional features will be realised.

For example, advances during the development regarding the design of the GC make it possible to further reduce the time needed for parametrisation. By use of identification integers (ID) the import of parametrisation data from Geographical Information Systems (GIS) saved inside Excel files is going to be automated. Each consumer in the desired energy grid gets assigned an ID from which data used for parametrisation as well as corresponding load profiles are imported into the GC models in Modelica. Based on this approach, the automatic generation of GC models with the help of Excel is currently being developed. Additionally, a direct import of GIS data into Modelica is conceptualised using the programming language Python.

All shown work will be published within a future release of the *TransiEnt* library.

Acknowledgements

The authors thank the German Federal Ministry for Economic Affairs and Energy for funding the project »Integrierte Betrachtung von Strom-, Gas- und Wärmesystemen zur modellbasierten Optimierung des

Energieausgleichs- und Transportbedarfs innerhalb der deutschen Energienetze - IntegraNet« (FKZ 0324027B).

References

L. Andresen, P. Dubucq, R. Peniche, G. Ackermann, A. Kather, G. Schmitz. Abschlussbericht des Verbundvorhabens: Transientes Verhalten gekoppelter Energienetze mit hohem Anteil Erneuerbarer Energien. *Technische Informationsbibliothek, Hannover*. 2017. doi: 10.2314/GBV:1002659345

M. Behnert, A. Hartke, T. Bruckner Spannungsfeld Netzstabilität – Lehren aus vergangenen Blackouts für eine sichere zukünftige Stromversorgung. *Energiewirtschaftliche Tagesfragen*, No. 9, pp. 10-14, 2018.

M. Blesl. Räumlich hoch aufgelöste Modellierung leitungsgebundener Energieversorgungssysteme zur Deckung des Niedertemperaturwärmebedarfs. *Universität Stuttgart – IER*, 2002.

Deutscher Wetterdienst (DWD). Measurements from Climate Data Center (CDC). 2017.

J. Garzon-Real, B. Dahlmann, M. Zdrallek, J. Hüttenrauch, M. Wupperfeld, J. Benthin, A. Heyer, F. Burmeister, R. Albus, W. Köppel, K. Peters. Entwicklung und Validierung eines kombinierten Strom- und Gasnetzautomatisierungs-konzepts auf Verteilnetzebene. *gwf Gas + Energie,* No 10, pp.68-81, 2018.

B. Heijde, M. Fuchs, et al. Dynamic equation-based thermo-hydraulic pipe model for district heating and cooling systems. *Energy Conversion and Management*, No. 151, pp. 158 - 169, 2017. doi: 10.1016/j.enconman.2017.08.072.

R. Hägg. Dynamic Simulation of District Heating Networks in Dymola. *Masterthesis*, Department of Energy Sciences, Lund Universitet, 2017.

IntegraNet, https://www.integranet.energy/, Fraunhofer UMSICHT, Gas- und Wärme-Institut Essen e.V., 2018.

TransiEnt, https://www.tuhh.de/transient-ee/, *Technische Universität Hamburg Harburg*, 2018.

Verein Deutscher Ingenieure. VDI 4655 Referenzlastprofile von Ein- und Mehrfamilienhäusern für den Einsatz von KWK-Anlagen. *Beuth Verlag GmbH*, 2008.

Session 1C: FMI 1

OMSimulator – Integrated FMI and TLM-based Co-simulation with Composite Model Editing and SSP

Lennart Ochel[1] Robert Braun[1] Bernhard Thiele[2] Adeel Asghar[1] Lena Buffoni[1] Magnus Eek[3] Peter Fritzson[1] Dag Fritzson[4] Sune Horkeby[5] Robert Hällquist[3] Åke Kinnander[5] Arunkumar Palanisamy[1] Adrian Pop[1] Martin Sjölund[1]

[1]PELAB – Programming Environment Lab, Dept. of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden, {lennart.ochel, robert.braun}@liu.se
[2]Institute of System Dynamics and Control, German Aerospace Center (DLR), 82234 Weßling, Germany, bernhard.thiele@dlr.de
[3]Saab AB, Bröderna Ugglas gata, SE-582 54 Linköping, Sweden
[4]SKF AB, SE-415 50 Göteborg, Sweden
[5]Siemens Turbomachinery AB, Slottsvägen, SE-612 31 Finspång, Sweden

Abstract

OMSimulator is an FMI-based co-simulation tool and recent addition to the OpenModelica tool suite. It supports large-scale simulation and virtual prototyping using models from multiple sources utilizing the FMI standard. It is integrated into OpenModelica but also available stand-alone, i.e., without dependencies to Modelica-specific models or technology. OMSimulator provides an industrial-strength open-source FMI-based modelling and simulation tool. Input/output ports of FMUs can be connected, ports can be grouped to buses, FMUs can be parameterized and composed, and composite models can be exported according to the (preliminary) SSP (System Structure and Parameterization) standard. Efficient FMI-based simulation is provided for both model-exchange and co-simulation. TLM-based tool connection is provided for a range of applications, e.g., Adams, Simulink, Beast, Dymola, and OpenModelica. Moreover, optional TLM (Transmission Line Modelling) domain-specific connectors are also supported, providing additional numerical stability to co-simulation. An external API is available for use from other tools and scripting languages such as Python and Lua. The paper gives an overview of the tool functionality, compares with related work, and presents experience from industrial usage.

Keywords: FMI, FMU, SSP, modelling, simulation, co-simulation, composite

1 Introduction

The use of virtual prototyping methods in product development has become an indispensable tool to manage the complexity of competitive modern products and industrial processes. Modelling the dynamic behaviour of such products and processes often requires considering systems that are composed of physical subsystems (usually from different physical domains) together with computing and networking. The Modelica language, which allows integrating discrete-time dynamics (e.g., control software) and continuous-time dynamics (process behaviour), is well suited for this task.

However, a frequent problem in larger industrial projects is that although component-level models are available, it is a big hurdle to integrate them into larger system simulations. This is because different development groups and disciplines, e.g., electrical, mechanical, hydraulic, and software, often use their own approaches and special purpose tools for modelling and simulation.

To improve the interoperability of behavioural models, the MODELISAR project (MODELISAR Consortium, 2011), developed the Functional Mock-up Interface (FMI) as a standardized exchange format for behavioural models. Figure 1 illustrates the basic concept: Model components are exported as Functional Mock-up Units (FMUs) from their respective discipline specific tool, another simulator tool can import the FMUs and integrate them into a Functional Mock-up using a suitable master algorithm for coupling the individual units. In October 2014, the improved version FMI 2.0 was released to the public (FMI development group, 2014).

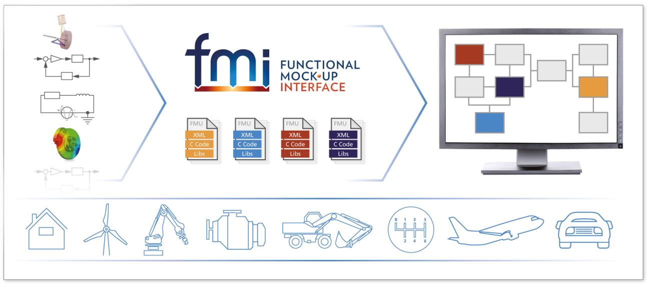


Figure 1. Model integration using FMI (source: https://www.fmi-standard.org/).

The motivation behind FMI is easily understood, however, coupling different simulator codes is a major challenge and an active research area. Modular simulation of a global system by coupling different simulator codes may easily result in an unstable integration or may require proceeding in prohibitively small time steps (Schierz and Arnold, 2012). Successful co-simulation needs:

- A suitable module interface (this is what FMI standardizes) and
- A suitable master algorithm for coupling the modules (not standardized in FMI).

In previous work, Transmission Line Modelling (TLM) was integrated as one possible approach to co-simulation in OpenModelica (Siemers et al., 2006), also considered as one approach to gain speed-up by simulation parallelization during the RTSIM project (Sjölund et al., 2010; Sjölund, 2015), however, this was not based on FMI. Additional difficulties arise if discrete-time models (e.g., control software) are included within a co-simulation setup (hybrid co-simulation). With regards to hybrid co-simulation, the latest FMI 2.0 standard was shown to have deficiencies and different proposals to amend these deficiencies were discussed, e.g., (Broman et al., 2013; Cremona et al., 2016; Tavella et al., 2016; Cremona et al., 2017).

This paper describes an industrial-strength co-simulation approach. First, it discusses FMI for co-simulation in general and then it introduces the OMSimulator tool framework in Section 3. Based on that, the graphical user interface is outlined in Section 4 and some industrial applications are discussed in Section 5.

2 FMI for Co-Simulation

The FMI 2.0 standard defines two interfaces (FMI development group, 2014, p. 4):

- FMI for Model Exchange (FMI-ME): The intention is that a modelling environment can generate C code of a dynamic system model that can be utilized by other modelling and simulation environments.
- FMI for Co-Simulation (FMI-CS): The intention is to provide an interface standard for coupling simulation tools in a co-simulation environment.

The two interfaces share common parts and concepts, in particular:

- FMI C-application programming interface (API): All computations are evaluated by calling standardized C-functions.
- FMI Extensible Markup Language (XML) description schema: The schema describes the structure and content of an XML file (named modelDescription.xml) generated by the modelling environment which exports an FMU. This modelDescription.xml file contains the definition of all variables and other structural information of an FMU in a standardized form.
- An FMU is delivered as a zip file which contains the XML description file, the code that provides the C-API either in binary form as shared library or as source code, as well as potential additional resources, e.g., tables, model icon, and documentation.

Basically, FMI-ME differs from FMI-CS in that it requires the importing tool to provide a numerical solver for simulating the FMU. Such solvers require vectors for states, derivatives and zero-crossing functions which are exposed by the FMI-ME API. By contrast, FMI-CS does not require the importing tool to provide a numerical solver. Instead, all required solvers are embedded within the FMI-CS and the related information is not exposed by the FMI-CS API.

2.1 FMI-based Co-Simulation

An FMI-based composite model for co-simulation can be constructed with both co-simulation and model-exchange FMUs. The building blocks determine certain constraints of the composite model structure. A straightforward derived structure from the FMI specification is given in Table 1.

Table 1. Overview of co-simulation building blocks.

solver	**components**
master algorithm	co-simulation units • CS-FMU • integrator + set of ME-FMUs
integration method	set of ME-FMUs

The master algorithm forces the so-called global time steps, which are used to exchange information between co-simulation units. Each co-simulation unit takes its own local time steps to reach the next forced global time step.

A co-simulation unit can be composed of a set of ME-FMUs. In this case, these ME-FMUs can communicate with a higher exchange rate than the global time step, basically at each local time step.

2.1.1 Initialization

Initialization must be performed within a dedicated initialization mode. A consistent initial state is computed based on the dependency information provided by the FMUs (optional FMI feature) and the actual connections between the FMUs. First, all parameters will be set to either predefined values or explicitly overwritten by the user's input. The same applies to start values, which might be crucial for internal nonlinear systems and external algebraic loops. After that, all the information is propagated based on the dependency information.

2.1.2 Simulation

The continuous simulation is performed by a master-algorithm which synchronises all co-simulation units and exchanges information between them based on internal output-input dependencies and external input-output dependencies. The continuous simulation gets interrupted if an event is detected or the final simulation time is reached.

2.1.3 Event handling

The discrete event simulation takes place if discrete changes are detected. Co-simulation FMUs cannot expose internal events, which means that only discrete changes in output variables at communication time points can be detected. In that case, the changes are propagated and a new consistent model state is computed. This might be an iterative process in case of algebraic loops.

The situation for model-exchange FMUs is a bit different. A set of connected exchange-FMUs are simulated using a shared solver and events can be processed and communicated within this set of FMUs directly when they occur.

2.2 Numerically Stable Co-Simulation

Co-simulation requires different parts of the complete model to be solved separately by isolated solvers. This will inevitably delay the interchanged variables to the next communication step. Such delays may affect numerical stability and simulation accuracy.

In many cases, a master algorithm with fixed communication step size is used and the step size is reduced until the results appear to be stable for the given problem. This is performance consuming and can only ensure stability in the observed working points. More sophisticated solutions include adaptive communication step-size (Schierz et al., 2012) or relaxation techniques (Schweizer et al., 2016). Such methods typically rely on rollback mechanisms, which are often not available (state serialization in FMUs is optional).

One technique that addresses this issue is TLM (Krus, 2011). Every physical element has a finite information propagation speed. By mapping the physically motivated delays to the communication points in the model, artificial time delays can be avoided. As a result, the stability properties of the simulation model will reflect the stability properties of the physical system it represents. In other words, the separation into different solvers will not affect the numerical stability of the complete model. The TLM implementation in OMSimulator is based on previous work by SKF (Siemers et al., 2009; Fritzson et al., 2018). The boundary equations for a TLM connection are shown in Equation 1 and 2:

$$e_1(t) = e_2(t-\Delta t) + Z_c\left[f_1(t) + f_2(t-\Delta t)\right] \tag{1}$$

$$e_2(t) = e_1(t-\Delta t) + Z_c\left[f_2(t) + f_1(t-\Delta t)\right] \tag{2}$$

e_1, e_2:	effort variables
f_1, f_2:	flow variables
Z_c:	characteristic impedance
Δt:	time delay

It can be noted that the effort variable on one side of the connection is always independent of variables on the other side within a (usually small) time frame of Δt, during which solvers on both sides can work independently.

With FMI for co-simulation, sub-models can only exchange variables at communication time points. This induces sampling errors, which greatly reduces the benefits of TLM. OMSimulator addresses this by supporting interpolation, either by sending derivatives of the input signals or by providing the sub-models with interpolation tables (Braun et al., 2017b). Based on the assumption that sampling errors arise from aliasing, a related solution could be to use anti-aliasing filters (Benedikt et al., 2013; Drenth, 2017). Another solution based on increasing communication step size using context-based extrapolation was proposed by (Khaled et al., 2014). Both interpolation and anti-aliasing features would greatly benefit from callback functions for writing intermediate outputs and requesting intermediate inputs. This improvement has been suggested to the FMI design group.

3 OMSimulator Tool Framework

OMSimulator is a unified co-simulation tool that supports FMI 2.0 for model exchange and co-simulation. One of its unique features is the support of TLM for numerically stable co-simulation. Simulations can be performed as soft real-time or offline simulations.

3.1 Main Framework Aspects

OMSimulator is developed as a standalone open-source simulation library with a rich C-API. The integration into the OpenModelica graphical editor OMEdit demonstrates how the C-API can be utilized for providing an intuitive (graphical) user experience. Additionally, OMSimulator provides a command-line interface (CLI) and scripting interfaces for Python and Lua. These different interfaces can be used to integrate OMSimulator into third-party tools and specialized applications, e.g. flight simulators and optimization applications.

The open-source implementation enables research on various co-simulation questions, e.g. dependency-graph-based master algorithms for parallel and multi-rate execution of FMI components.

3.2 Simulation Architecture

Composite models are constructed as a tree of certain building blocks. The root node is either a TLM system, weakly-coupled system (WC system), or strongly-coupled system (SC system). The systems differ in the way connections are handled:

- **TLM** systems contain TLM connections, which can basically be considered as physical-motivated delayed connections.

- **Weakly-coupled** systems are used for actual co-simulation. All simulation units run independently

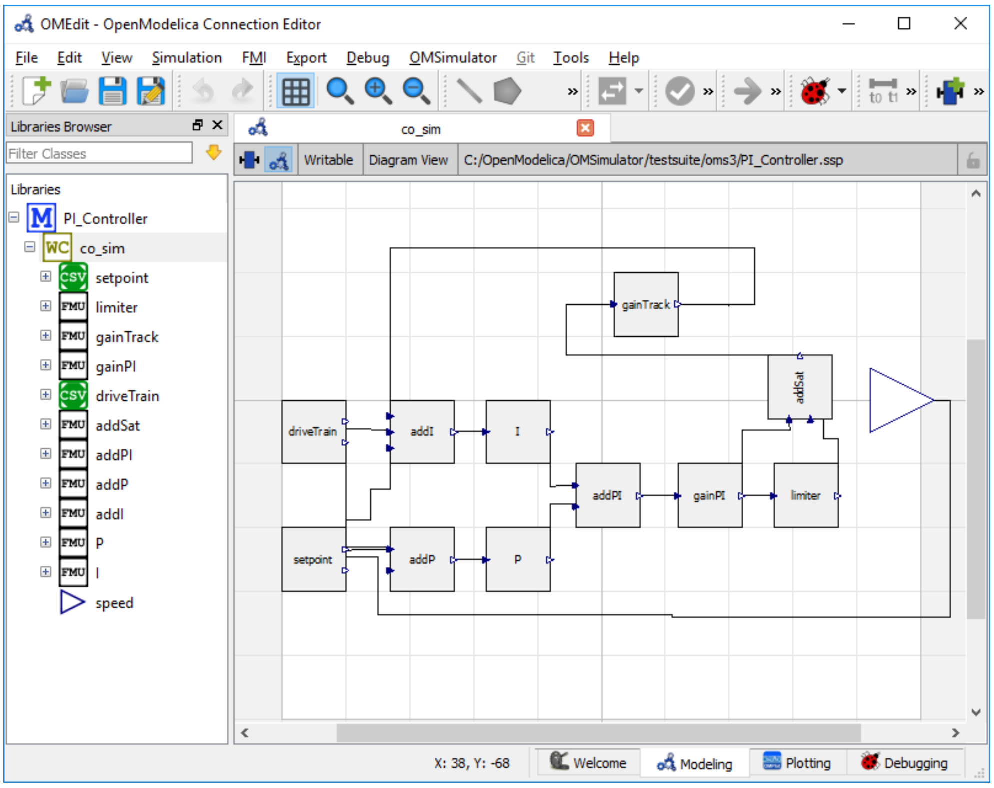


Figure 2. PI controller model created from 9 FMUs and connected to 2 lookup tables for the boundary conditions.

and are synchronized by a master algorithm at certain communication time points.

- **Strongly-coupled** systems are used to wrap-up model exchange FMUs into a co-simulation unit. They share a common solver and use a continuous communication schema.

A system can contain other systems, components (i.e. FMUs or lookup tables), connectors, and buses.

4 Graphical User Interface with Composite Model Editor

A graphical user interface has been developed as an extension to the existing OpenModelica Connection Editor (OMEdit) (Asghar and Tariq, 2010) and the composite model editor presented in (Mengist et al., 2015). OMEdit communicates with OMSimulator through the C-API for visual composite modelling.

4.1 Visual Modelling

The graphical user interface allows the user to create composite models and add systems, components (FMUs, tables and external models), connectors, buses and connections to the model. Each composite model is displayed in the form of a hierarchical tree as shown in the left column of Figure 2.

Each element in the hierarchical tree consists of an icon, diagram and text view except for the top-level model, connectors and buses. The model element does not have an icon view and the connectors and buses are non-editable shapes.

A user can create a connection between two connectors or between two buses. A bus or a TLM bus consists of a list of connectors. When a connection between two buses is made, a bus connection dialog is shown (see Figure 3). The dialog maps the inputs and outputs of the buses automatically. This allows making connections for large systems trivial.

4.2 Simulation and Post Processing

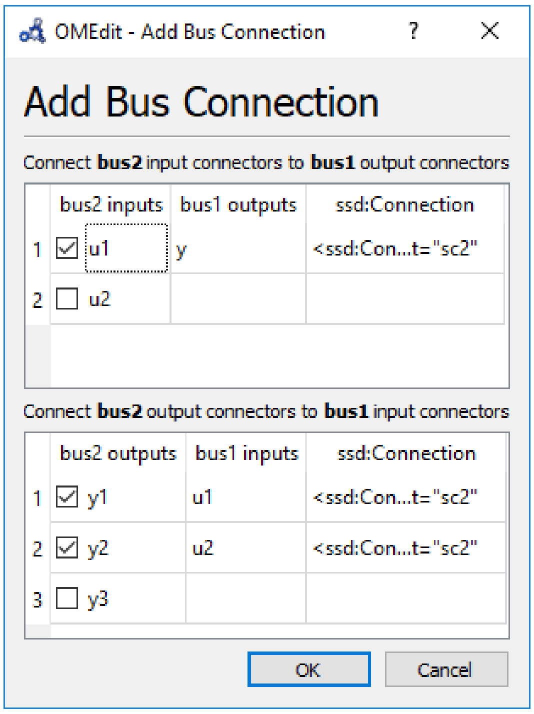
The model needs to be in the instantiated state before performing the simulation. Once the model enters into the instantiation phase the user can set the FMU parameters and start the simulation. The user interface shows the simulation status and progress using the callback functions from the C-API. The simulation results are visualized in the plotting perspective of OMEdit as shown in Figure 4.

5 Industrial Applications and Benchmarks

In this section, several industrial applications are presented.

5.1 Saab Use Case

Analysing and designing sub-systems separately is not enough in modern aircraft development. A competitive product needs to be developed considering the joint behaviour of tightly coupled sub-systems in order to avoid



OMEdit - Add Bus Connection

Add Bus Connection

Connect **bus2** input connectors to **bus1** output connectors

	bus2 inputs	bus1 outputs	ssd:Connection
1	☑ u1	y	<ssd:Con...t="sc2"
2	☐ u2		

Connect **bus2** output connectors to **bus1** input connectors

	bus2 outputs	bus1 inputs	ssd:Connection
1	☑ y1	u1	<ssd:Con...t="sc2"
2	☑ y2	u2	<ssd:Con...t="sc2"
3	☐ y3		

Figure 3. Bus connection.

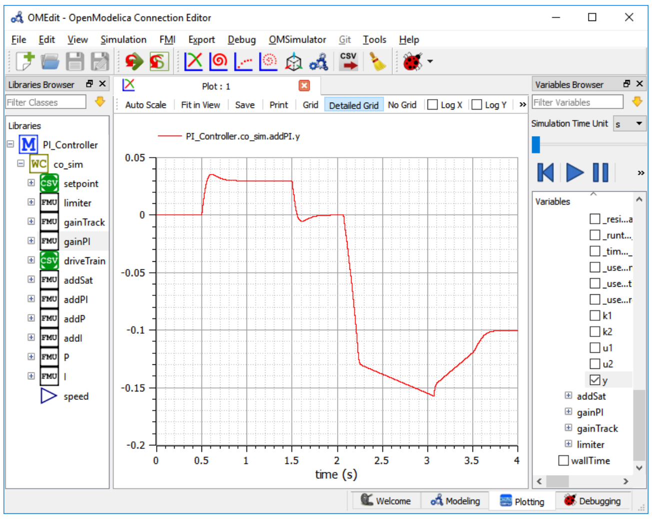


Figure 4. Simulation result.

sub-optimization as well as to achieve the desired high level of aircraft integration. Engineers and researchers, therefore, need to have the means of detailed analysis using coupled simulation models, developed in a wide variety of different domain-specific tools, available on their desktop computers. Scalable, numerically stable, and distributed simulations need to be achieved while preventing tool vendor lock-in effects as well as minimizing licensing costs (Hällqvist et al., 2018).

A detailed aircraft vehicle systems simulator is developed throughout the OpenCPS project. The simulator aims to serve as an industrially relevant platform for testing standardized methods for connecting and simulating models from different tools in the OMSimulator as well as other integrating simulation tools. The aircraft systems simulator is developed in parallel to the OMSimulator, continuously exposing industrial needs and requirements that were not captured during the master simulation engine specification phase (OpenCPS project partners, 2016). An early prototype of the aircraft vehicle systems simulator was presented in (Hällqvist et al., 2017). The simulator was further developed and expanded to enable studies of pilot thermal comfort connected to Environmental Control System (ECS) performance (Hällqvist et al., 2018; Schminder et al., 2018). The latter combines the domains of hardware, software, and human factors modelling. Two different composite models of the same system were created: one using only traditional connections between FMUs, referred to as an FMI composite model, and one with only TLM type connections, referred to as a TLM composite model.

A schematic description of the different included subsystems is presented in Figure 5. The simulator includes an engine model designed to provide the included ECS with air at high temperature and pressure depending on the aircraft boundary conditions. The boundary conditions are expressed by the aircraft operational point along with outputs from the included atmosphere model. In turn, the ECS provides its consumers with conditioned air at the correct mass flow, temperature, and pressure. The specified mass flows, temperatures, and pressures are achieved via a total of five modelled motorized valves controlled by a modelled software, denoted ECS Control in the figure. The included consumers are a thermoregulatory cockpit model, described in detail by Schminder et al. in (Schminder et al., 2016), along with two simple place-holder consumers representing subsystems requiring air cooling and/or pressurization. The cockpit model provides necessary inputs to the included pilot comfort model which incorporates numerous well-established comfort measures into the simulation, such as the Fighter Index of Thermal Stress (Nunneley and Stribley, 1979). The Engine, ECS, and ECS Control models are expressed using the Modelica language whereas the atmosphere, cockpit, and pilot comfort models are developed in Matlab/Simulink (MathWorks). All models are exported as FMUs for co-simulation, the Modelica models using Dymola (Dassault Systemes AB), and the Matlab/Simulink models using the Dassault developed toolbox FMI Kit for Simulink. The Modelica models are all exported with the variable order and variable step solver CVODE (Lawrence Livermore National Laboratory) whereas the Matlab/Simulink models are exported with fixed step solvers.

In (Braun et al., 2017b), different approaches to establishing interoperability between FMI for Co-Simulation and TLM were developed and evaluated. The most suited approach of using callback functions for FMUs to request inputs at the times they are needed is not possible with FMI 2.0. One feasible workaround is to use fine-grained

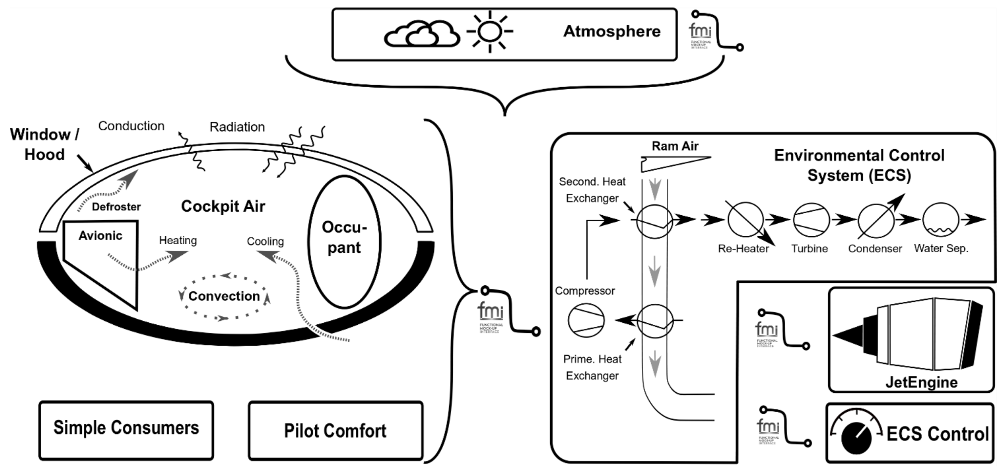


Figure 5. Schematic overview of an aircraft systems simulator comprising detailed sub-system simulation models.

interpolation inside FMUs, see Section 2.2. The mission simulation presented here serves as an industry grade verification test-case for the method of using fine-grained interpolation, in the OMSimulator, to ensure numerical stability during transient conditions.

A subset of the mission boundary conditions are presented in Figure 6, the altitude corresponds to the left-hand side y-axis and the Mach number to the right-hand side y-axis. The ECS consumer supply pressure is plotted for simulations using only TLM connections (Red) and using only traditional native FMI connections (Black) in Figure 7. The results are similar and the discrepancies are likely a result of aliasing effects resulting from a slightly too long communication interval in the native FMI simulation. The main difference between the presented simulations is that all included FMUs are executed in parallel, using physically motivated delays, for the TLM composite model. In contrast to the native FMI zero-order-hold sampling resulting in constant input to FMUs during each master step, the TLM solution guarantees the availability of interpolated input data at the discretion of each FMU's internal solver. For this particular composite model example, the TLM parallelization does not decrease the simulation execution time compared to the native FMI simulation. The main reason for this is that the composite model is not well structured from a parallelization perspective, and thus the FMU representing the ECS physical system clearly dominates the computational effort. In addition, the native FMI simulation is tuned to use the largest communication interval possible challenging the numerical stability of the master simulation, whereas the communication interval in the TLM simulation is based on the real physics and does not compromise numerical stability. Further up-scaling by adding FMUs for other aircraft subsystems, such as a fuel system, a hydraulic system, and an auxiliary power unit, would reveal the scalability benefits, in terms of execution time, of the TLM solution.

The presented use case demonstrates the OMSimulator as an industrially relevant open-source alternative or complements to existing FMI-supporting master simulation tools in aircraft vehicle systems applications. Combining the TLM technique with the more traditional method of simulating coupled FMUs is a most promising and flexible approach for scalable, numerically stable and accurate, distributed simulation. The use case shows that combining models from multiple modelling and simulation domains, from both industry and academia, is feasible using the OMSimulator. In (Hällqvist et al., 2018) and (Schminder et al., 2018), the focus is placed on studies relating ECS performance to pilot thermal comfort. Other possible areas of application are various optimization studies, e.g., minimizing the engine air consumed by the ECS while maximizing pilot comfort.

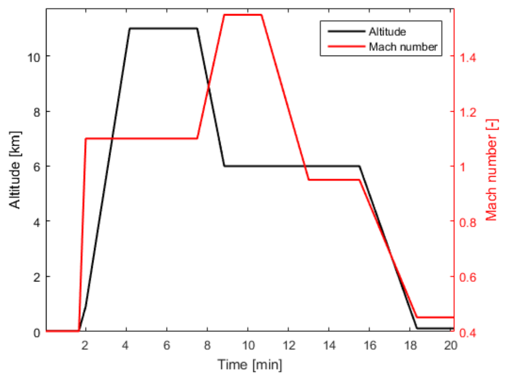


Figure 6. A subset of the simulation boundary conditions, Altitude and Mach number.

5.2 Energy Demonstrator

Driven by the need to limit global warming, the energy systems worldwide are in a phase of expanding renewable energy as an alternative to conventional power plants.

The design and control of combined cycle power plants are expected to become increasingly more important as a method of balancing the electric networks with a large share of renewable energy input. This demonstrator is mo-

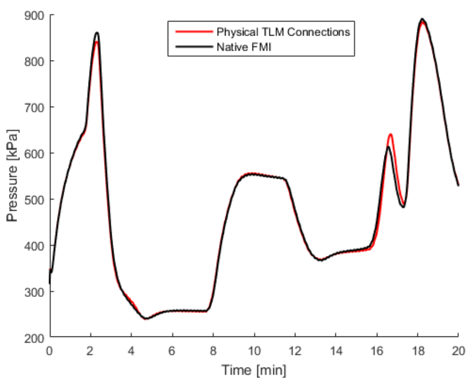


Figure 7. ECS supply pressure to included fuel system model. Results from the TLM Composit model simulation is depictd as red and the Native FMI Composite model simulation as black.

tivated by the need to enable suppliers, in an early design phase, to test the complete functionality by utilizing well-verified models from different sources, without having to convert all models to run by the same tool. This is needed for the entire electrical grid.

The project goal of this joint energy demonstrator was to combine FMUs from four different suppliers, to show that each supplier's verified knowledge, expressed by their FMU, could be used for design, and transient analysis of a power plant.

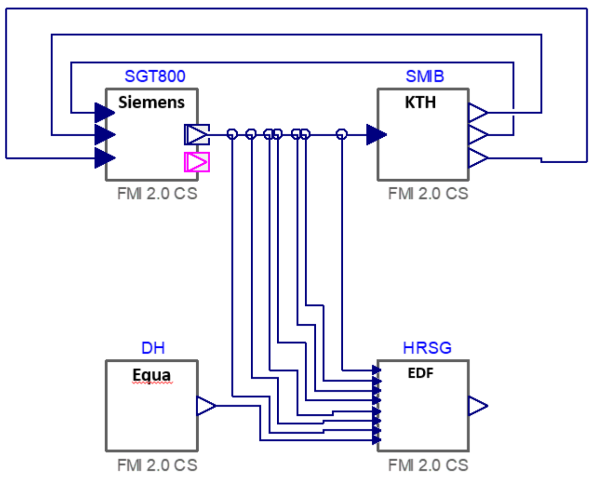


Figure 8. Energy demonstrator of a combined cycle power station with detailed accurate models (total over 30000 equations) from different suppliers, provided as FMUs.

The power plant in Figure 8 is a combined cycle plant (CCPP) with steam extraction to a district heating system. The FMUs are a gas turbine (GT) supplying flue gases to a heat recovery steam generator (HRSG) that supplies steam to a district heating system (DH). The GT shaft drives a generator connected to a large utility network, the model named SMIB. The HRSG also supplies steam to a steam turbine that is included in the HRSG model.

Following entities supplies FMUs for the CCPP:

1. Siemens Industrial Turbomachinery AB supplies the GT
2. KTH supplies the net model
3. EDF supplies the HRSG with ST
4. Equa AB supplies the DH

The simulation results of the generator power during GT start-up from the model shown in Figure 8 depend on communication interval and error tolerance. To achieve accurate results, the simulation settings need to be tightened up which increases the simulation time dramatically. This encouraged us to develop further advanced simulation technologies, such as a master-algorithm with variable step size and input extrapolation based on output derivative information.

OMSimulator has the capability for early multi-domain simulations in the design and configuration phase but also supporting behaviour control in the operational and recycling phase and support closed loops for a sustainable environment.

With this new technology and with the promising test results we will be able to support the vision of sustainable zero emission power plants with optimized solutions and also bridge technologies from different partners.

5.3 SKF 3D Mechanical Demonstrator

Models of 3D mechanics typically contain stiff equations and short time constants. This makes them especially sensitive to delayed variables and thereby poses an interesting challenge for co-simulation. To demonstrate the stability benefits of TLM, a model of a hydraulic crane with two actuators was developed, see Figure 9. The intention is to simulate a model of a roller bearing from SKF together with the surrounding system for achieving accurate boundary conditions. SKF is one of the world's largest suppliers of bearings and has a great interest in simulating their bearing models together with models from customers. This model constitutes a typical scenario, where a system model developed by a customer is connected to a bearing model developed by the supplier. An early prototype of the demonstrator was presented in (Braun et al., 2017a). Table 2 shows an overview of the sub-models in the composite model. All mechanical bodies are modelled in Dymola and exported as FMUs. The crane arms are connected through a roller bearing modelled in SKF BEAST (Fritzson et al., 2014, 2018). The crane mechanics is modelled using rigid bodies, while the bearing model contains flexible bodies and contact mechanics. Motion is controlled by a hydraulic system modelled in Hopsan, a system simulation tool specialized for hydraulic and mechatronic systems developed by Linköping University (Axin et al., 2010). Experiments show that the model works well with FMI for model exchange. With FMI for co-simulation, either callback functions or fine-grained interpolation (see section 2.2) are required to achieve stable

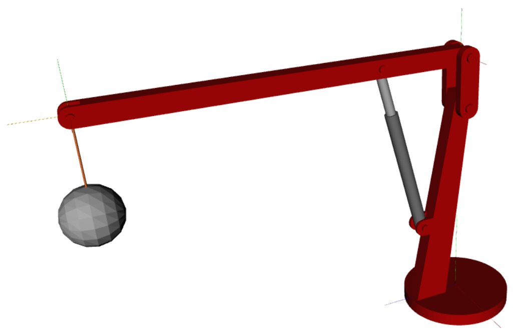


Figure 9. The SKF crane demonstrator model is used to verify stability in 3D connections.

results. When using sampled inputs with zero-order hold, stability cannot be achieved even when using a step-size 1000 times smaller than the other methods. Results and performance cannot be compared to a monolithic implementation because there is no tool capable of simulating all three parts of the model: the bearing, the crane and the hydraulic system. Nevertheless, results are fully realistic. The motion agrees well with simplified models. Static forces and torques all have correct magnitudes. No unaccountable phenomena have been observed.

Table 2. Overview of the different sub-models in the SKF demonstrator model.

Sub-model	Tool
Boom	FMU (Dymola)
Jib	FMU (Dymola)
Load	FMU (Dymola)
Piston	FMU (Dymola)
Bearing	BEAST
Hydraulics	Hopsan
Controller	Hopsan

6 Related Work

Table 3 compares related tools and libraries that also support similar co-simulation functionality. There are both commercial and free-of-charge solutions available with different licences. All tools support both model-exchange and co-simulation FMUs. PyFMI is the only tool which does not include built-in support for lookup tables. This feature is quite handy, but not critical because it can be isolated and solved by dedicated FMUs.

Except for Simulink, all tools support handling of algebraic loops. In Simulink, however, it is not possible to execute such composite models. Introducing a delayed signal can circumvent this issue, but is not considered as an appropriate solution since it introduces unintended dynamics.

All the tools except FMI Composer (Modelon), provide some kind of scripting interface. DACCOSIM (Virginie et al., 2015), Simulink and Dymola have proprietary solutions and OMSimulator, PySimulator (Pfeiffer et al., 2012; Asghar et al., 2015), FMI Go! (Lacoursière and Härdin, 2017), and PyFMI (Christian et al., 2016) are based on open scripting languages.

OMSimulator uses the upcoming SSP standard as an exchange format for composite models, as well as FMI Go!, and FMI Composer.

7 Conclusions

OMSimulator 2.0 is part of the OpenModelica 1.13.0 release and also available as a standalone application. It provides the following functionality. It supports both FMI variants, i.e. model-exchange and co-simulation. It supports also the TLM technique to decouple co-simulation units and potentially stabilize the simulation. TLM connections enable direct tool-coupling as well, e.g. with Adams, Beast, and Simulink.

The OpenModelica graphical editor OMEdit is connected to OMSimulator via a C-API and provides a rich user experience.

The SSP standard, which is still under development, is supported as an early prototype to enable exchanging models with an open and independent standard. As an extension to the current SSP version, signal grouping and bus connections are supported and integrated into the SSP using annotations.

Compared to other tools, OMSimulator has outstanding features like TLM and SSP support. The open-source implementation facilitates use by academics and also in industry. It can be used as a research platform for co-simulation.

Acknowledgements

This work has been supported by Vinnova in the ITEA OPENCPS, and EMPHYSIS projects and in the Vinnova RTISIM project. Support from the Swedish Government has been received from the ELLIIT project. The OpenModelica development is supported by the Open Source Modelica Consortium. Many students, researchers, and engineers have contributed to the OpenModelica system. There is not room here to mention all these people, but we gratefully acknowledge their contributions.

References

Adeel Asghar, Andreas Pfeiffer, Arunkumar Palanisamy, Alachew Mengist, Martin Sjölund, Adrian Pop, and Peter Fritzson. Automatic regression testing of simulation models and concept for simulation of connected FMUs in PySimulator. In Fritzson and Elmqvist (2015). doi:10.3384/ecp15118671.

Syed Adeel Asghar and Sonia Tariq. Design and implementation of a user friendly OpenModelica graphical connection editor. Master's thesis, Linköping University, Department of Computer and Information Science, December 2010. URL `http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-65864`.

Table 3. Comparison of related tools.

	OMSimulator	DACCOSIM	Simulink	PyFMI
Commercial	No	No	Yes	No
Open-source	OSMC-PL, GPL	AGPL2	No	LGPL
Lookup Table	Yes	Yes	Yes	No
Alg. Loops	Yes	Yes	No	Yes
Scripting	Python, Lua	proprietary	proprietary	Python
GUI	Yes	Yes	Yes	No
SSP	Yes	No	No	No
platform	Linux/Win/macOS	Linux/Win	Linux/Win/macOS	Linux/Win/macOS

	Dymola	PySimulator	FMI Go!	FMI Composer
Commercial	Yes	No	No	Yes
Open-source	No	BSD	MIT	No
Lookup Table	Yes	Yes	Yes	Yes
Alg. Loops	Yes	Yes	Yes	Yes
Scripting	proprietary	Python	Go	No
GUI	Yes	Yes	No	Yes
SSP	No	No	Yes	Yes
platform	Linux/Win	Linux/Win	Linux/Win/macOS	Linux/Win/macOS

Mikael Axin, Robert Braun, Alessandro Dell'Amico, Björn Eriksson, Peter Nordin, Karl Pettersson, Ingo Staack, and Petter Krus. Next generation simulation software using transmission line elements. In *Fluid Power and Motion Control*, Bath, England, September 2010.

Martin Benedikt, Daniel Watzenig, and Anton Hofer. Modelling and analysis of the non-iterative coupling process for co-simulation. *Mathematical and Computer Modelling of Dynamical Systems*, 19(5):451–470, 2013. doi:10.1080/13873954.2013.784340.

Robert Braun, Adeel Asghar, Adrian Pop, and Dag Fritzson. An open-source framework for efficient co-simulation of fluid power systems. In *Proceedings of 15th Scandinavian International Conference on Fluid Power, June 7-9, 2017*, number 144, pages 393–400, Linköping, Sweden, June 2017a. Linköping University Electronic Press, Linköpings universitet.

Robert Braun, Robert Hällqvist, and Dag Fritzon. TLM-based Asynchronous Co-simulation with the Functional Mockup Interface. In *IUTAM Symposium on Solver Coupling and Co-Simulation*, Darmstadt, Germany, September 2017b.

David Broman, Christopher Brooks, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of fmus for co-simulation. Technical Report UCB/EECS-2013-153, EECS Department, University of California, Berkeley, Aug 2013. URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-153.html`.

Andersson Christian, Åkesson Johan, and Führer Claus. PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface. Technical Report 2, Centre for Mathematical Sciences, Lund University, 2016. URL `https://lup.lub.lu.se/search/publication/961a50eb-e4a8-43bc-80ac-d467eef26193`.

Fabio Cremona, Marten Lohstroh, Stavros Tripakis, Christopher Brooks, and Edward A. Lee. Fide: An fmi integrated development environment. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 1759–1766, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3739-7. doi:10.1145/2851613.2851677.

Fabio Cremona, Marten Lohstroh, David Broman, Edward A. Lee, Michael Masin, and Stavros Tripakis. Hybrid co-simulation: it's about time. *Software & Systems Modeling*, November 2017. ISSN 1619-1374. doi:10.1007/s10270-017-0633-6.

Dassault Systemes AB. Dymola. URL `https://www.3ds.com/products-services/catia/products/dymola/`.

Edo Drenth. Method and system for control and co-simulation of physical systems, March 2 2017. US Patent App. 15/232,261.

FMI development group. Functional Mock-up Interface for Model Exchange and Co-Simulation v2.0. Modelica Association Project "FMI", October 2014. URL `https://www.fmi-standard.org/`. Standard Specification.

Dag Fritzson, Lars-Erik Stacke, and Jens Anders. Dynamic simulation–building knowledge in product development. *Evolution*, 1, 2014.

Dag Fritzson, Robert Braun, and Jan Hartford. Composite modelling in 3-d mechanics utilizing transmission line modelling (tlm) and functional mock-up interface (fmi). 2018.

Peter Fritzson and Hilding Elmqvist, editors. *Proceedings of the 11th International Modelica Conference*, September 2015. Modelica Association and Linköping University Electronic Press. doi:10.3384/ecp15118.

Robert Hällqvist, Rober Braun, and Petter Krus. Early Insights on FMI-based Co-Simulation of Aircraft Vehicle Systems. In

Proceedings of the 15th Scandinavian International Conference on Fluid Power, Linköping, Sweden, 2017.

Robert Hällqvist, Jörg Schminder, Magnus Eek, Robert Braun, Roland Gårdhagen, and Peter Krus. A novel FMI and TLM-based simulator for detailed studies of thermal pilot comfort. In *Proceedings of the 31st Congress of the International Council of the Aeronautical Sciences (ICAS)*, Belo Horizonte, Brazil, 2018.

Abir Ben Khaled, Laurent Duval, Mohamed El Mongi Ben Gaïd, and Daniel Simon. Context-based polynomial extrapolation and slackened synchronization for fast multi-core simulation using fmi. In *International Modelica Conference*, pages 225–234. Linköping University Electronic Press, 2014.

Petter Krus. Robust modelling using bi-lateral delay lines for high speed simulation of complex systems. In *DINAME 2011: 14th International Symposium on Dynamic Problems in Mechanics*, 2011. URL `http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-67897`. Invited conference contribution.

Claude Lacoursière and Tomas Härdin. Fmi Go! A Simulation runtime environment with a client server architecture over multiple protocols. In *12th Int. Modelica Conference*, Prague, Czech Republic, 2017. URL `https://www.modelica.org/events/modelica2017/proceedings/html/submissions/ecp17132653_LacoursiereHardin.pdf`.

Lawrence Livermore National Laboratory. SUNDIALS: SUite of Nonlinear and Differential/ALgebraic Equation Solvers. URL `https://computation.llnl.gov/projects/sundials/cvode`.

MathWorks. Simulink. URL `https://www.mathworks.com/products/simulink.html`.

Alachew Mengist, Adeel Asghar, Adrian Pop, Peter Fritzson, Willi Braun, Alexander Siemers, and Dag Fritzson. An open-source graphical composite modeling editor and simulation tool based on fmi and tlm co-simulation. In Fritzson and Elmqvist (2015). doi:10.3384/ecp15118181.

MODELISAR Consortium. MODELISAR - From System Modeling to S/W running on the Vehicle, 2011. URL `https://itea3.org/project/modelisar.html`.

Modelon. FMI COMPOSER. URL `https://www.modelon.com/products-services/modelon-deployment-suite/fmi-composer/`.

Sarah Nunneley and Richard Stibley. Fighter Index of Thermal Stress: Development of Interim Guidance for Hot-Weather USAF Operations. *Journal of the American Society of Heating and Ventilating Engineers*, 50:639–642, 1979.

OpenCPS project partners. FMI Master Simulation Tool Requirement Specification, December 2016. URL `https://itea3.org/project/opencps.html`.

Andreas Pfeiffer, Matthias Hellerer, Stefan Hartweg, Martin Otter, and Matthias Reiner. PySimulator – A Simulation and Analysis Environment in Python with Plugin Infrastructure. In *9th Int. Modelica Conference*, Munich, Germany, 2012. URL `http://www.ep.liu.se/ecp/076/053/ecp12076053.pdf`.

Tom Schierz and Martin Arnold. Stabilized overlapping modular time integration of coupled differential-algebraic equations. *Applied Numerical Mathematics*, 62(10):1491 – 1502, 2012. ISSN 0168-9274. doi:10.1016/j.apnum.2012.06.020.

Tom Schierz, Martin Arnold, and Christoph Clauß. Co-simulation with communication step size control in an FMI compatible master algorithmnak. In *9th Int. Modelica Conference, Munich, Germany*, pages 205–214, 2012.

Jörg Schminder, Roland Gårdhagen, Elias Nilsson, Karl Storck, and Matts Karlsson. Development of a Cockpit-Pilot Model for Thermal Comfort Optimization During Long-Mission Flight. In *Proceedings of the AIAA Modeling and Simulation Technologies Conference*, 2016.

Jörg Schminder, Robert Hällqvist, Magnus Eek, and Roland Gårdhagen. Pilot performance and heat stress assessment support using a cockpit thermoregulatory simulation model. In *Proceedings of the 31st Congress of the International Council of the Aeronautical Sciences (ICAS)*, Belo Horizonte, Brazil, 2018.

Bernhard Schweizer, Daixing Lu, and Pu Li. Co-simulation method for solver coupling with algebraic constraints incorporating relaxation techniques. *Multibody System Dynamics*, 36(1):1–36, 2016. ISSN 1573-272X. doi:10.1007/s11044-015-9464-9.

Alexander Siemers, Dag Fritzson, and Peter Fritzson. Meta-Modeling for Multi-Physics Co-Simulation applied for OpenModelica. In *International Congress on Methodologies for Emerging Technologies in Automation (ANIPLA2006)*, Rome, Italy, November 13–15 2006.

Alexander Siemers, Dag Fritzson, and Iakov Nakhimovski. General meta-model based co-simulations applied to mechanical systems. *Simulation Modelling Practice And Theory*, 17(4):612–624, 2009. doi:doi:10.1016/j.simpat.2008.10.006.

Martin Sjölund. *Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models*. Doctoral thesis No 1664, Linköping University, Department of Computer and Information Science, 2015.

Martin Sjölund, Robert Braun, Peter Fritzson, and Petter Krus. Towards efficient distributed simulation in modelica using transmission line modeling. In *3rd International Workshop on Equation-Based Object-Oriented Languages and Tools.*, Oslo, Norway, October 2010. URL `http://www.ep.liu.se/ecp/047/`.

J. P. Tavella, M. Caujolle, S. Vialle, C. Dad, C. Tan, G. Plessis, M. Schumann, A. Cuccuru, and S. Revol. Toward an accurate and fast hybrid multi-simulation with the fmi-cs standard. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–5, September 2016. doi:10.1109/ETFA.2016.7733616.

Galtier Virginie, Vialle Stephane, Dad Cherifa, Jean-Philippe Tavella, Lam-Yee-Mui Jean-Philippe, and Plessis Gilles. Fmi-based distributed multi-simulation with daccosim. pages 39–46, 2015. URL `http://dl.acm.org/citation.cfm?id=2872965.2872971`.

FMU-proxy: A Framework for Distributed Access to Functional Mock-up Units

Lars Ivar Hatledal[1] Houxiang Zhang[1] Arne Styve[2] Geir Hovland[3]

[1]Department of Ocean Operations and Civil Engineering, NTNU, Norway, {laht,hozh}@ntnu.no
[2]Department of ICT and Natural Sciences, NTNU, Norway, asty@ntnu.no
[3]Department of Engineering Sciences, UiA, Norway, geir.hovland@uia.no

Abstract

The main goal of the Functional Mock-up Interface (FMI) standard is to allow simulation models to be shared across tools. To accomplish this, FMI relies on a combination of XML-files and compiled C-code packaged in a zip archive. This archive is called an Functional Mock-up Unit (FMU) and uses the extension *.fmu*. In theory, an FMU can support multiple platforms, however this is not always the case and depends on the type of binaries the exporting tool was able to provide. Furthermore, a library providing FMI support may not be available in a particular language, and/or it may not support the whole standard. Another issue is related to the protection of Intellectual Property (IP). While an FMU is free to only provide the C-code in binary form, other resources shipped with the FMU may be unprotected.

In order to overcome these challenges, this paper presents FMU-proxy, an open-source framework for accessing FMUs across languages and platforms. This is done by wrapping one or more FMUs behind a server program supporting multiple language independent Remote Procedure Call (RPC) technologies over several network protocols. Currently, Apache Thrift (TCP/IP, HTTP), gRPC (HTTP/2) and JSON-RPC (HTTP, WebSockets, TPC/IP, ZeroMQ) are supported. Together, they allow FMUs to be invoked from virtually any language on any platform. As users don't have direct access to the FMU or the resources within it, IP is more effectively protected.

Keywords: RPC, FMI, Co-simulation, Model Exchange

1 Introduction

No one simulation tool is suitable for all purposes, and complex heterogeneous models may require components from several different domains, perhaps developed in separate domain specific tools. How such components could be integrated in a standardized way is a problem the Function Mock-up Interface (FMI) (Blochwitz et al., 2012) aims to solve. More specifically, FMI is a tool independent standard to support both Model Exchange (ME) and Co-Simulation (CS) of dynamic models. Currently at version 2.0, the standard was one of the results of the MODELISAR project and is today managed by the Modelica Association.

A model implementing the FMI standard is known as an Functional Mock-up Unit (FMU), and is distributed as a zip-file with the extension *.fmu*. This archive contains:

- An XML-file that contains meta-data about the model, named *modelDescription.xml*.
- C-code implementing a set of functions defined by the FMI standard.
- Other optional resources required by the model implementation.

The FMI standard consists of two main parts:

- *FMI for Model Exchange (ME):* Models are exported without solvers and are described by differential, algebraic and discrete equations with time-, state- and step-events.
- *FMI for Co-Simulation (CS):* Models are exported with a solver, and data is exchanged between subsystems at discrete communication points. In the time between two communication points, the subsystems are solved independently from each other.

It's worth noting that a single FMU may support both ME and CS, and that the former may be wrapped by an importing tool into the latter.

FMI has seen high adaption rates since it's inception in 2011. The official tools page at `fmi-standard.org/tools` currently shows about 120 tools supporting FMI in one way or another. Clearly, the standard is solving a real problem. However, there are still some practical challenges related to it.

- FMI is cross platform in theory, but in practice this depends on the exporting tools ability to cross-compile native binaries. This is often not the case, making some FMUs unavailable for a certain platform.
- While FMI has been implemented in several languages, such as C (JModelica, 2017; QTronic, 2014), C++ (Widl et al., 2013; Hatledal, 2018), Python (Dassault Systems, 2017; Andersson et al.,

2016) and Java (Hatledal et al., 2018; Cortes Montenegro, 2014; Broman et al., 2013), out-of-the-box support for FMI is still missing in many languages.

- An FMU may require a license or pre-installed software on the target computer, making the FMU unavailable on many systems.
- Some FMI implementations only supports CS, making parts of the standard unavailable. Others may support ME also, but may not provide an easy way of solving them. Thus, some users may find the threshold for utilizing this feature too high.
- IP protection is not covered by the standard, however, model exporters are free to implement such mechanism as they see fit. Regardless, some model owners may worry about leaking IP and might be reluctant in sharing FMUs with others.

In order to resolve these issues, we present FMU-proxy, a framework for accessing FMUs compatible with FMI 2.0 for CS and ME in a language and platform independent way. The language and platform independent nature of the framework is achieved using well established RPC technologies, allowing clients and servers for FMU-proxy to be written in almost any language, on any platform. As noted by (Durling et al., 2017), server solutions such as presented in this paper are effective at protecting IP and unintended distribution. Furthermore, they allow FMUs with special requirements, such as pre-installed software and licence requirements, to be utilized on other systems.

Server implementations already exist for C++ and for the Java Virtual Machine (JVM), while client implementations exist for C++, Python, JavaScript and the JVM. Thanks to the stub generation capability of selected RPC frameworks, additional implementations in other languages are easy to realize as most of the code will be generated by the RPC compiler.

FMU-proxy is different from other similar frameworks offering distributed execution of FMUs in that it completely separates itself from the master algorithm. It is a completely standalone project which provides the infrastructure required to invoke FMUs over the wire. And just that.

Rather than having a number of tools creating their own, perhaps non-modular or internal, distribution mechanism, we hope FMU-proxy can be considered as an alternative or drop-in replacement for existing solutions. Possibly, creating a eco-system of remotely available FMUs in the process.

The source code of FMU-proxy is available online[1] under a permissive MIT license.

The rest of the paper is organized as follows. First some related work is given, followed by a presentation of the high-level architecture of the framework and subsequent implementation notes. Finally, a conclusion and future works are given.

2 Related work

Since the inception of the FMI standard, a multitude of libraries and software tools supporting the standard has been implemented. As of November 2018, the official FMI web page lists 120 such tools. Most of which supports invocation of FMI 2.0 compatible simulation models. A list of open-source tools with FMI import capabilities are given in Table. 1. Of these tools, four support distributed invocation of FMUs. These are:

DACCOSIM (Distributed Architecture for Controlled CO-SIMulation) (Galtier et al., 2015; Dad et al., 2016), a FMI compatible master algorithm, that lets the user design and execute a simulation requiring the collaboration of multiple FMUs on multi-core computation nodes or clusters. DACCOSIM is implemented in Java and is built on-top of the Eclipse Rich Client Platform, which provides the user with a GUI for setting up and running co-simulations. For complex scenarios with many FMUs and/or connections, a DSL can be used to replace the GUI. JavaFMI (Cortes Montenegro, 2014) is used for simulating and building FMUs. For communications, the ZeroMQ middleware is used. DACCOSIM is released under the AGPL license and is available for both Windows and Linux.

Coral (Sadjina et al., 2017) is a free and open-source software for distributed FMI based co-simulation, licensed under the MPL 2.0. Coral support FMI 1.0 and 2.0 for CS and was developed as part of the R&D project Virtual Prototyping of Maritime Systems and Operations (ViProMa) (Hassani et al., 2016). According to the authors, Coral is primarily a C++ library, but also acts as a tool as it requires setting up and running several programs in a distributed fashion. Additionally, it comes with a Command Line Interface (CLI) for running simulations. Coral works by installing a server program called a *slave provider* on each of the machines that should participate in a simulation. This program is responsible for publishing information on which FMUs are available on that machine, and exposes a subset of the FMI standard, compatible with both FMI 1.0 and 2.0, over the network. It also handles loading and running FMUs at the request of the master software, which acts as a client. Coral relies on the FMI Library (JModelica, 2017) to interact with FMUs, while networking is facilitated by the ZeroMQ middleware. Google Protocol Buffers are used for encoding/decoding messages sent over the network. A special feature of Coral is that slaves run in parallel, with variable values passed between them in a distributed fashion. Loggers and visualizers must therefore be implemented as FMUs themselves.

FMI Go! (Lacoursière and Härdin, 2017) is an open-source (MIT) distributed software infrastructure to perform distributed simulations with FMI compatible com-

[1]`https://github.com/NTNU-IHB/FMU-proxy`

Table 1. Open Source Software tools for simulating FMUs

	FMI support											
	CS		*ME*									
Name	*v1.0*	*v2.0*	*v1.0*	*v2.0*	**Standalone**	**Plugin**	**Distributed**	**API**	**CLI**	**GUI**	**Version**	**License**
Coral	x	x			x		x	x	x		0.9.0	MPLv2
DACCOSIM		x			x		x			x	2.1.0	AGPL
FMI Go!	x	x	x	x	x		x		x		-	MIT
FIDE		x				x				x	-	-
FUMOLA	x	x	x	x		x		x		x	alpha	-
Hopsan	x	x								x	2.10.0	GPLv3
INTO-CPS		x			x					x	-	MIT
MasterSim	x	x			x			x	x	x	0.5.0	LGPLv3
Ptolemy II	x	x	x	x	x			x		x	10.0.1	MIT
Xcos FMU wrapper	x	x	x			x				x	0.6	CeCILL
λ-Sim	x						x			x	-	-
OpenModelica	x		x	x	x					x	1.12.0	GPLv3

ponents, that runs on Windows, Linux and Mac OS X. Both CS and ME FMUs are supported, where ME FMUs are wrapped into CS FMUs. ME FMUs are preferred, as then the FMI Go! run-time environment can provide rollback and directional derivatives of the FMU. In CS FMUs, these features are considered optional and are often lacking, but may be required to achieve accurate and or stable simulations. FMI Go! used a client-server architecture, where a server hosts an individual FMU. Google Protocol Buffers are used for mapping the various FMI functions to messages that are transmitted using the ZeroMQ middleware. The Message Passing Interface (MPI) is also supported. The global stepper is then a client, consuming results produced by the FMUs. For applications that would want access to the simulation data, such as loggers, visualization etc., the global stepper serves also as a server. The System Specification and Parameterization (SSP) (Köhler et al., 2016) is used for defining the structure of a simulation. Additionally, a bare-bone CLI for this purpose also exists.

λ-Sim (Bonvini, 2016) is a tool implemented on top of Amazon Web Services (AWS) that converts FMI based simulation models into REST APIs. Provided with an FMU bundled with a JSON configuration file, λ-Sim builds a series of AWS services that will run simulations upon requests from a RESTful API. A web-based GUI is available, allowing users to load the generated API, simulate the model and visualize the results.

In (Hatledal et al., 2015) a software architecture for simulation and visualization based on FMI and web technologies was presented, using the Java only Remote Method Invocation (RMI) system for distributed access of FMUs.

Efforts has also been made to integrate the High Level Architecture (HLA) (Dahmann et al., 1997) and FMI in the works of (Awais et al., 2013) and (Garro and Falcone, 2015).

Additionally, the emerging standard Distributed Co-Simulation Protocol (DCP) (Krammer et al., 2018) should be mentioned. It is subject to proposal as a standard for real-time and non-real-time system integration and simulation, and standardization as a Modelica Association Project (MAP). The DCP is compatible with FMI and just like FMI, it defines only the slave. The design of a master is not in scope of the specification.

FMU-proxy is similar to the DSP in that it aims to enable distributed Co-Simulation. However, it does not define a standard, but mimics FMI for function definitions and leverages existing RPC frameworks and protocols for serialization and networking. It also makes no special considerations for real-time system integration like DSP does.

FMU-proxy differs from the other tools mentioned above as it does not actually simulate any FMUs. It merely provides access to the FMUs in a flexible way, supporting multiple RPCs and network protocols. Time stepping, variable routing, plotting etc. and other typical task performed by a master tool is left implemented by the integrating tool. This is a feature, allowing FMU-proxy to be lightweight, easy to use and re-usable in different software tools.

3 Software Architecture

This section introduces the high level concepts of FMU-proxy. The software architecture is shown in Fig. 1 and consists of three main parts:

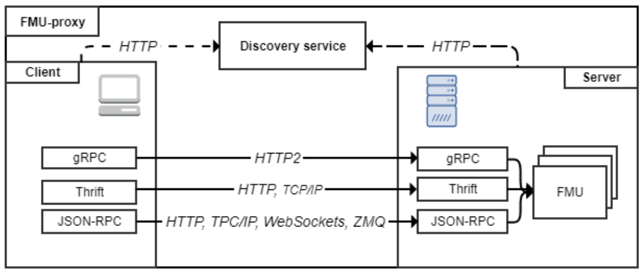


Figure 1. Software architecture.

1. **Discovery Services** A discovery service is a web application whose main responsibility is to communicate to users information about and the location of available FMUs. This information can be obtained visually through a web interface, or programmatically through an HTTP request.

   The discovery service has the following three HTTP services:

   - */availablefmus*: Called by user applications. Returns a JSON formatted string containing information about all available FMUs registered with the discovery service. The information include data from the *modelDescription.xml* as well as the IP address of the host machine and the RPC port(s).
   - */register*: Called by proxy-servers on start-up. Registers the server with the discovery server. Transmits network information, and information about the *modelDescription.xml* for each locally available FMU.
   - */ping*: Called by the proxy-servers at regular intervals, otherwise they will be considered to be offline by the discovery service.

   The discovery service is an optional feature and is not required when the remote end-point of an RPC service can be easily obtained. For instance when running the server on a physically accessible machine, allowing the IP address and RPC port(s) to be manually obtained. Another use case could be running both the client and server on *localhost* to enable invocations on FMUs from an otherwise unsupported language.

   Multiple discovery services may be online at any given time.

2. **Proxy-server**

   A proxy-server is responsible for making available one or more FMUs over a set of RPCs. At the very least, an implementation should support both Thrift and gRPC. Additional RPCs, such as JSON-RPC are optional.

   In addition to the RPC support, an implementation must be able to communicate with the discovery service over HTTP. Upon starting the server, the remote address of a discovery service should be specified. In order to ensure that the list of available FMUs are kept up to date, a heartbeat connection to the discovery service is established. At regular intervals, the server sends a ping - or heartbeat - over HTTP signalling that it is still online. When enough time has passed without such a notification, the server is considered offline and it's listing is subsequently removed from the discovery service.

   FMU-proxy supports both ME and CS FMUs running on the back-end, but the user is only provided with a CS API, as ME models are wrapped. Which solver and parameters to use are configurable by the user, however the availability of certain solvers are dependent on the server implementation.

3. **Proxy-clients**

   Proxy clients are used to connect with the FMUs hosted by the remote server(s). FMU-proxy aims to provide flexibility, such that clients can be implemented in a wide variety of languages and platform.

   Using Thrift or gRPC, the process of generating the required source-code for interacting with an remote FMU is quite straightforward. Listing. 1 shows the command required for generating the required sources when targeting Thrift in JavaScript. Similarly, Listing. 2 shows how C++ sources for gRPC are generated.

   Listing 1. Generating JavaScript sources for interfacing with remote FMUs using Thrift.

   ```
   thrift -js service.thift
   ```

   Listing 2. Generating C++ sources for interfacing with remote FMUs using gRPC.

   ```
   protoc -I=. --plugin=protoc-gen-grpc=
       grpc_cpp_plugin --cpp_out=. --
       grpc_out=. service.proto
   ```

The framework accomplishes several things, such as:

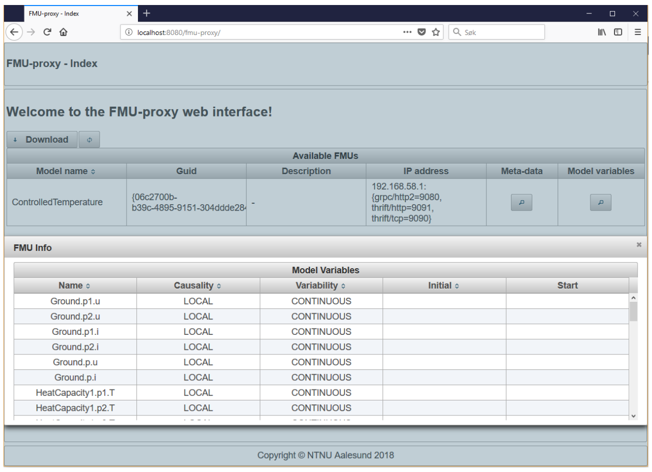
- **Additional language support**. FMUs can be accessed in previously unsupported languages with low effort, as no XML has to be parsed and no C-code has to be interfaced. Depending on the RPC used, stubs are auto-generated.
- **Cross platform access to any FMU**. FMUs can be invoked from unsupported platforms, i.e an FMU compiled only for Windows can be invoked from a Linux system. Naturally, a server running on a platform supported by the FMU is required.
- **FMI compliance without FMU packaging**. It allows models to be compliant with the FMI standard without actually being packaged as an FMU. From a client's perspective, there is no difference between a "physically backed" FMU and one implemented in-memory. All the client sees is the RPC interface mimicking FMI.
- **Relaxed run-time constraints**. FMUs that require special software and/or licenses can be invoked from otherwise incompatible systems.
- **Re-usability**. As the framework is decoupled from the master algorithm, it can be used by any software tool with a centralized master architecture that wants to support distributed execution of FMUs.

4 Implementation

This section describes some of the implementation details related to FMU-proxy. Currently, it comes with server implementations for C++ and the JVM. Client implementations exist also for C++ and the JVM. Additionally, proof of concept implementations for Python and JavaScript are bundled. In addition to the servers and clients, FMU-proxy comes bundled with an implementation of a discovery service.

4.1 The Discovery Service

The discovery service has been implemented in Kotlin, a statically typed language 100% interoperable with Java. The front-end seen in Fig. 2 has been implemented using PrimeFaces, a UI component framework for Java Server Faces (JSF). It offers basic functionality such as the ability for users to download available RPC schemas and to view information about available FMUs in a structured way.


FMU-proxy - Index

localhost:8080/fmu-proxy/

FMU-proxy - Index

Welcome to the FMU-proxy web interface!

Download

Available FMUs

Model name	Guid	Description	IP address	Meta-data	Model variables
ControlledTemperature	{06c2700b-b39c-4895-9151-304ddde28	-	192.168.58.1: {grpc/http2=9080, thrift/http=9091, thrift/tcp=9090}		

FMU Info

Model Variables

Name	Causality	Variability	Initial	Start
Ground.p1.u	LOCAL	CONTINUOUS		
Ground.p2.u	LOCAL	CONTINUOUS		
Ground.p1.i	LOCAL	CONTINUOUS		
Ground.p2.i	LOCAL	CONTINUOUS		
Ground.p.u	LOCAL	CONTINUOUS		
Ground.p.i	LOCAL	CONTINUOUS		
HeatCapacity1.p1.T	LOCAL	CONTINUOUS		
HeatCapacity1.p2.T	LOCAL	CONTINUOUS		

Copyright © NTNU Aalesund 2018

Figure 2. The discovery service's web interface. Here available FMUs are listed, showing network information and data from the *modelDescription.xml*.

4.2 Proxy-server

Two server implementations have been realized, each described more in detail below. Which one to deploy in production depends on the users need for RPCs supported, stability, stability, quality of the available ME solvers, memory foot-print and performance. No one implementation will excel at everything.

4.2.1 JVM

The JVM implementations is written in Kotlin and rely on FMI4j (Hatledal et al., 2018) for interacting with FMUs. FMI4j supports FMI 2.0 for CS and ME. ME models can be wrapped as CS ones using solvers from Apache Commons Math.

The implementation supports Thrift (TPC/IP - binary, HTTP - JSON), gRPC (HTTP2 - protocol buffers) as well as JSON-RPC (HTTP, TCP/IP, WebSockets, ZeroMQ). Of the two current implementations, this one is considered the most stable and feature rich.

4.2.2 C++

The C++ implementation is cross-platform and is written in C++17. All dependencies are available using the library manager *vcpkg*, making it easy to build on any platform. Currently, Thrift (TPC/IP - binary, HTTP - JSON) and gRPC (HTTP2 - protocol buffers) are supported RPCs.

FMI4cpp (Hatledal, 2018) is used for interacting with FMUs. It supports FMI 2.0 for CS and ME. ME models can be wrapped as CS ones using solvers from Boost odeint.

4.3 Proxy-client

FMU-proxy comes bundled with client implementations for C++, the JVM, Python and JavaScript. The two latter are crude and ought to be considered as proof of concept. They are, however, bundled with the source code to showcase how easy it is to interface with FMU-proxy from new languages. A MATLAB demo using JSON-RPC over HTTP is also available.

The C++ and JVM implementations are more elaborate, providing a unified, higher level API for the users. No matter which RPC is used, there is no difference between a remote and local FMU slave for the user. As illustrated by Figure. 3, they all share the same interface, defined by FMI4cpp and FMI4j for C++ and JVM implementations respectively. Assuming a tool is using one of these FMI implementations, support for distributed execution can be seamlessly added with minimal changes to the existing code base.

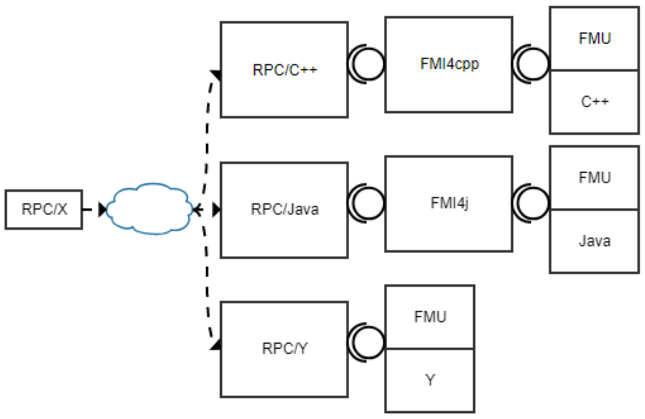


Figure 3. FMI4cpp and FMI4j's slave interface could hide slaves stemming from either an in-memory implementation or an actual FMU. A slave in any language supported by the chosen RPC could also be implemented directly behind the RPC layer.

5 Conclusion and Future Work

In this paper an open-source framework for working with FMUs across languages and platforms, named FMU-proxy, has been presented. It has been designed to allow distributed execution of FMUs, which also enables access to FMUs in previously unsupported languages and on incompatible platforms. Since FMU-proxy is independent of the master algorithm, it can be re-used across software projects.

Some features of FMU-proxy include:

- Brings FMI capabilities to previously unsupported languages and otherwise incompatible platforms.
- By implementing the RPC functions directly, FMI compliant models can be implemented without having to package them into FMUs.
- Allows code re-use between projects that requires distributed execution of FMUs, independent of implementation language.
- Enables companies to securely share FMUs. By hosting their own proxy server and directory service, neither the FMUs nor the knowledge about them leaves the company controlled servers.
- A unified slave interface for C++ and JVM users. On these platforms, local and remote slaves implement the same interface.

Server implementations exists for C++ and the JVM, while client implementations exists for JavaScript, Python, C++ and the JVM. Due to the language independent nature of the RPC frameworks and protocols used, and especially the code-generation feature of selected RPC frameworks, further client implementations in additional languages require little effort.

Several enhancements to FMU-proxy is planned for the future, including:

1. Automatic distribution of FMUs over the network. It should be possible to upload an FMU to the Discovery Service, which in turn should find a suitable server for it to run on.
2. Manual distribution of FMUs over the network. It should be possible for the user to directly upload an FMU to an available proxy-server.
3. Publication of the C++ implementation to the cross-platform C++ library manager *vcpkg*.
4. Benchmark results, comparing the different implementations, RPCs and local vs. distributed execution of FMUs.
5. Once released, FMI 3.0 support will be added.

FMU-proxy is available from GitHub at `https://github.com/NTNU-IHB/FMU-proxy`. Here, pre-built server executables can be obtained. Client libraries for Java are available through *maven* at `https://jitpack.io/#NTNU-IHB/FMU-proxy`, while client libraries for C++ will be available through *vcpkg*.

6 Acknowledgement

The research presented in this paper is supported by the Norwegian Research Council, SFI Offshore Mechatronics, project number 237896.

References

Christian Andersson, Johan Åkesson, and Claus Führer. Pyfmi: A python package for simulation of coupled dynamic models with the functional mock-up interface. *Technical Report in Mathematical Sciences*, 2016(2), 2016.

Muhammad Usman Awais, Peter Palensky, Atiyah Elsheikh, Edmund Widl, and Stifter Matthias. The high level architecture rti as a master to the functional mock-up interface components. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 315–320. IEEE, 2013.

Torsten Blochwitz, Martin Otter, Johan Akesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 076, pages 173–184. Linköping University Electronic Press, 2012.

Marco Bonvini. Lambdasim, 2016. URL `https://github.com/mbonvini/LambdaSim`. (Date accessed 11-November-2018).

David Broman, Christopher Brooks, Edward A. Lee, Thierry S. Nouidui, Stavros Tripakis, and Michael Wetter. Jfmi - a java wrapper for the functional mock-up interface, 2013. URL `https://ptolemy.eecs.berkeley.edu/java/jfmi/`. (Date accessed 23-June-2018).

Johan Sebastian Cortes Montenegro. Javafmi una librería java para el estándar functional mockup interface. 2014.

Cherifa Dad, Stephane Vialle, Mathieu Caujolle, Jean-Philippe Tavella, and Michel Ianotto. Scaling of distributed multi-simulations on multi-core clusters. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2016 IEEE 25th International Conference on*, pages 142–147. IEEE, 2016.

Judith S Dahmann, Richard M Fujimoto, and Richard M Weatherly. The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*, pages 142–149. IEEE Computer Society, 1997.

Dassault Systems. Fmpy, 2017. URL `https://github.com/CATIA-Systems/FMPy`. (Date accessed 23-June-2018).

Erik Durling, Elias Palmkvist, and Maria Henningsson. Fmi and ip protection of models: A survey of use cases and support in the standard. pages 329–335, 07 2017.

Virginie Galtier, Stephane Vialle, Cherifa Dad, Jean-Philippe Tavella, Jean-Philippe Lam-Yee-Mui, and Gilles Plessis. Fmi-based distributed multi-simulation with daccosim. In

Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, pages 39–46. Society for Computer Simulation International, 2015.

Alfredo Garro and Alberto Falcone. On the integration of hla and fmi for supporting interoperability and reusability in distributed simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pages 9–16. Society for Computer Simulation International, 2015.

Vahid Hassani, Martin Rindarøy, Lars T Kyllingstad, Jørgen B Nielsen, Severin Simon Sadjina, Stian Skjong, Dariusz Fathi, Trond Johnsen, Vilmar Æsøy, and Eilif Pedersen. Virtual prototyping of maritime systems and operations. In *ASME 2016 35th International Conference on Ocean, Offshore and Arctic Engineering*, pages V007T06A018–V007T06A018. American Society of Mechanical Engineers, 2016.

Lars Ivar Hatledal. Fmi4cpp, 2018. URL `https://github.com/SFI-Mechatronics/FMI4cpp`. (Date accessed 16-November-2018).

Lars Ivar Hatledal, Hans Georg Schaathun, and Houxiang Zhang. A software architecture for simulation and visualisation based on the functional mock-up interface and web technologies. In *Proceedings of The 57th Conference on Simulation and Modelling (SIMS 56): October, 7-9, 2015, Linköping University, Sweden*. Linköping University Electronic Press, Linköpings universitet, 2015.

Lars Ivar Hatledal, Houxiang Zhang, Arne Styve, and Geir Hovland. Fmi4j: A software package for working with functional mock-up units on the java virtual machine. In *Proceedings of The 59th Conference on Simulation and Modelling (SIMS 59), 26-28 September 2018, Oslo Metropolitan University, Norway*, number 153, pages 37–42. Linköping University Electronic Press, 2018.

JModelica. Fmi library, 2017. URL `http://www.jmodelica.org/FMILibrary`. (Date accessed 09-December-2017).

Jochen Köhler, Hans-Martin Heinkel, Pierre Mai, Jürgen Krasser, Markus Deppe, and Mikio Nagasawa. Modelica-association-project "system structure and parameterization"– early insights. In *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*, number 124, pages 35–42. Linköping University Electronic Press, 2016.

Martin Krammer, Martin Benedikt, Torsten Blochwitz, Khaled Alekeish, Nicolas Amringer, Christian Kater, Stefan Materne, Roberto Ruvalcaba, Klaus Schuch, Josef Zehetner, et al. The distributed co-simulation protocol for the integration of real-time systems and simulation environments. In *Proceedings of the 50th Computer Simulation Conference*, page 1. Society for Computer Simulation International, 2018.

Claude Lacoursière and Tomas Härdin. 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*, number 132, pages 653–662. Linköping University Electronic Press, 2017.

QTronic. Fmu sdk, 2014. URL `http://www.qtronic.de/de/fmusdk.html`. (Date accessed 23-June-2018).

Severin Sadjina, Lars T Kyllingstad, Martin Rindarøy, Stian Skjong, Vilmar Æsøy, Dariusz Eirik Fathi, Vahid Hassani, Trond Johnsen, Jørgen Bremnes Nielsen, and Eilif Pedersen. Distributed co-simulation of maritime systems and operations. *arXiv preprint arXiv:1701.00997*, 2017.

Edmund Widl, Wolfgang Müller, Atiyah Elsheikh, Matthias Hörtenhuber, and Peter Palensky. The fmi++ library: A high-level utility package for fmi for model exchange. In *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on*, pages 1–6. IEEE, 2013.

Standardized Integration of Real-Time and Non-Real-Time Systems: The Distributed Co-Simulation Protocol

Martin Krammer[1] Klaus Schuch[2] Christian Kater[3] Khaled Alekeish[4] Torsten Blochwitz[4]
Stefan Materne[5] Andreas Soppa[6] Martin Benedikt[1]

[1]VIRTUAL VEHICLE Research Center, Austria, {martin.krammer,martin.benedikt}@v2c2.at
[2]AVL List GmbH, Austria, klaus.schuch@avl.com
[3]Leibniz Universität Hannover, Germany, kater@sim.uni-hannover.de
[4]ESI-ITI GmbH, Germany, {torsten.blochwitz,khaled.alekeish}@esi-group.com
[5]TWT GmbH, Germany, stefan.materne@twt-gmbh.de
[6]Volkswagen AG, Germany, andreas.soppa@volkswagen.de

Abstract

Co-simulation techniques have evolved significantly over the last 10 years. System simulation and hardware-in-the-loop testing are used to develop complex products in many industrial sectors. The Functional Mock-Up Interface (FMI) represents a standardized solution for integration of simulation models, tools and solvers. In practice the integration and coupling of heterogeneous systems still require enormous efforts. Until now no standardized interface or protocol specification is available, which allows the interaction of real-time and non-real-time systems of different vendors. This paper presents selected technical aspects of the novel *Distributed Co-simulation Protocol* (DCP) and highlights primary application possibilities. The DCP consists of a data model, a finite state machine, and a communication protocol including a set of protocol data units. It supports a master-slave architecture for simulation setup and control. The DCP was developed in context of the ACOSAR project and was subsequently adopted by Modelica Association as a Modelica Association Project (MAP). It may be used in numerous industrial and scientific applications. The standardization of the DCP allows for a modular and interoperable development between system providers and integrators. In the end, this will lead to more efficient product development and testing.

Keywords: DCP, co-simulation, real-time, integration, standard

1 Introduction

Modeling and simulation represent key methods for successful development of cyber-physical systems. With the introduction of co-simulation methodologies, holistic cross-domain or system simulations became possible. This enabled exchange and integration of simulation models, tools, and solvers from different sources. The automotive industry is characterized by a multi-tiered organization. A deep hierarchy of suppliers performs distributed development and integration of automotive components, parts, and systems, that in the end are manufactured to complete vehicles. Depending on the stage of development, simulation models or real prototypes are available. The advantage of simulation models is that they can be tested in terms of software. Software tests are comparably cheap. However, they typically do not consider timing aspects or uncertainties of measured quantities. On the other hand, prototypes are advantageous when it comes to product validation. A prototype shows real-world behaviour and interacts with the environment. The disadvantages are that prototypes are usually very expensive, and safety critical or rare situations are difficult to test. For these reasons it seems advantageous to combine simulation and real-world prototype based testing approaches. For certain use cases this is considered as a possible solution to cope with the arising complexity, due to the high number of different scenarios and situations. This especially includes the field of automated driving (Doms et al., 2018). The European Union's automotive investment in research and development has increased to 53.8 billion Euro annually (European Automobile Manufacturers Association, 2018). Testing efficiency is key to successful product development. Interoperability of simulation tools and test infrastructure contributes to testing efficiency. Therefore the use of standards is essential.

The DCP (Distributed Co-Simulation Protocol) was developed in the ACOSAR project (Krammer et al., 2016). ACOSAR stands for "Advanced Co-Simulation Open System Architecture". ACOSAR was an ITEA 3[1] (Information Technology for European Advancement) project. Three original equip-

[1]http://www.itea3.org

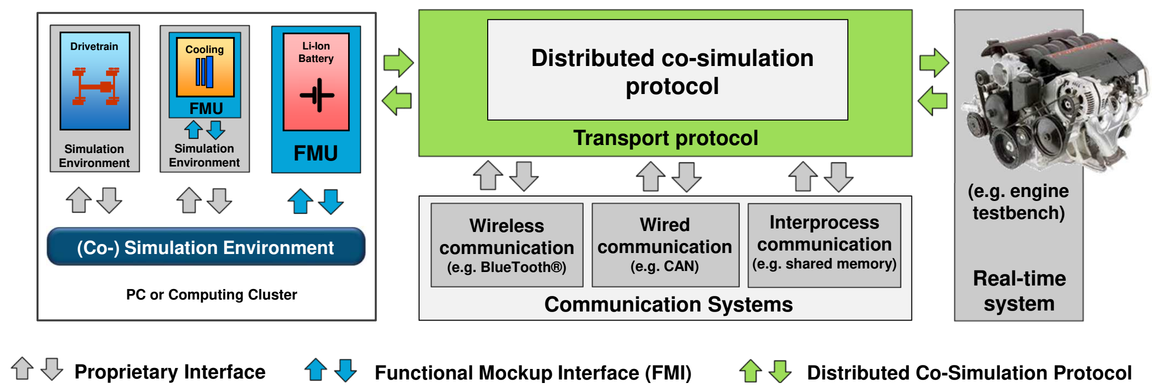

Drivetrain
Simulation Environment
Cooling
FMU
Simulation Environment
Li-Ion Battery
FMU
(Co-) Simulation Environment
PC or Computing Cluster
Distributed co-simulation protocol
Transport protocol
Wireless communication (e.g. BlueTooth®)
Wired communication (e.g. CAN)
Interprocess communication (e.g. shared memory)
Communication Systems
(e.g. engine testbench)
Real-time system
Proprietary Interface
Functional Mockup Interface (FMI)
Distributed Co-Simulation Protocol

Figure 1. DCP concept.

ment manufacturers (OEM), 9 companies from the automotive supply chain, including simulation tool vendors, system and component providers, as well as 4 partners from research and academia cooperated. Their main goals were (1) the specification and demonstration of the DCP, and (2) preparation of standardization of the DCP with a recognized standardization body in order to promote it as the next co-simulation standard. Figure 1 shows an overview of the DCP's concept.

2 Related Work

The Functional Mock-up Interface (FMI) is introduced in (Blochwitz et al., 2011). The FMI was proposed to solve the need for interoperability between models and solvers. It was developed in the MODELISAR project, starting in 2008. The FMI specification is standardized as a Modelica Association Project (MAP). Its current version is 2.0 and was released in 2014. The FMI specification defines an interface for model exchange and co-simulation. Today more than 100 software tools support the FMI[2]. For distributed simulation environments, network communication technologies are frequently used in practice. However, such a "communication layer is not part of the FMI standard" (Modelisar Consortium and Modelica Association Project "FMI", 2014, p.93).

The Distributed Co-Simulation Protocol (DCP) is introduced in (Krammer et al., 2018). Its five main design ideas are highlighted; the improvement of interoperability between systems from different vendors, the integration of distributed real-time systems, the compatibility to a broad range of computing platforms, the support of multiple transport protocols, and development efficiency. The paper also introduces a typical architecture description of a DCP slave. It also describes the DCP's three different operating modes, namely hard real-time (HRT), soft real-time (SRT), and non real-time (NRT). They describe a DCP slave's relationship to absolute time. In general, deadlines must be kept for HRT and SRT operating modes. Simulation time must or should be synchronous to absolute time. The NRT operating mode can be used for distributed, computational co-simulation. In NRT operating mode, simulation time is independent from absolute time. The DCP specifies a state machine that governs the behaviour of a DCP slave. It defines five phases of a simulation cycle. Furthermore, the paper describes the main concepts of the communication protocol, including the design of protocol data units (PDU), the request and response mechanism, as well as the mechanism for configuration and exchange of input and output data. An example for UDP as a transport protocol is given, explaining the mechanism in detail.

In (Krammer and Benedikt, 2018) an algorithm for efficient generation of configurations for exchange of input and output data is given. The problem of finding such a configuration is an instance of the bin packing problem. In order to run such an algorithm, a co-simulation scenario description is required. The paper suggests a solution based on an XML schema description.

3 The Distributed Co-Simulation Protocol

The DCP is designed as a novel communication protocol on application level. It is intended for configuration and data exchange in co-simulation applications. The following sections provide details on features and technical novelties. Furthermore, the relationship to the FMI standard is highlighted.

3.1 DCP Feature Overview

3.1.1 Communication Architecture

The DCP implements the master-slave principle. It enables a DCP master to organize and configure its DCP slaves, so that a specific co-simulation scenario can be realized. A DCP slave represents a single subsystem of the co-simulation scenario. It can be a hardware-in-the-loop (HiL) system, a test bench, a

[2] `http://fmi-standard.org/tools/`

simulation tool, or similar system.

The DCP is a communication protocol intended for co-simulation configuration and data exchange. It is defined as a communication protocol that is independent of the underlying transport protocol. Classification of the DCP according to the Open Systems Interconnection (OSI) model (Zimmermann, 1980; International Telecommunication Union, 1994) is ambiguous. Its main properties fulfill major criteria for the application layer, e.g. access for application processes to the OSI environment. This is the highest layer defined in the OSI model. The DCP also features properties of the presentation layer, e.g. the design of DCP protocol data units (PDU), their associated fields and corresponding data types. The DCP implements a registration scheme, that allows the setup and simulation of co-simulation scenarios. This can be interpreted as a session. For the transport layer, the DCP defines mechanisms like the PDU sequence ID. Despite the fact that some transport protocols target properties like reliability (e.g. transmission control protocol, TCP), the DCP provides basic mechanisms to achieve similar behavior when a transport protocol is used that does not support this property (e.g. user datagram protocol, UDP).

3.1.2 State Machine

The DCP protocol is operated by a discrete state machine. The main design goal of this state machine is to ensure safe and reliable operation of real-time and non-real-time systems. In total, the DCP state machine consists of a set of 19 states grouped in 6 superstates. The entry point to the state machine is reached when the DCP software implementation is loaded to the DCP slave, the latter also indicates that the slave becomes available for registration by the master. A simulation cycle represents one complete pass through the DCP state machine.

The state machine enables simulation cycles having 6 different phases. In phase 1, a DCP slave is registered with a master which takes ownership of its registered slave. The later DCP slave is then exclusively controlled by its master. In phase 2, the DCP master configures its DCP slaves by generating a valid configuration scenario based on the DCP slave description of its slaves. Also for connection oriented transport protocols, a connection is established during the current phase. In phase 3, an iterative initialization process is carried out, the outcome of this process is establishing a consistent initial state over interconnected slaves. (see 3.2.2 for more details). In phase 4, The DCP slave in real-time operating modes is running and inputs/outputs are exchanged according to the configurations. Moreover, simulation time is mapped to absolute time. For non-real-time operating mode, simulation time does not progress at this phase. See section 3.2.3 for more details. Phase 5 applies only to non-real-time systems and each slave at this phase computes exactly one communication step and output is communicated to other slaves. Also the virtual simulation time is incremented by the number of specified steps. Phase 6 is intended to stop the simulation in a safe way, a stop of simulation can be triggered either by the master or by the slave itself.

3.1.3 Communication Protocol

To facilitate the communication between the master and slaves, DCP introduces the concept of Protocol Data Units (PDUs) that can be exchanged between the master and slaves. DCP addresses different types of PDUs which are used for different purposes and they serve distinct functionalities. So according to the functionalities of the PDUs, they are categorized in different families. DCP defines three top PDUs families named as Control, Notification (NTF) and Data (DAT) PDUs. The Control PDUs are further divided into Request and Response (RSP) families. Note that the Request PDUs are only sent by the master to its slaves and they consist of Configuration (CFG), State Change (STC) and Information (INF) requests. A slave upon receiving a request from its master has to acknowledge by sending a RSP PDU. DCP slaves can use NTF PDUs to inform the DCP master about certain events, for example, when the slave changes its state. Data PDUs can be used to transmit inputs and outputs between DCP slaves (slave-to-slave communication) and between the DCP master and its DCP slaves. Parameters (fixed or tunable), which are also packed in Data PDUs, can only be transmitted by the DCP master.

The Control PDUs are exchanged according to the request-response pattern. The latter pattern allows the DCP master to send specific requests to its slaves, it also enables each slave to inform its DCP master about the result of a requested action. Considering that DCP might be used on top of an unreliable transport protocol, packets loss might occur during the exchange of Control PDUs. Handling the latter situation can be determined by the DCP master and DCP slaves. For example, the DCP master might decide to initiate the retransmission of a Request PDU after a certain period of time.

3.1.4 DCP Data Exchange

DCP facilitates the exchange of input/output data between slaves. It enables a slave either to send data to other slaves directly or to send data to the master which passes this data on to all destination slaves. While the former communication way saves time and resources, the latter is intended for more sophisticated co-simulation configurations including extrapolation techniques or step-size control. A co-simulation DCP slaves configuration consists of a set of their DCP slave descriptions, the connections between their inputs and outputs as well as some other settings chosen by the master. This configuration is rolled out to the slaves during the configuration phase. A slave that needs to send output data, receives a `CFG_output` PDU from the master, for each output data. The same applies to input data, the slave receives a `CFG_input` PDU for each input data it is going to receive. In addition to the two mentioned types of Control PDUs, the master also sends `CFG_target_network_information` and `CFG_source_network_information` PDUs. The latter two types of Control PDUs enable slaves to know where to send or from where to receive data, respectively, and their contents depend on the communication medium.

In addition to the input and output data, DCP also enables the master to send data for the parameters of its slaves and only the master can send this kind of data. Parameters can be either fixed or tunable, both types can be set during the configuration phase using the `CFG_parameter` PDU. While fixed parameters can be set only using the latter PDU, tunable ones can be set using the `DAT_parameter` PDU during any of the states that allow `DAT_input_output` PDUs to be sent. In the same way like the other Data PDUs, `DAT_parameter` PDUs are sent according to the stored configuration information which is received using the `CFG_tunable_parameter` PDUs during the configuration phase.

3.2 Technical Novelties

3.2.1 Integration Process

The DCP specification document describes the design of a DCP slave only. A DCP master is required to control a co-simulation scenario, which includes at least one DCP slave. In order to design and set up such a scenario, the DCP defines a non-normative default integration methodology. It defines the roles of a *DCP slave provider*, and a *DCP integrator*. The DCP integrator uses the DCP slave descriptions and a DCP master for configuration and control of the scenario.

The DCP slave description (DCPX) is a XML (Extensible Markup Language) file which describes one single DCP slave. It contains all static information related to one specific DCP slave. Its structure is defined by a normative XML XSD (XML Schema Definition) file. The top level structure of this schema definition file is shown in Figure 2. The DCP slave provider must provide an accompanying DCP slave description together with a DCP slave. The DCP master can attain all required information about available slaves by accessing their description files.

According to the specification, the DCP slave description must be stored in a single file named `dcpSlaveDescription.dcpx`, which in turn must be placed in a DCP file. The DCP file is a zip encoded

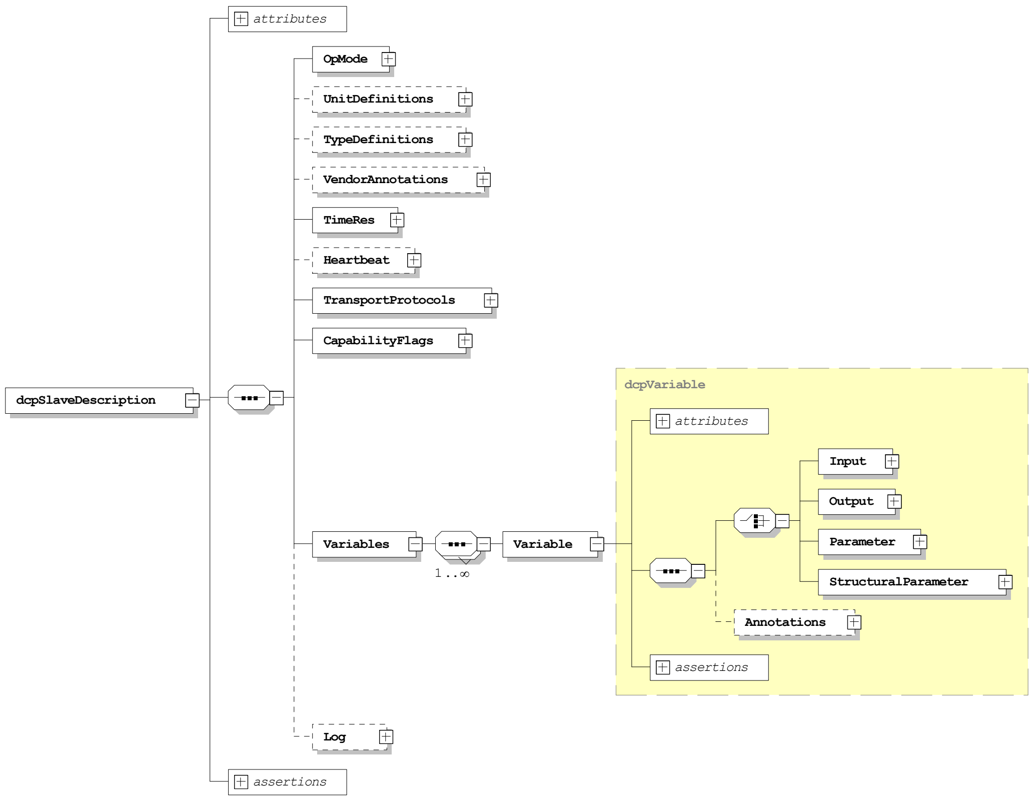


Figure 2. DCP slave description schema definition.

file (ISO/IEC JTC 1/SC 34, 2015) having the extension .dcp. Its internal structure is normative and designed to hold multiple DCPX files which are compliant to different DCP version numbers. This is one example of several design provisions taken into account to provide a future-proof DCP specification.

The set of DCP slave description schema files is normative. It does not only define the required structures of elements and attributes, but also supplementary assertions and constraints. Assertions and constraints are highly efficient for expressing logical relationships between elements and attributes.

Assertions are expressed in the `xs:assert` tag using the XML Path Language (XPath). An XPath expression addresses parts of an XML document in terms of a tree structure (Document Object Model, DOM). One location step in this tree consists of axis, node-test, and an optional predicate. An example for such an assertion is shown in Listing 1. It links the capability flag `canMonitorHeartbeat` to the defined XML child element `Heartbeat`. This prevents e.g. a set capability flag while the associated configuration information contained in the child element is missing. Assertions are a feature of XSD version 1.1. However, an XSL transformation (XSLT) file is specified, transforming the provided XSD version 1.1 schema definition file into a XSD version 1.0 schema definition file.

Furthermore, `xs:unique`, `xs:key` and `xs:keyref` tags are used to express constraints. Typical examples of application include the verification of uniqueness of names and the verification of cross-referenced key values.

In context of the DCP specification assertions and constraints provide strong formalisms which can be used for automated DCPX validation. This has shown to be advantageous in comparison to informal textual rules given in the specification document.

```
<xs:assert test="
((./CapabilityFlags/@canMonitorHeartbeat
    eq true()) and boolean(./Heartbeat))
         or
((./CapabilityFlags/@canMonitorHeartbeat
    eq false()) and boolean(./Heartbeat)
    eq false())
"/>
```

Listing 1. Assertion for capability flag and XML child element, as defined in the DCP slave description schema file.

3.2.2 Simulation Initialization

The DCP supports initialization calculations to achieve a consistent initial condition of connected DCP slaves. The DCP description file contains information about the DCP slave's dependencies. A dependency describes if an output is controllable by an input or parameter. Dependency information can be specified for the Initialization and Run superstates separately. The first is applicable prior to simulation, whereas the latter is applicable during simulation. Additionally, a DCP slave can mark outputs to be valid only in Initialization superstate. Such outputs are called *initial outputs*.

In the initialization phase simulation time does not progress. Hence, the master may roll out a configuration where the master receives all outputs and sends all inputs to the DCP slaves. The inputs sent by the master to the DCP slaves are not necessarily the outputs of other DCP slaves, a sophisticated master could send values chosen by a numerical solver instead (to solve algebraic loops). Algebraic loops in the context of FMI are explained in (Broman et al., 2013).

Connected DCP slaves may form pseudo algebraic loops. Such pseudo algebraic loops can be detected by exploiting the dependency information provided by the individual DCP slaves.

3.2.3 Simulation Synchronization

The master can observe the whole system to check if a global stable state was reached. The master informs the slaves afterwards to start the actual simulation test run. The achieved initial consistent configuration might still not correlate with reality. An output of a DCP slave could represent a physical quantity which typically fluctuates within certain boundaries. To minimize this difference and to circumvent this issue separate states were introduced. Each slave has the possibility to indicate that a local stable state has been reached, after fade out of transient oscillations. The master may observe the whole scenario to check if a global stable state was reached. If this is the case, the master may start the actual simulation run.

3.2.4 Connection-oriented Transport Protocols

The DCP supports connection-oriented and packet-oriented transport protocols.

To support connection-oriented protocols, two major mechanism were introduced to the DCP.

First of all, new states were introduced to distinguish between opening an endpoint and opening a connection. This is necessary to enable coordinated slave-to-slave communication. Without this distinction it would not be possible to detect if a slave has successfully opened its endpoints, ready to accept connections. Using this mechanism the master is able to instruct all slaves to open all endpoints first. After that, the slaves may establish their connections.

Second, the length of each PDU is sent on the stream, ahead of the actual PDU of the connection-oriented transport protocol. This eases implementation of slaves, because a slave is free to decide how many bytes he has to receive, independent from a

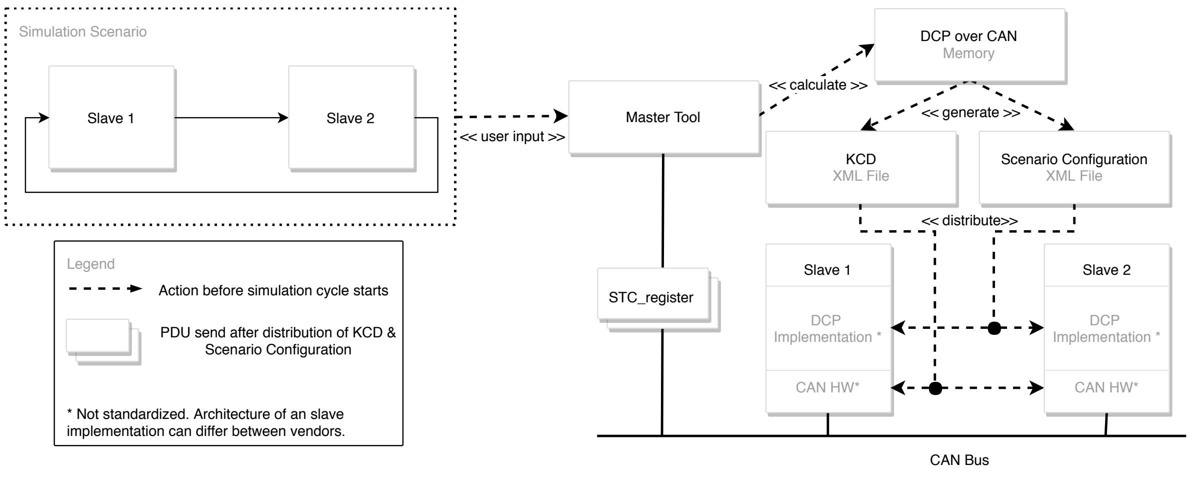


Figure 3. Possible scenario of DCP over CAN

slaves' configuration. In addition to that, PDU length verification also became possible.

Without the length ahead of the PDU a slave can only guess the length, based on its own assumptions. Misbehavior by other participants in terms of PDU length would not be detectable.

3.2.5 Non-native Transport Protocols

The DCP distinguishes between native and non-native transport protocols. Native DCP means that the mapping of PDUs to the transport protocol preserves the bit sequence.

If a transport protocol cannot fulfill this condition it is called a non-native transport protocol. One example of such a non-native transport protocol is the CAN bus communication system. Due to limitations of CAN, e. g. the CAN payload is limited to 8 bytes, not all Control PDUs can be send via CAN. For this reason the configuration of a slave will not be communicated by CAN.

To support the exchange of configuration PDUs for CAN an XML model is specified. The information contained in this model has to be generated by a master tool. It must be transmitted to the slave as a static configuration before simulation start. This model contains a K-matrix and the scenario configuration. The K-matrix contains all elements to describe the messages and signals of the CAN bus and the participation of the bus members to the messages. The scenario configuration contains all elements to describe the co-simulation scenario. When using a native DCP transport protocol instead, this information would be distributed to each DCP slave using configuration PDUs. In addition, the co-simulation scenario contains various other information, like DCP slave names, DCP slave identifiers, and their UUIDs (universally unique identifiers). The UUID is used to match information from these elements to DCP slaves. However, the way how information from the XML model is transferred to the DCP slaves is out of scope of DCP.

Figure 3 shows a possible scenario how DCP over CAN may be used in practice. A user defines the desired co-simulation scenario in master tool, supporting DCP over CAN. Based on this scenario the master calculates the K-Matrix and the scenario configuration in the DCP over CAN model and stores these information in different files. For the K-matrix e.g. the open source file format KCD was chosen. Any other file format describing CAN communications, e.g. DBC from Vector, would also be possible. After slaves are started, the CAN hardware is configured using the KCD file. The DCP implementation is configured using the scenario information. As a result, all slaves are waiting in state alive. The master tool sends out the register PDUs using the CAN bus and starts the simulation cycle.

3.2.6 Complex data types

New sensor technologies are currently evolving, for example camera, lidar or radar systems for the automotive market. In the automotive domain, these sensor types are used to enable advanced driver assistance systems (ADAS), to pursue the goal of automated driving. Test and operation of these systems rely on transmission of multidimensional or binary data types.

The DCP defines a binary data type to transmit arbitrary information. The binary representation consists of a 32 bit unsigned integer value that specifies the length in bytes of the actual data, followed by the binary data itself. The data is transmitted as given without any change in bit or byte order. Thus, the maximum length of data is limited to $2^{32}-1$ bytes. A DCP slave can limit this maximum length per variable, by specification of a maximum length in the DCP slave description. It is also possible to specify a MIME type compliant to RFC 2045 (Freed and Borenstein, 1996). The DCP integrator has to ensure compatibility between outputs and connected inputs of binary data type, in the sense of maximum length and MIME type.

The DCP offers the possibility to define variables as arrays. An array variable is a data structure consisting of a collection of variables of the same type, each identified by an array index. A variable may have a constant number of dimensions. Each dimension has a size, defined by a constant or a structural parameter. By using a structural parameter it is possible to change the size of a dimension at any time.

3.2.7 Logging

The DCP supports the transmission of arbitrary log data from a DCP slave to its master. For that, it defines two different approaches, namely log-on-request and log-on-notification.

For log-on-request, log messages are stored by the DCP slave. They are picked up by the master on request and at any time. Thereby the master can avoid a high workload caused by log messages in the real-time-critical superstate Run. For log-on-notification, log messages are not stored within the DCP slave. Instead, they are transmitted to the master immediately. This mechanism supports devices with limited memory capacities, like micro-controllers.

The exact format of a log message is defined in the DCP slave description by using log templates. A log template consists of a category, level and a message. The category is defined in the DCP slave description. The possible values for the level are defined by the DCP. The category and the level can be used by the master to configure the logging of the DCP slave in a group wise manner. It is not necessary to configure every single log template individually.

The message of a log template defines the actual log string which is displayed to the user. In this string placeholders can be set, which define the values sent by a DCP slave to the master with the log message as seen in Figure 4. The full log message is then generated by the master, by replacing the placeholders with the received values from the slave.

3.3 Interaction with FMI

Right from the beginning of the ACOSAR project existing solutions for distributed co-simulation and system integration were carefully surveyed (Lichtenstein et al., 2016). Today, the FMI represents one of the most frequently used standards in the field of simulation. It is applied in many domains, including automotive, aerospace, maritime, or power grid domains. It is implemented in more than 100 commercial and open source tools. The ACOSAR consortium members recognized the feature set of FMI which represents the current state-of-the-art for co-simulation. As a consequence, the consortium proposed the adoption and extension of available concepts. The most important ones are described below.

The FMI follows a master-slave principle. In FMI for co-simulation different simulators can be coupled, if they are able to communicate data during simulation at certain time points. The master algorithm must handle data exchange between functional mock-up units (FMU) (Bastian et al., 2011). For example, it connects the output of an FMU to the input of another FMU. A co-simulation scenario represents a collection of interconnected FMUs. This introduces numerous challenges to the design of a master. The sequence of FMU calculations, or interpolation and extrapolation algorithms for FMUs operating with different step sizes represent some examples. A DCP master also connects the outputs of DCP slaves with the inputs of DCP slaves. In order to do so, a DCP master must be able to generate and roll out a configuration based on the intended simulation scenario (Krammer and Benedikt, 2018). In contrast to the FMI, the DCP also enables direct slave-to-slave communication. As an immediate consequence, dedicated coupling algorithms, like NEPCE (Benedikt et al., 2013) may only be applied if communication between DCP slaves is routed via the master.

The master-slave principle also follows economic goals. Slave providers agree on a standard, but com-

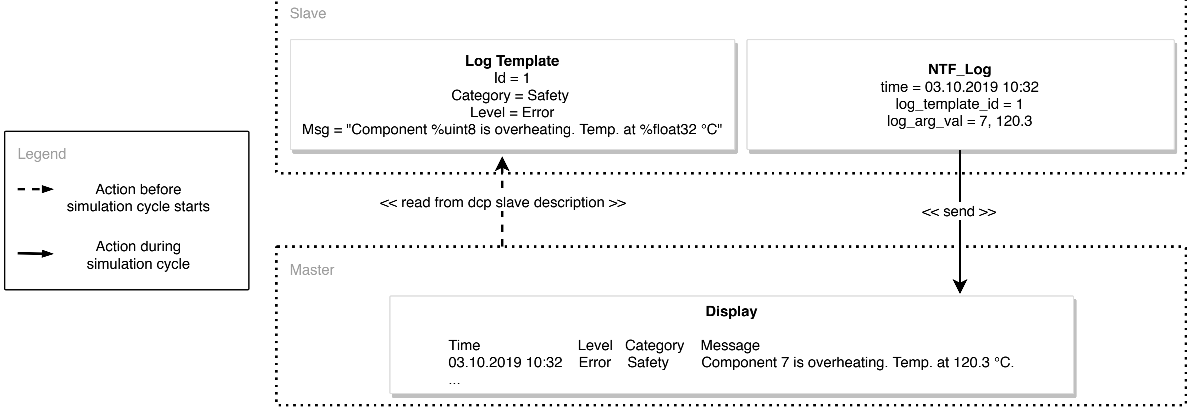

Slave
Log Template
Id = 1
Category = Safety
Level = Error
Msg = "Component %uint8 is overheating. Temp. at %float32 °C"
NTF_Log
time = 03.10.2019 10:32
log_template_id = 1
log_arg_val = 7, 120.3
Legend
Action before simulation cycle starts
Action during simulation cycle
<< read from dcp slave description >>
<< send >>
Master
Display
Time
Level
Category
Message
03.10.2019 10:32
Error
Safety
Component 7 is overheating. Temp. at 120.3 °C.
...

Figure 4. Example of log-on-notification mechanism.

pete in slave implementation. This allows an integrator to choose from best-in-class solutions. From a slave provider's perspective the market entrance barrier is lowered, since he is able to offer accessible solutions. Furthermore, the master algorithm, which is not standardized neither for FMI nor for DCP, may enable a stronger position on the market.

The FMI is operated using a state machine. Since state machines are one major method for the design and operation of communication protocols, the DCP was also defined on the basis of a state machine. The specification defines which PDUs can be sent and received in each state, the possible transitions between states, and the possible behaviour in each state. The DCP defines an `Initialization` superstate, which corresponds to the *Initialization Mode* of FMI.

The integration process of FMUs is supported by a standardized XML schema definition. It is used to generate one `modelDescription.xml` file per FMU. It contains the necessary information for instantiation and use of an FMU. It must be placed in the root directory inside an FMU, to allow an FMI master to read this information. Furthermore, a FMU may contain source code and/or compiled libraries. Due to the nature of DCP slaves, the inclusion of source code and/or compiled files within a DCP slave file is currently not explicitly specified.

4 Use Case

4.1 Overview

Typical use cases for the DCP include vehicle test benches, where real and virtual components are integrated into the same simulation scenario. This allows the execution of test cases that would not be possible in reality, due to cost, availability of components, or safety reasons. In this section we present a use case that is based on an engine testbed (PUMA from AVL List GmbH[3]) that interacts with a simulated vehicle and a simulated driver. A schematic overview of this use case is shown in Figure 5. The vehicle and the driver are simulated within one DCP slave ("Vehicle"), and the testbed available as another DCP slave ("Engine"). This use case is simulated as an SRT scenario. The connections of output variable to input variables between DCP slaves are shown as solid arrows in Figure 6.

4.2 Dependency Structures

The outputs of the DCP slave "Vehicle" are y_{torque} and y_{alpha}; the output of the DCP slave "Engine" is y_{speed}. The two DCP slaves are connected in the following

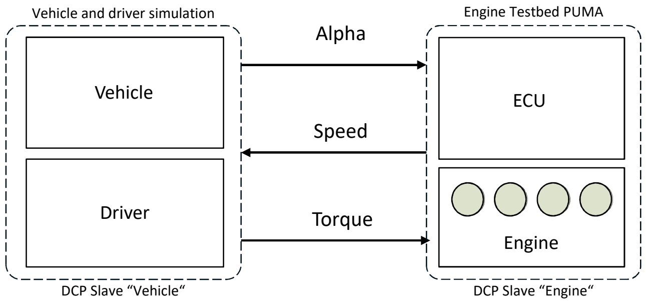


Figure 5. Vehicle-engine co-simulation use case.

way:

$$\begin{aligned}\text{Vehicle}.y_{\text{torque}} &\rightarrow \text{Engine}.u_{\text{torque}}\\ \text{Vehicle}.y_{\text{alpha}} &\rightarrow \text{Engine}.u_{\text{alpha}}\\ \text{Engine}.y_{\text{speed}} &\rightarrow \text{Vehicle}.u_{\text{speed}}\end{aligned}$$

To be able to start from a non-trivial start condition, both slaves declare parameters (Vehicle: $p^{\text{start}}_{\text{velocity}}$, $p^{\text{start}}_{\text{gear}}$, $p^{\text{start}}_{\text{alpha}}$; Engine: $p^{\text{start}}_{\text{speed}}$) that can be set in the `Initialization` superstate (see Section 3.2.2). In the `Initialization` superstate, the DCP slave "Vehicle" calculates the initial output as follows:

$$y^{\text{init}}_{\text{speed}} := f_{\text{speed}}(p^{\text{start}}_{\text{velocity}}, p^{\text{start}}_{\text{gear}}, \mathbf{p}_{\text{Vehicle}})$$

and the DCP slave "Engine" provides the initial output as follows:

$$y^{\text{init}}_{\text{alpha}} := f_{\text{alpha}}(u_{\text{torque}}, p^{\text{start}}_{\text{speed}}, \mathbf{p}_{\text{Engine}})$$

$\mathbf{p}_{\text{Vehicle}}$ and $\mathbf{p}_{\text{Engine}}$ are the vectors that contain all not explicitly mentioned parameters of the Vehicle and the Engine, respectively.

These initial outputs are used to set parameters (Engine.$p^{\text{start}}_{\text{speed}}$, Vehicle.$p^{\text{start}}_{\text{alpha}}$) of the opposite DCP slave:

$$\begin{aligned}\text{Vehicle}.y^{\text{init}}_{\text{speed}} &\rightarrow \text{Engine}.p^{\text{start}}_{\text{speed}}\\ \text{Engine}.y^{\text{init}}_{\text{alpha}} &\rightarrow \text{Vehicle}.p^{\text{start}}_{\text{alpha}}\end{aligned}$$

If the master uses an output value of one DCP-slave to set a parameter of another DCP-slave, we call this a parameters connection. Such parameter connections are shown in Figure 6 as dotted arrows.

The dependency of outputs on other variables may be different in the `Initialization` superstate and in the `Run` superstate. In the `Initialization` superstate, the outputs of the DCP slave "Vehicle" are calculated according to:

$$\begin{aligned}y_{\text{torque}} &:= f_{\text{torque}}(p^{\text{start}}_{\text{velocity}}, p^{\text{start}}_{\text{gear}}, \mathbf{p}_{\text{Engine}})\\ y_{\text{alpha}} &:= p^{\text{start}}_{\text{alpha}}\end{aligned}$$

The output y_{speed} of the DCP slave "Engine" in the `Initialization` superstate is determined by the parameter $p^{\text{start}}_{\text{speed}}$, i.e.:

$$y_{\text{speed}} := p^{\text{start}}_{\text{speed}}$$

[3]`http://www.avl.com`

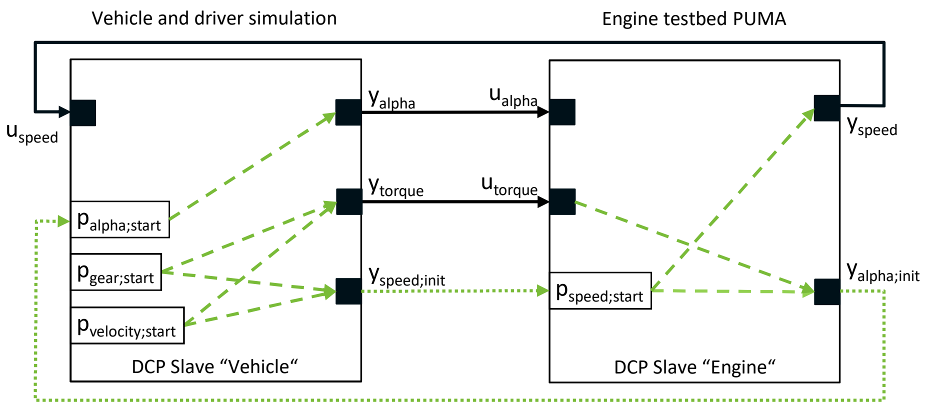


Figure 6. Vehicle-engine co-simulation scenario including dependency structure information during initialization.

4.3 Analysis

DCP slaves can provide information about the dependency structure of their outputs in the DCP slave description file (see Section 3.2.1). A DCP master may use this information to check if algebraic loops must be solved to achieve a consistent initial configuration. A graph may be used for such a check, where the nodes are variables of the DCP slaves. Each connection, parameter connection or dependency represents an edge of the graph. If the graph is acyclic, no algebraic loop needs to be solved. Note that without a given dependency structure, the DCP master would have to assume that each output depends on all inputs and parameters. The dependencies of outputs on inputs and parameters in the `Initialization` superstate of the described DCP slaves are shown in Figure 6. A dashed arrow from a variable x (an input or a parameter) to an output y indicates a dependency of y on x. It can be seen immediately that the graph does not contain any loops. Hence, a simple sequence of setting inputs/parameters after receiving output values is sufficient to achieve a consistent initial configuration. The DCP slaves state machines can subsequently be transitioned to superstate `Run` in order to perform synchronization (see Section 3.2.3) followed by the actual test case.

5 Standardized Solution

The Modelica Association[4] is a non-profit, non-governmental organization with members from Europe, North America, and Asia. Since 1996, its simulation experts have been working to develop the open standard Modelica and the open source Modelica Standard Library. Today it aims at coordinated standardization, development of software technology, and corresponding methods in the fields of cyber-physical systems and systems engineering. Currently the Modelica Association operates five Modelica Association Projects (MAP), where the DCP represents the most recent addition to the portfolio. The Modelica Association requires that all MAP results must be made available under an open source license.

The DCP was accepted as a MAP in 2018. The DCP specification document is initially published under a *Creative Commons Attribution Share-Alike 4.0* license[5]. The DCP slave description schema files, the DCP C++ reference implementation, and other supporting materials are initially published under a *BSD 3-clause* license[6].

MAP DCP follows its own rules. They are negotiated between its members and must be acknowledged by the Modelica Association. Contributions to MAP DCP are welcome. *Visitors* may contribute to MAP DCP in an informal way. *Advisory Committee* members actively support the design of the DCP. Its members must attend project meetings and sign a contributor's license agreement. They have access to development infrastructure, including mailing lists and file repositories. *Steering Committee* members have voting rights and define the strategy, feature roadmap, and future releases of the DCP. Furthermore, they must provide an implementation of the DCP specification, or part of it, in a commercial or open source tool. They should actively use DCP in industrial projects. Further information on these topics can be found on the DCP website[7].

6 Conclusion

The DCP enables integration of real-time systems and simulation environments in a standardized way. A stronger relationship between virtual and real worlds demands for new methodologies in simulation and test. Applications like automated driving, where high numbers of real world scenarios can be simulated before tests are conducted, can significantly benefit from the DCP.

[4]`http://www.modelica.org`

[5]`https://creativecommons.org/licenses/by-sa/4.0/`

[6]`https://opensource.org/licenses/BSD-3-Clause/`

[7]`http://www.dcp-standard.org`

The DCP specification version 1.0 is released by the Modelica Association. It represents the new state-of-the-art for co-simulation and test. The DCP is developed further by a consortium of original equipment manufacturers (OEM), simulation tool providers and software vendors, as well as suppliers for components and test equipment.

Despite the fact that the DCP was developed with other standards in mind, like the FMI, there are still challenges ahead. The FMI compatibility can still be improved, and the development of other software technologies like the SSP (System Structure and Parameterization) will require additional alignment activities in the future.

References

Jens Bastian, Christoph Clauß, Susann Wolf, and Peter Schneider. Master for Co-Simulation Using FMI. In *Proceedings of the 8th International Modelica Conference*, pages 115–120, 2011. doi:10.3384/ecp11063115.

Martin Benedikt, Daniel Watzenig, Josef Zehetner, and Anton Hofer. NEPCE - A nearly energy-preserving coupling element for weak-coupled problems and co-simulations. *International Conference on Computational Methods for Coupled Problems in Science and Engineering*, pages 1–12, 2013.

Torsten Blochwitz, Martin Otter, Martin Arnold, Constanze Bausch, Christoph Clauß, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, Manuel Monteiro, Thomas Neidhold, Dietmar Neumerkel, Hans Olsson, Jörg-Volker Peetz, and Susann Wolf. The functional mockup interface for tool independent exchange of simulation models. In *In Proceedings of the 8th International Modelica Conference*, pages 105–114, 03 2011. ISBN 978-91-7393-096-3. doi:10.3384/ecp11063105.

David Broman, Christopher Brooks, Lev Greenberg, Edward a. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of FMUs for co-simulation. In *2013 Proceedings of the International Conference on Embedded Software, EMSOFT 2013*, pages 1–12. Ieee, sep 2013. ISBN 9781479914432. doi:10.1109/EMSOFT.2013.6658580. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6658580`.

Thomas Doms, Benedikt Rauch, Bernhard Schrammel, Christoph Schwald, Edvin Spahovic, and Christian Schwarzl. Highly Automated Driving - The new challenges for Functional Safety and Cyber Security. White paper, TÜV Austria Holding AG and VIRTUAL VEHICLE, Vienna, Austria, 2018.

European Automobile Manufacturers Association. The Automobile Industry Pocket Guide 2018/2019. Technical report, European Automobile Manufacturers Association, Brussels, Belgium, 2018. URL `http://www.acea.be`.

Ned Freed and Dr. Nathaniel S. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045, November 1996. URL `https://rfc-editor.org/rfc/rfc2045.txt`.

International Telecommunication Union. Information technology – Open Systems Interconnection – Basic Reference Model: The basic model. ITU-T Recommendation X.200, International Telecommunication Union, 1994.

ISO/IEC JTC 1/SC 34. Information technology - Document Container File - Part 1: Core. Standard, International Organization for Standardization, Geneva, Switzerland, October 2015.

Martin Krammer and Martin Benedikt. Configuration of slaves based on the distributed co-simulation protocol. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 195–202. IEEE, 2018.

Martin Krammer, Nadja Marko, and Martin Benedikt. Interfacing Real-Time Systems for Advanced Co-Simulation - The ACOSAR Approach. In Catherine Dubois, Francesco Parisi-Presicce, Dimitris Kolovos, and Nicholas Matragkas, editors, *STAF 2016 Doctoral Symposium and Projects Showcase*, pages 32–39, Vienna, Austria, 2016. Dubois, Catherine Parisi-Presicce, Francesco Kolovos, Dimitris Matragkas, Nicholas.

Martin Krammer, Martin Benedikt, Torsten Blochwitz, Khaled Alekeish, Nicolas Amringer, Christian Kater, Stefan Materne, Roberto Ruvalcaba, Klaus Schuch, Josef Zehetner, Micha Damm-Norwig, Viktor Schreiber, Natarajan Nagarajan, Isidro Corral, Tommy Sparber, Serge Klein, and Jakob Andert. The distributed co-simulation protocol for the integration of real-time systems and simulation environments. In *Proceedings of the 50th Computer Simulation Conference*, SummerSim '18, pages 1:1–1:14, San Diego, CA, USA, 2018. Society for Computer Simulation International. URL `http://dl.acm.org/citation.cfm?id=3275382.3275383`.

Leonid Lichtenstein, Florian Ries, Michael Völker, Jos Höll, Christian König, Josef Zehetner, Oliver Kotte, Isidro Corral, Lars Mikelsons, Nicolas Amringer, Steffen Beringer, Janek Jochheim, Stefan Walter, Corinna Mitrohin, Natarajan Nagarajan, Torsten Blochwitz, Desheng Fu, Timo Haid, Jean-Marie Quelin, Rene Savelsberg, Serge Klein, Pacome Magnin, Bruno Lacabanne, Viktor Schreiber, Martin Krammer, Nadja Marko, Martin Benedikt, Stefan Thonhofer, Georg Stettinger, Markus Tranninger, and Thies Filler. Literature Review in the Fields of Standards, Projects, Industry, and Science. Technical report, ACOSAR Consortium, 2016.

Modelisar Consortium and Modelica Association Project "FMI". Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0, 2014.

Hubert Zimmermann. OSI reference model–The ISO model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28(4):425–432, 1980.

SESSION 1D: AUTOMOTIVE 1

Anti-Roll Bar Modeling for NVH and Vehicle Dynamics Analyses

Jakub Tobolář[1] Martin Leitner[1] Andreas Heckmann[1]

[1]German Aerospace Center (DLR), Institute of System Dynamics and Control, Wessling,
Jakub.Tobolar@DLR.de

Abstract

The latest extension of the *DLR FlexibleBodies* Library concerns the field of automotive applications, namely the anti-roll bar. For the particular purposes of NVH and vehicle dynamics, the anti-roll bar module provides two appropriate levels of detail, both being based upon the beam preprocessor. In this paper, the procedure on preparing the models and their application for particular automotive related analyses is presented.

Keywords: anti-roll bar, vehicle chassis, flexible body, beam model, finite element

1 Introduction

Whenever an automotive suspension is excited in vertical direction due to road irregularities or driving maneuvers in an asymmetrical way, i.e. differently on the right and the left side of the vehicle, the roll motion of the car body is stimulated. This concerns – in common case – the comfort and driving experience of the car passengers. In limit conditions' situations, such a roll motion can influence the road-holding forces in a way that vehicle's driving safety is affected significantly. Consequently, it is advantageous to introduce an additional suspension component in particular tailored to influence the dynamical roll motion characteristics independently from the layout of the vertical suspension. This so-called anti-roll bar (also called stabilizer or anti-sway bar), see e.g. (Rill, 2012) or (Heißing and Ersoy, 2011), connects the suspensions on the right and the left side of the vehicle's axle by a cranked bar that acts as a torsional spring, see Figure 1.

Therefore, the design of the anti-roll bar is mainly targeted on its torsional stiffness, but also has to comply with the available space at the underfloor and must allow for attachments to the vehicle body and to both vertical suspensions. These requirements quite often result in the anti-roll bar to be a geometrical complex structural element that is prone for dynamical vibrations.

In daily practice, the Finite Element (FE) method turned out to be the adequate tool to design the geometrical and the structural properties of anti-roll bars. However, the driving behavior of vehicles, to which the anti-roll bar significantly contributes, is commonly developed using multibody simulation – generally utilizing the *MultiBody* package of the Modelica Standard Library in the Modelica community. In addition, the *DLR FlexibleBodies* Library

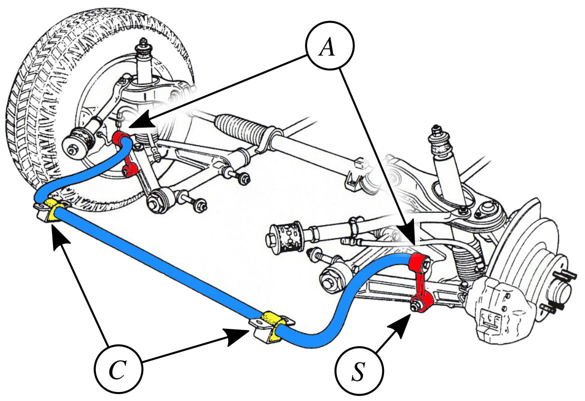


Figure 1. Vehicle axle with an anti-roll bar (color emphasized, courtesy of Wikimedia Commons).

(Heckmann et al., 2006) provides capabilities to incorporate data that originate from FE models in Modelica models. Thus, a tool chain to perform vehicle dynamics simulation including the structural characteristics of anti-roll bars is in principle available.

In common design tasks, driving maneuvers or noise, vibration and harshness (NVH) scenarios are first analyzed in multibody simulations. Then, the FE method is used to redesign the anti-roll bar in order to improve its characteristics. Subsequently, a FE to multibody interface has to be used to prepare the new FE data for the *DLR FlexibleBodies* Library and, finally, the vehicle dynamics simulation has to be invoked again in order to assess the modification. This tool chain or loop, respectively, is inconvenient and makes it difficult to set up computational optimization procedures.

The given background motivates the introduction of a new modeling capability called *AntiRollBar* into the *DLR FlexibleBodies* Library. In the present paper, a principle of the flexible body modeling and of the beam theory behind the *AntiRollBar* model is given in Sections 2.1 and 2.2, respectively. In Section 3, a framework of the *AntiRollBar* and its parametrization is discussed. Section 4 presents first simulation experiments provided.

2 Theoretical Background

2.1 Flexible Bodies Theory

The mechanical description of flexible bodies in multibody systems is based on the floating frame of reference

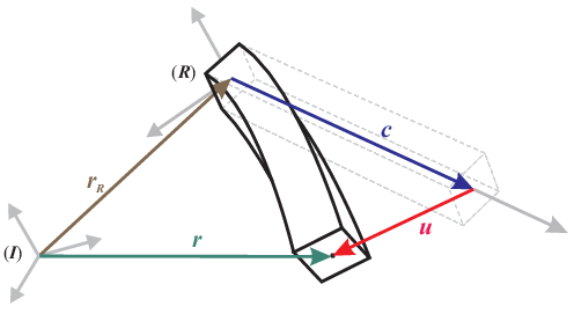


Figure 2. Vector chain of the floating frame of reference.

approach, i.e. the absolute position[1] $\boldsymbol{r} = \boldsymbol{r}(\boldsymbol{c},t)$ of a specific body particle is subdivided into three parts:

- the position vector $\boldsymbol{r}_R = \boldsymbol{r}_R(t)$ to the body's reference frame,
- the initial position of the body particle within the body's reference frame, i.e. the Lagrange coordinate $\boldsymbol{c} \neq \boldsymbol{c}(t)$,
- and the elastic displacement $\boldsymbol{u}(\boldsymbol{c},t)$ that is approximated by a Taylor expansion, here limited to first order terms, with space-dependent mode shapes $\boldsymbol{\Phi}(\boldsymbol{c}) \in \mathbb{R}^{3,n}$ and time-dependent modal amplitudes $\boldsymbol{q}(t) \in \mathbb{R}^n$, cf. (Wallrapp, 1994):

$$\boldsymbol{r} = \boldsymbol{r}_R + \boldsymbol{c} + \boldsymbol{u}\,, \qquad \boldsymbol{u} = \boldsymbol{\Phi}\,\boldsymbol{q}\,. \tag{1}$$

All terms in equation (1) are resolved w.r.t. the body's floating frame of reference (R). That's why the angular velocity of the reference frame $\boldsymbol{\omega}_R$ have to be taken into account when the kinematic quantities velocity $\boldsymbol{v}$ and acceleration $\boldsymbol{a}_R$ of a particle are derived, see (Heckmann et al., 2006). The decomposition in equation (1) makes it possible to superimpose a large nonlinear overall motion of the reference frame with small elastic deformations.

The kinematic quantities are inserted into Jourdain's principle of virtual power. Subsequently, the equations of motion of an unconstrained flexible body are formulated neglecting deflection terms of higher than first order (Wallrapp, 1994, (38)):

$$\begin{bmatrix} m\boldsymbol{I}_3 & & \textit{sym.} \\ m\tilde{\boldsymbol{d}}_{CM} & \boldsymbol{J} & \\ \boldsymbol{C}_t & \boldsymbol{C}_r & \boldsymbol{M}_e \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_R \\ \dot{\boldsymbol{\omega}}_R \\ \ddot{\boldsymbol{q}} \end{bmatrix} = $$
$$= \boldsymbol{h}_\omega - \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{K}_e\boldsymbol{q} + \boldsymbol{D}_e\dot{\boldsymbol{q}} \end{bmatrix} + \boldsymbol{h}_e, \tag{2}$$

[1]Both vectors and matrices are written in bold symbols, whereby vectors are of lower case letters and matrices of upper case letters.

where the following quantities and symbols appear:

m	body mass,
$\boldsymbol{I}_3$	3×3 identity matrix,
$\boldsymbol{d}_{CM}(\boldsymbol{q})$	position of center of mass,
$\boldsymbol{J}(\boldsymbol{q})$	inertia tensor,
$\boldsymbol{C}_t(\boldsymbol{q})$	inertia coupling matrix (translational),
$\boldsymbol{C}_r(\boldsymbol{q})$	inertia coupling matrix (rotational),
$\boldsymbol{h}_\omega(\boldsymbol{\omega},\boldsymbol{q},\dot{\boldsymbol{q}})$	gyroscopic and centripetal forces,
$\boldsymbol{h}_e$	external forces,
$\boldsymbol{M}_e$	structural mass matrix,
$\boldsymbol{K}_e$	structural stiffness matrix,
$\boldsymbol{D}_e$	structural damping matrix.

In the context of the anti-roll bar modeling, the structural mass, stiffness and damping masses are gained as the result of a FE preprocessing step whose background is given in the following section. Note, that the FE preprocessing is implemented internally so that the user does not need to switch to a different modeling tool.

2.2 FE Beam Theory

The structural models used in multibody analysis are usually obtained from FE analysis and subsequently reduced by e.g. modal decomposition approaches. For the anti-roll bar structural models, a simple, classical finite element beam formulation is employed. Therein, the three-dimensional problem is split into a two-dimensional, cross-sectional analysis and a subsequent, one-dimensional analysis along the beam's reference axis. Solving the two-dimensional problem simply involves integration of the material properties (Young's modulus, shear modulus and density) over the specified cross-sectional geometry. With the resulting cross-sectional stiffness and inertia resultants, the corresponding constitutive matrix $\boldsymbol{C}$ can be built. Along with a given strain field, one arrives at the description for the force and moment distributions along the beam axis.

The one-dimensional analysis is based on (Bazoune et al., 2003), where an adjustable Timoshenko beam element was implemented, that uses linear shape functions for longitudinal displacements and torsional deformation. In order to describe the bending deformation, a cubic ansatz function is used in the lateral displacements and corresponding rotational fields. The unknown coefficients can be solved using the description for the total slopes including a constant transversal shear, the force and moment equilibrium equations and the discrete boundary conditions at both ends of the beam. These equations can then be partitioned by discerning between displacement field contribution and discrete boundary condition excitation. In a parametric space from [0,1], the shape functions that are needed for the shear displacements and bending rotations read:

$$
\begin{aligned}
N_{bs}^1 &= \frac{1}{1+\Theta}(1-3\xi^2+2\xi^3),\\
N_{bs}^2 &= \frac{1}{1+\Theta}(3\xi^2-2\xi^3),\\
N_{bb}^1 &= \frac{1}{1+\Theta}(\xi-2\xi^2+\xi^3+1/2(2\xi-\xi^2)\Theta),\\
N_{bb}^2 &= \frac{l}{1+\Theta}(-\xi^2+\xi^3+1/2\xi^2\Theta),\\
N_{ss}^1 &= \frac{\Theta}{1+\Theta}(1-\xi),\\
N_{ss}^2 &= \frac{\Theta}{1+\Theta}\xi,\\
N_{sb}^1 &= -\frac{\Theta l}{1+\Theta}1/2\xi,\\
N_{sb}^2 &= -\frac{\Theta l}{1+\Theta}1/2\xi, \qquad (3)
\end{aligned}
$$

where ξ denotes the parametric coordinate along the beam axis, l the beam length and Θ the bending to shear stiffness ratio. The shape function superscripts indicate the left (1) or right (2) beam end and the two-letter subscripts specify the shear (s) and bending (b) field types (first letter) and the contributing boundary condition (second letter). As previously described, their derivatives w.r.t. the parametric coordinates and the constitutive equations of cross-sectional resultants can then be used to calculate the shear and bending moment distributions along the beam and integrated using e.g. a Gauss' quadrature.

In accordance with Galerkin's weighted residual method, one can substitute into the classic expression of virtual work done by all internal forces, to arrive at a formulation for the linear element stiffness matrix $\boldsymbol{K}$, which in matrix form reads,

$$\boldsymbol{K} = \int_l \boldsymbol{B}^T \boldsymbol{C} \boldsymbol{B}\,\mathrm{d}l, \qquad (4)$$

where $\boldsymbol{B}$ is the matrix of shape function derivatives describing the element strain field. Rotating all element stiffness matrices into the reference inertial coordinate system and assembly in a unified degree-of-freedom set, leads to the total stiffness matrix of the structure. In a similar fashion the consistent mass matrix $\boldsymbol{M}$ can be determined, using cross-sectional inertia resultants and the matrix of shape functions N. In case of the anti-roll bar a decoupled, lumped mass approach was chosen instead, where structural inertia is distributed equally at both ends of the beam. Shear center, neutral axis and center of gravity offsets from the reference line were realized as wrappers around the element stiffness and element mass matrices in order to be able to tailor the bar properties more accurately.

To facilitate a further reduction of the number of degrees of freedom, an optional Guyan's reduction (Guyan, 1964) can be performed next. The Guyan's reduction is essentially a static residualization of stiffness onto a chosen few degrees of freedom (DoF). Therein, all DoF are partitioned into an "analysis" set (index a) and an "omitted" set (index o) and a constraint w.r.t. the omitted degrees of freedom being force free is applied. The resulting transformation matrix between the analysis set and the original, full DoF set reads,

$$\begin{bmatrix}\boldsymbol{x}_a\\ \boldsymbol{x}_o\end{bmatrix} = \begin{bmatrix}\boldsymbol{I}\\ -\boldsymbol{K}_{oa}^{-1}\boldsymbol{K}_{oa}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_a\end{bmatrix}. \qquad (5)$$

In order for the mass and stiffness matrix to be compatible with the generalized equations of motion, a normal modes analysis is performed afterwards. The resulting eigenfrequencies and eigenvectors form a modal solution set that is used to reduce the structural degrees of freedom to the user-specified number of frequencies and mode shapes retained. Additional information required for the flexible multibody approach such as, center of gravity location, mass, inertia tensor and the inertia coupling terms are then calculated with the help of six, linearly independent rigid body mode shapes. Currently only first order inertia terms are considered, while integration of the nonlinear, second order terms is still subject to future work.

3 Beam Based Anti-Roll Bar Model

The introduced beam model can be utilized in various applications. In the following, we focus on a typical use case in the automotive area – the anti-roll bar.

The *AntiRollBar* model implemented in *DLR FlexibleBodies*, see the model's icon in Figure 3, allows for the modeling of flexible bars with an (almost) arbitrary geometrical shape in a user-friendly way and considers the attachments to the vehicle body and the suspensions elaborated in Section 3.1. It is tailored to be used for driving maneuvers, where frequencies higher than 20 Hz are out of interest but require unnecessarily large computational resources. Alternatively, the user may specify the *AntiRollBar* model to be employed for NVH analysis up to 400 Hz. And last but not least, an animation of the new *AntiRollBar* model and its deformation field is also provided in order to assess simulation results visually.

3.1 Anti-Roll Bar Arrangement

To reduce the number of input parameters, the common shape of the anti-roll bar and its mounting to the vehicle's parts are considered. As can be seen in Figure 1, there are

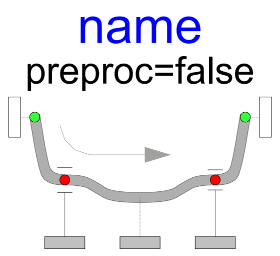


Figure 3. Icon of the new *AntiRollBar* model in the *DLR FlexibleBodies* Library.

typically two mounts (C) on the vehicle's body holding the anti-roll bar. To fulfill its operational goal of stabilizing the rolling vehicle, each of the anti-roll bar's ends is additionally connected to a suspension part on each vehicle's side (S). Thus, exactly four mounting points are incorporated to connect the implemented beam based *AntiRollBar*. Since the common anti-roll bar is made of semifinished tube with ring-shaped cross section, the input parameters are additionally limited to outer and inner diameter of the cross section.

Considering the abovementioned restrictions, the geometry input reduces to Cartesian coordinates x, y and z of meaningful geometry points along the anti-roll bar's center line and its outer and inner diameters (d_o and d_i, respectively) at these points. Additionally, four of the points have to be marked as mountings to the vehicle. Consequently, the input reads as:

```
// x      y      z     do     di    connect
  0.10   0.51   0.0   0.02   0.014    1
 -0.17   0.42   0.0   0.02   0.014    0
 -0.20   0.38   0.0   0.02   0.014    0
   ...
```

Note that the four points relevant for mountings are marked in the last column by "1".

The input parameters of the implemented *AntiRollBar* are detailed in the following sections. They reflect especially the two intended application areas of the *AntiRollBar* implementation – the vehicle handling analysis and the NVH. Another important aspect – which applies for both of the analyses – is the option to either input some particular predefined data of the analyzed anti-roll bar or to calculate it in preprocessing steps by the *AntiRollBar* model itself. The workflow of the latter is depicted in Figure 4.

3.2 Parameters for Noise, Vibration and Harshness Analysis

Let us consider the parameter input mask as shown in Figure 5 first. Here, the first parameter labeled *Analysis* specifies the option to activate the model for the NVH. Thus, the flexible body model based on a modal description will be activated in the *AntiRollBar* model background – in particular the *ModalBody* component of the *DLR FlexibleBodies* library. This submodel incorporates a plenty of parameters of which just three are present in the input mask of the *AntiRollBar*, namely:

- *fileFlexBody* – a shared name of files which describe the flexible body dynamics and animation – *SID* [2] and *obj* [3], respectively. To simplify the input, this name is required without the file suffix, assuming that both SID and obj files of the modeled anti-roll bar have the same name.

[2]Standard Input Data file

[3]File with 3D data in Wavefront OBJ format, see e.g. http://www.fileformat.info/format/wavefrontobj/egff.htm

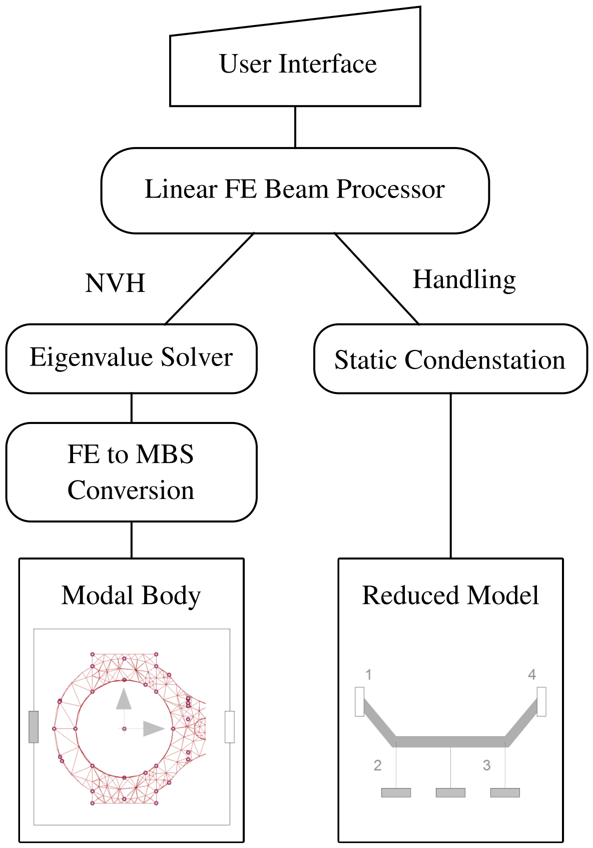
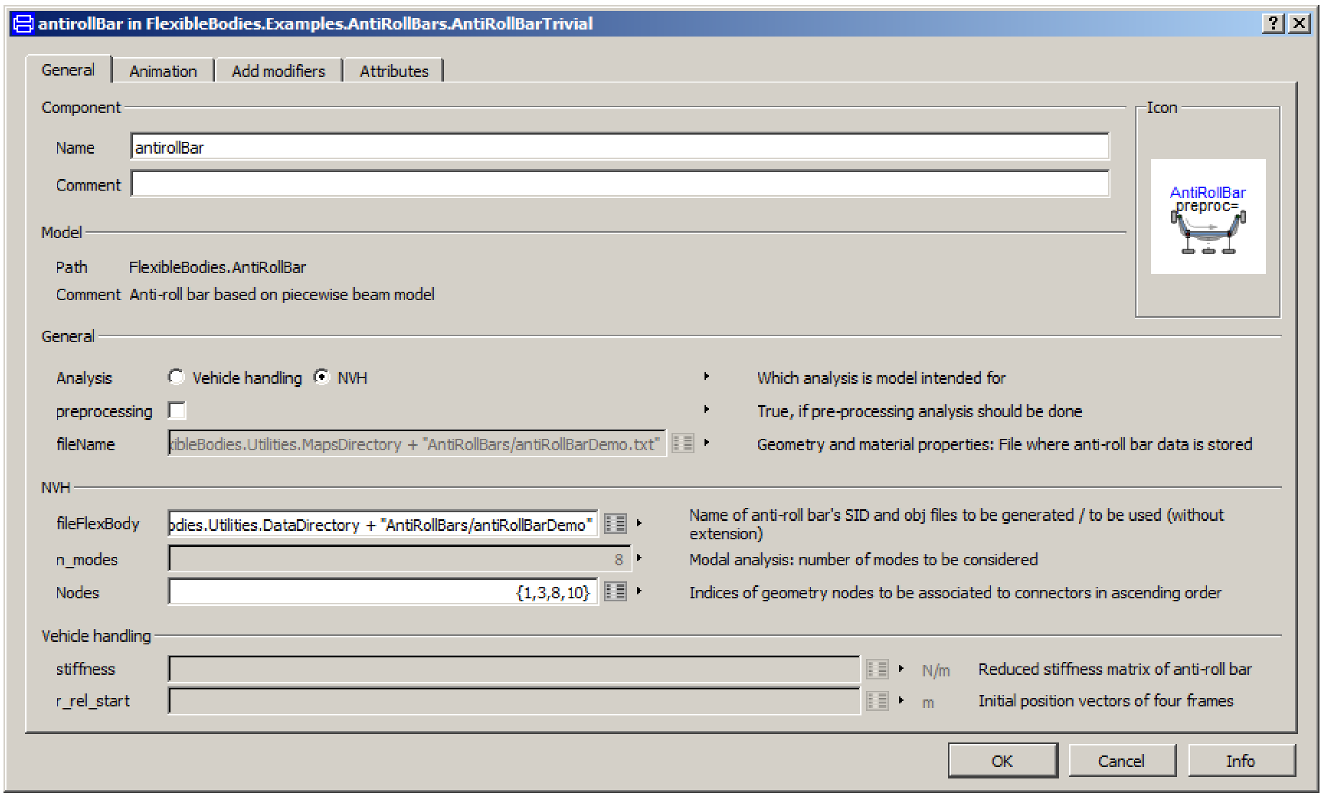


Figure 4. Preprocessing steps of the *AntiRollBar* model with two branches for NVH or vehicle handling scenarios.

- *n_modes* – the number of eigenmodes to be considered.
- *Nodes* – (exactly four) specific node numbers to be associated to the *AntiRollBar* connector frames.

At this point, the meaning of the parameter *preprocessing* shall be further explained. In Figure 5, this parameter is set to *false*. Thus, it is required that the user inputs both the SID and obj files by defining *fileFlexBody* and the indexes of the connector nodes using *Nodes*. The information on the number of eigenmodes is, in contrast, not relevant, and therefore disabled.

The situation changes when the user wishes to generate the data in a preprocessing step setting *preprocessing = true*. Then, *fileFlexBody* indicates no more the name of the existing files but the name of files to be generated by the preprocessor, see below, and *n_modes* is the information being additionally required. The parameter array *Nodes* is then read from the last column of the geometry table shown in Section 3.1. This table, called *geometry*, has to be saved in the input file indicated by the parameter *fileName*. This file must additionally contain a table called *material* with anti-roll bar material properties. In particular, material density ρ, Young's modulus E and shear modulus G are required in the current *AntiRollBar* implementation. An example on the input data file format is given in Appendix A. For better understanding on the


antirollBar in FlexibleBodies.Examples.AntiRollBars.AntiRollBarTrivial

General | Animation | Add modifiers | Attributes

Component

Name	antirollBar
Comment	

Model

Path	FlexibleBodies.AntiRollBar
Comment	Anti-roll bar based on piecewise beam model

General

Analysis	Vehicle handling [✓ NVH]	Which analysis is model intended for
preprocessing	☐	True, if pre-processing analysis should be done
fileName	ibleBodies.Utilities.MapsDirectory + "AntiRollBars/antiRollBarDemo.txt"	Geometry and material properties: File where anti-roll bar data is stored

NVH

fileFlexBody	odies.Utilities.DataDirectory + "AntiRollBars/antiRollBarDemo"	Name of anti-roll bar's SID and obj files to be generated / to be used (without extension)
n_modes	8	Modal analysis: number of modes to be considered
Nodes	{1,3,8,10}	Indices of geometry nodes to be associated to connectors in ascending order

Vehicle handling

stiffness		N/m	Reduced stiffness matrix of anti-roll bar
r_rel_start		m	Initial position vectors of four frames

Figure 5. Parameter menu to specify *AntiRollBar* model.

preprocessing procedure, the steps are highlighted in the left branch of Figure 4.

As mentioned above, there is a particular output if the preprocessing is enabled for the NVH analysis. Then, two files are generated – one in SID and the other one in wavefront format. The SID file, see (Wallrapp, 1994), contains the modal reduced anti-roll bar structure where the input *n_modes* defines the number of retained modes. The Wavefront file (signalized by an *obj* suffix) enables the visualization of the anti-roll bar, see Figure 6. As common to *DLR FlexibleBodies* library, the anti-roll bar is visualized by both solid and wireframe elements, whereby the number of vertices and face elements of the wireframe grid can be influenced by the user.

3.3 Parameters for Vehicle Dynamics

The next *Analysis* option is the one for the vehicle dynamics. This case utilizes the following parameters:

- *stiffness* – a 4×4 stiffness matrix and
- *r_rel_start* – a 4×3 matrix containing Cartesian coordinates to get the proper position of the four connector frames.

The simplified matrix *stiffness* applies according to equation (5) for purely vertical DoF's considered in the four anti-roll bar mountings. If both *stiffness* and *r_rel_start* should be generated by preprocessor, the input simplifies to only *fileName*. The right branch of Figure 4 shows all necessary preprocessing steps in this case.

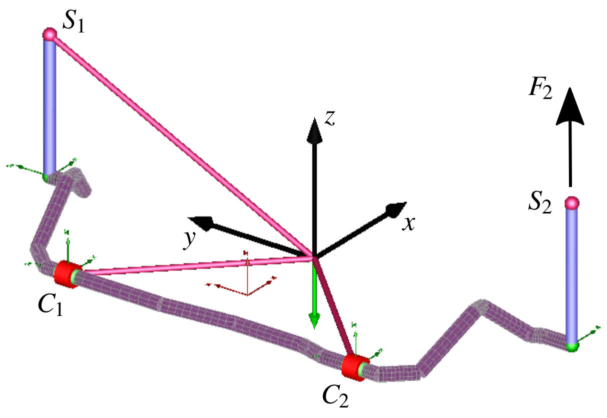


Figure 6. Anti-roll bar model with a connecting link on each side.

4 Simulation Experiment

For parametrization of the *AntiRollBar*, an important question concerns the number of eigenmodes which have to be considered when generating the SID file for NVH analysis. This influences not only the simulation results significantly, but also the simulation time.

To evaluate this phenomena we have defined a simple virtual experiment, depicted in Figure 6, which is intended to excite all the considered eigenmodes – similarly to an experimental modal analysis with single point excitation. In this experiment, the anti-roll bar is connected to the inertial frame at the vehicle's body mounts C_1 and C_2 via two

Table 1. Eigenfrequencies of the evaluated anti-roll bar.

Eigenmode	*Eigenfrequency* / Hz
1	61,7
2	100,7
3	121,9
4	139,9
5	217,2
6	321,4
7	375,0
8	467,9
9	521,6
10	556,8
11	640,0
12	817,8
13	927,0

spring-damper elements. Additionally, a link with two ball joints – a typical connection in automotive applications – is used to join one anti-roll bar's end to the inertial frame at fixed point S_1. The other end is connected similarly to point S_2 which, in contrast, can freely move in vertical direction (z-axis in Figure 6).

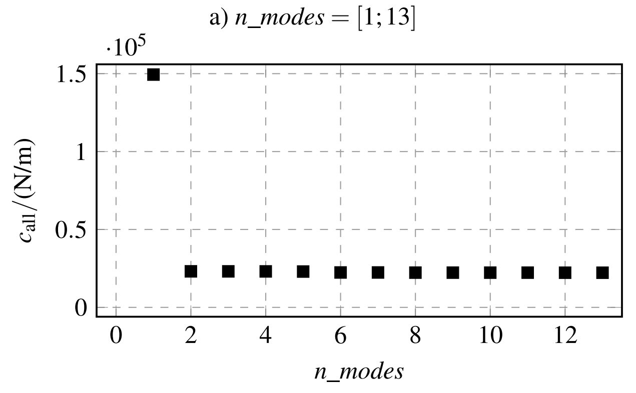
In the simulation scenario, the anti-roll bar is first preloaded at S_2 with a constant vertical force F_2. After some time period which guarantees that the mechanism is at rest, the link connection is "released" by fast drop of the force towards zero. Thus, a desired damped oscillation of the anti-roll bar around its unloaded state is induced. With this procedure, two criteria can be assessed: a) the overall stiffness of the anti-roll bar by relating the applied force F_2 to the static displacement of point S_2 and b) the simulation time t_{CPU}.

The anti-roll bar used in this example is asymmetric in vehicle's longitudinal plane, see also Appendix A for particular data input. For this geometry, the eigenmodes up to the frequency $f = 1000\,\mathrm{Hz}$ are given in Table 1.

The Figures 7 and 8 show the resulting overall stiffness c_{all} of the anti-roll bar and the simulation times t_{CPU}, respectively, over the number of considered modes *n_modes*. Since there is a significant drop in c_{all} between $n\_modes = 1$ and $n\_modes = 2$ in Figure 7 a), the parameter c_{all} is plotted in Figure 7 b) without the value for $n\_modes = 1$.

The deployment c_{all} in Figure 7 b) reflects the influence of single modes on the anti-roll bar stiffness. The eigenmode 2 is the most significant as there is an extraordinary change in the stiffness. Another change can be observed for eigenmode 6. For higher eigenmodes, the modifications in c_{all} are marginal and – as can be seen in Figure 8 – only lead to unnecessary increase of the simulation time.

Consequently, at least the first six eigenmodes, i.e. $n\_modes = 6$, should always be considered for the evaluated anti-roll bar's geometry. This applies even for NVH analyses in lower frequency range of interest. A higher



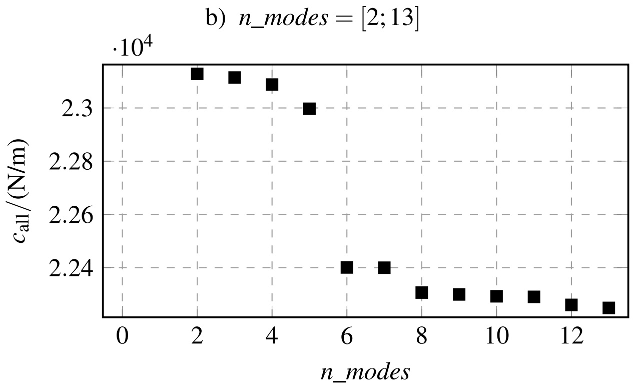


Figure 7. Anti-roll bar stiffness c_{all} for changing number of modes.

number of eigenmodes could nevertheless be introduced, but then a progressive increase of simulation time has to be taken into account.

5 Conclusions

The presented automotive anti-roll bar model can be applied for both the vehicle dynamics and the NVH analysis. The simulation experiment emphasizes the changes in model behavior depending on the structure's eigenmodes and shows how to indicate significant eigenmodes, which should always be included in the analyses. Due to the dependency of the eigenmodes on the anti-roll bar's geom-

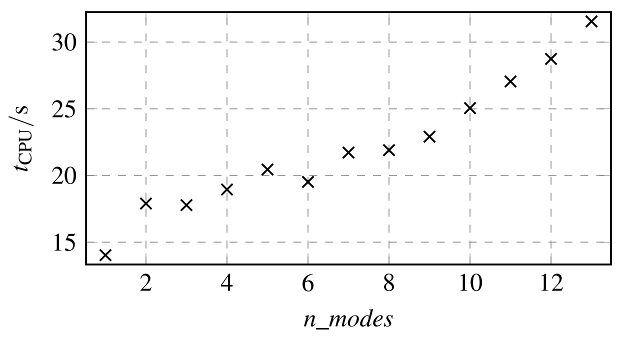


Figure 8. Simulation time t_{CPU} for $n\_modes = [1;13]$.

etry and material data, this identification has to be performed for each particular anti-roll bar.

The future development of the presented model will focus on the incorporation of structural damping as input parameter. Moreover, higher order models should be implemented for higher model fidelity.

Acknowledgements

The authors would like to thank to Frank Su and Jim Alanoly from Ford Motor Company, Dr. Bilal Maiteh from Dassault Systemes and John Batteh, John Griffin and Johan Andreasson, all from Modelon AB, for initialization of this work and important support provided at the beginning period of the project.

References

A. Bazoune, Y. A. Khulief, and Stephen N. G. Shape Functions of Three-Dimensional Timoshenko Beam Elements. *Journal of Sound and Vibration*, pages 473–480, 2003.

R. J. Guyan. Reduction of stiffness and mass matrices. *AIAA Journal*, 3:380, 1964.

A. Heckmann, M. Otter, S. Dietz, and J. D. López. The DLR FlexibleBody library to model large motions of beams and of flexible bodies exported from finite element programs. In *5th International Modelica Conference*, pages 85–95, 2006.

B. Heißing and M. Ersoy. *Chassis Handbook*. Vieweg+Teubner Verlag, 1 edition, 2011. DOI 10.1007/978-3-8348-9789-3.

G. Rill. *Road Vehicle Dynamics: Fundamentals and Modeling*. CRC Press, 2012. ISBN 978-1-4398-3898-3.

O. Wallrapp. Standardization of flexible body modeling in multibody system codes, Part 1: Definition of standard input data. *Mechanics of Structures and Machines*, 22(3):283–304, 1994.

A Appendix: Example of Input Data File

```
#1

# Map containing anti-roll bar data
# =================================
# Table "material"
#   Size: [3,1]
#   Contains material data:
#     Density rho [kg/m^3]
#     Young’s modulus E [N/m2]
#     Shear modulus G [N/m2]
#
# Table "geometry"
#   Size: [:,6]
#   Contains geometry data.
#   No. of rows = No. of geometry relevant
#   points on anti-roll bar’s center line.
#
# Note:
#   1) Both symmetric and non symmetric
#      anti-roll bar is applicable.
#   2) Exactly four connectors are
#      available at the moment,
#      i.e. the last column must contain
#      exactly four non zero integers
#      (the value of it plays no role).
#   3) The points are given successively
#      from the left end-point to the
#      right end-point.
#   4) Diameter outer must always be
#      greater then diameter inner.
#
double material(3,1)
  7.86e3
  2.07e11
  7.90e10
#
#  x      y      z     do      di    connect
double geometry(21,6)
  0.06   0.62  0.03  0.024  0.018  1
  0.06   0.60  0.03  0.024  0.018  0
  0.04   0.60  0.08  0.024  0.018  0
  0.00   0.60  0.10  0.024  0.018  0
 -0.06   0.58  0.05  0.024  0.018  0
 -0.11   0.56  0.04  0.024  0.018  0
 -0.16   0.51  0.04  0.024  0.018  0
 -0.16   0.46  0.04  0.024  0.018  1
 -0.16   0.10  0.04  0.024  0.018  0
 -0.23  -0.06  0.02  0.024  0.018  0
 -0.23  -0.24  0.02  0.024  0.018  0
 -0.23  -0.30  0.02  0.024  0.018  0
 -0.16  -0.41  0.04  0.024  0.018  0
 -0.16  -0.46  0.04  0.024  0.018  1
 -0.16  -0.51  0.04  0.024  0.018  0
 -0.11  -0.56  0.04  0.024  0.018  0
 -0.06  -0.58  0.07  0.024  0.018  0
  0.02  -0.60  0.10  0.024  0.018  0
  0.05  -0.60  0.08  0.024  0.018  0
  0.06  -0.60  0.03  0.024  0.018  0
  0.06  -0.62  0.03  0.024  0.018  1
```

System level heat pump model for investigations into thermal management of electric vehicles at low temperatures.

James Jeffs[1] Dr. Andrew McGordon[1] Alessandro Picarelli[2] Dr. Simon Robinson[3] Dr. W. Dhammika Widanage[1]

[1]WMG, University of Warwick, United Kingdom, j.jeffs@warwick.ac.uk
[2]Claytex ltd., United Kingdom, alessandro.picarelli@claytex.com@company
[3]Jaguar Land Rover, United Kingdom, srobin43@jaguarlandrover.com

Abstract

One of the challenges concerning electric vehicles is their performance in cold climates. As the temperature drops below 10°C battery capacity begins to reduce and heating demand starts to claim a larger proportion of total vehicle energy expenditure. Although efficient, electric vehicles waste heat through a few components, resulting in opportunity to harvest waste heat through a heat pump. With multiple options for harvesting heat and the option to heat the battery, a model and architecture has been developed to give flexibility in a wide range of thermal management scenarios. This paper explores the details of the model and presents two example cases of interest to demonstrate the model's applicability.

Keywords: Electric vehicle, thermal management, heat pump

1 Introduction

With electric vehicles beginning to take a larger market share of new vehicle sales, one common concern of electric vehicle drivers is the choice they may have to make at low temperature between heating and range (Allen, 2013; Bullis, 2013). Operation of electric vehicles below 10°C is hindered by the effect of low temperatures on battery capacity and the increased electrical consumption caused by cabin heating (Meyer et al., 2012). The combined effect of decreased battery performance and increased heating demand gives a range loss of up to 60% at −20°C compared to 20°C. Heat pumps are slowly being introduced as a solution to this problem; however the flexibility of heat pumps creates an opportunity for new, innovative and complex thermal management solutions, with many operational modes available for consideration and exploration (Jeffs et al., 2018).

1.1 Battery performance at low temperatures

Many investigations have been carried out into the performance of Lithium ion cells at low temperatures. The three areas which are cause for concern are ageing, power and capacity. The consensus on ageing of cells at low temperatures is that charging causes the formation of metallic lithium on the cathode, a process known as lithium plating. Lithium plating is associated with fast charging of a cell (over 0.5C, where 1C is the the current required to charge the cell in one hour) and so can be avoided by charging slowly and reducing the power harvested through regenerative braking. Power reduction has also been a concern when operating cells at low temperatures. Rui (Rui et al., 2011), Zheng (Zheng et al., 2016) and Jaguemont (Jaguemont et al., 2016) concluded that the primary cause of power capability reduction was due to an increase in charge transfer resistance. However, it has also been shown that for a typical pack sizing, a 70% reduction in power can be sustained while the vehicle is still able to complete usual drive cycles (UDDS, HWFET, US06) (Saxena et al., 2015). Reflecting on this literature, precautions are built into the model to take account of power reduction and increased ageing.

The biggest concern regarding electric vehicle operation at low temperatures is the reduction in range associated with reduced capacity. Much research has been published reporting cell capacity as a function of temperature and some examples are summarised in Figure 1 which displays the work of Nagasubramanian (Nagasubramanian, 2001), Zhang (Zhang et al., 2003), Ji (Ji et al., 2013), Jaguemont (Jaguemont et al., 2014), Dow Kokam (Dow, 2010) (manufacturer) and Panasonic (Pan, 2012) (manufacturer). The general consensus reached through this selection of work is that cell capacity decreases by approximately $20-40\%$ at -20°C and by as much as 70% at -40°C.

1.2 Cabin Heating

Cabin heating is the greatest non-powertrain consumer of energy on a vehicle at low temperatures (Lindgren and Lund, 2016; Broglia et al., 2012). In early production electric vehicles such as the first generation Nissan Leaf, the cabin heat was provided by a positive thermal coefficient (PTC) heater. PTC heaters are close to 100% efficient, but generate all of their heat using electrical energy from the battery. Due to the high demand of cabin heating, typically 4-7kW, this has a significant impact on the vehicle's range. Meyer *et al.* (Meyer et al., 2012) investigated the split between the impact of cabin heating and low temperature battery effects; performing tests with heating on

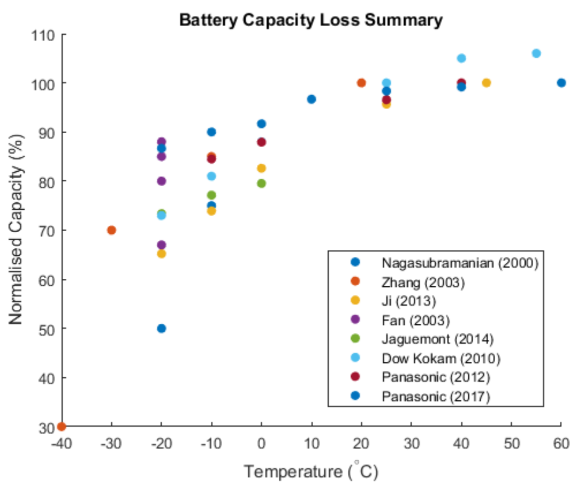


Figure 1. A summary of the capacity loss a function of temperature as reported by the following sources; Nagasubramanian (Nagasubramanian, 2001), Zhang (Zhang et al., 2003), Ji (Ji et al., 2013), Jaguemont (Jaguemont et al., 2014), Dow Kokam (Dow, 2010) (manufacturer) and Panasonic (Pan, 2012) (manufacturer).

full (continuously), and heating off at -7°C. During the two tests at -7°C over the LA4 drive cycle (also known as FTP-72 or the Urban Dynamometer Driving Schedule), Meyer *et al.* = found that the use of PTC heaters led to a 41% reduction in vehicle range compared to its 20°C range. In comparison, a range reduction of just 15% was found with the heating off at the same temperature, which can be attributed to the effect of low temperatures on an Li-ion battery. The combined effect of heating demand and reduced battery capacity is a 60% reduction in range at -20°C (Reyes et al., 2016).

Heat pumps are now becoming more dominant as a solution to vehicle heating, with examples found in vehicles such as; Jaguar iPace, BMW i3, Nissan Leaf (latest generation), Renault Zoe and others. The main advantage of a heat pump is that it doesn't solely generate heat, but can extract and upgrade heat; making it useful for cabin heating. This can be done at more than 100% efficiency. Examples of research in this area include Leighton *et al.* (Leighton, 2015), who demonstrated a lab bench system capable of extracting heat from ambient; upgrading it with heat from power electronics and a PTC heater, then heating the cabin and battery. In this example the power electronics, PTC heater, cabin and battery were in series in a coolant loop with the heat pump's condenser in the order stated. In 2014 Ahn *et al.* proposed a dual source heat pump which could harvest waste heat from the motor as well as ambient; this can be distinguished from Leighton's work as the motor would be located on the evaporator loop, rather than the condenser loop. Using simulation, Ahn showed that the addition of waste heat from the motor increased the maximum coefficient of performance (COP) from 3 to 3.4, but also allowed the heat pump to work more effectively at lower temperatures, where heat extraction from ambient is more difficult. In 2017 Jeffs *et al.* proposed the use of a thermal battery to further assist energy saving at low temperature. They showed that an optimally sized thermal battery would be able to effectively replace the PTC heater, which was previously needed to aid heat pump warm up. Here an average energy saving of 25.3% was made over a temperature range of -20°C to 14°C using a cruising drive cycle, while not compromising cabin warm up times (Jeffs et al., 2017). This selection of work shows the range of use cases which a heat pump may be subjected to, a point further demonstrated by Jeffs *et al.* in 2018, where 32 operational modes where identified and compared for a multiple source heat pump on an electric vehicle (Jeffs et al., 2018).

1.3 Goals for Model capability

From section 1.2 it is clear that a consensus on a thermal management architecture from low temperature perspective has not been reached. With multiple components on the vehicle wasting heat, as well as the added complexity of balancing cabin and battery heating, and the added complexity of optimally operating a heat pump in a dynamic environment, a model which allows users to flexibly reconfigure the thermal architecture of the vehicle could help guide the thermal architecture on vehicles in the future.

The model developed in this work has the objective of providing the following features.

1. The ability to dynamically connect and disconnect components from the thermal management system.

2. The ability to arbitrarily request heat flows between components (e.g. request 5kW for cabin heating), while being physically limited by sensibly sized heat exchangers.

3. Contain a control system for the heat pump which self regulates compressor speed regardless of vehicle configuration.

4. Run quickly enough to be useful for performing parameter sweeps and optimisations in suitable time frames. (PC configuration: Laptop, using i7-6600U @ 2.6GHz with 16GB RAM. Dymola version 2019 using Visual Studio 2015/Visual C++ 2015 Express Edition (14.0).)

2 Method

Here the details of the models are discussed with justifications of choices made during the development process. The top level of the vehicle is shown in Figure 2. In this figure the sub-models; heatDemandCommander, heat pump control unit HPCU, battery, Heat Pump, and Cabin model can be seen. These sub-models will be discussed in more detail in the following subsections, starting with the battery. Following the description of the model, two test

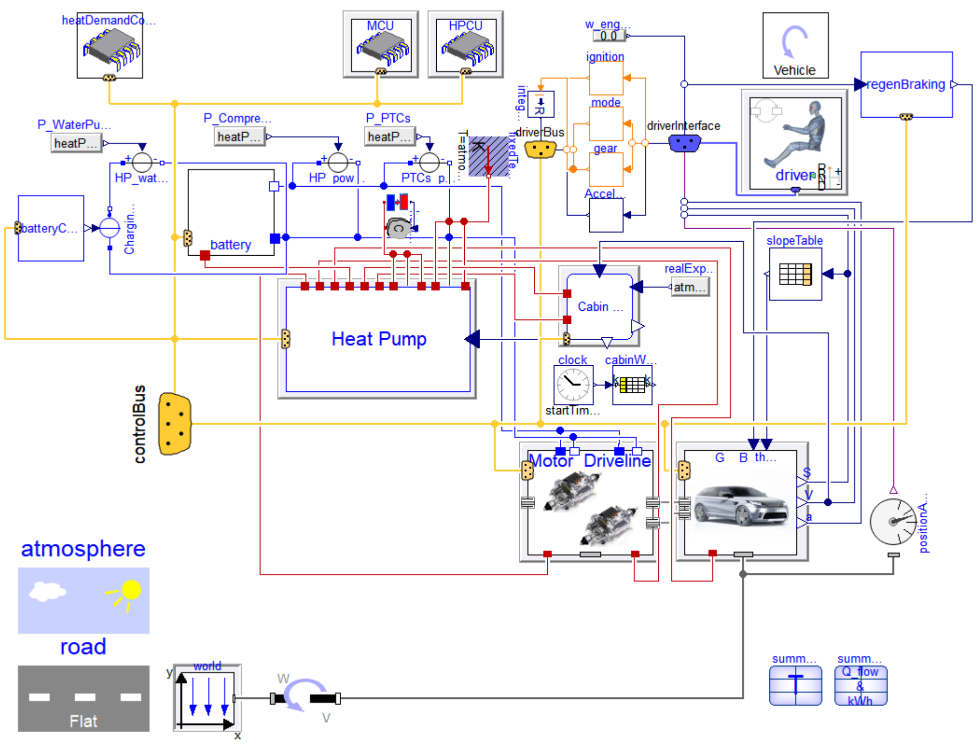


Figure 2. The top level of the model is shown.

cases are presented which demonstrate the model's versatility.

The model has been developed using Dymola based on the Modelica language. This work relies on the following libraries and providers; "Claytex VeSyMA-Powertrain" and "TLK-Thermal Systems library".

2.1 Battery

The battery is modelled both electrically and thermally. The electric side uses a first order RC network equivalent circuit model (ECM), as seen in Figure 3. The components in this model are parameterised using look up tables which are a function of component temperature, as measured using the thermal model, and the cell's state of charge (SOC). The resistor and open circuit voltage (OCV) values are scaled by the number of cells in series to produce an RC circuit which represents one string of the pack. This data was generated specifically for Xalt 40Ah cells by Yashraj Tripathy in (Tripathy et al., 2018). This type of battery model is typically seen in literature (Jaguemont et al., 2016; Ruan et al., 2014) when modelling pack size batteries for vehicle application. In this application the vehicle was configured in a 2p108s (2 parallel strings of 108 cells in series) arrangement giving a pack size of approximately 30kWh; although altering the pack size for different operations is possible. This can be achieved by reducing the number of cells in series by adjusting nS_cell, seen in the top left of Figure 3, this will also automatically adjust the sizing of thermal mass in the thermal model. Alternatively, individual strings may be deleted or duplicated, changing the number of cells in series, this would require the re-parametrisation of the thermal and state of charge models accordingly.

Since the parameters of the RC network are dependent on temperature, a thermal model was created to estimate the bulk temperature of the pack. Figure 3 shows that the ohmic losses from the resistors in the circuit are calculated and exported from the sub-model as Total waste power. The sum of these losses from each string in parallel is used as the heat generation through losses in the thermal model. The thermal model can be seen in Figure 4. In this model the heatCapacitor seen in the centre represents the bulk of the battery and its temperature is used as the battery temperature. It has three modes of heat exchange; input heat from waste heat generation in the cells, interaction with the thermal management system, and thermal losses to ambient. The latter heat exchange is modelled using the flat plate parallel flow equation (Bergman et al., 2011), given in Equation 1,

$$\bar{h} = (0.037 Re^{4/5} - 871) Pr^{1/3} \frac{k}{L} \qquad (1)$$

where k, Re and Pr are the thermal conductivity, Reynolds number and Prandtl number of the convection fluid. The Reynolds number and Prandtl number are defined in Equations 2 and 3. L is the length of the surface over which the fluid is flowing, here $4m$ is used as an approximation of the length of the underside of the vehicle.

$$Re = \frac{\rho v L}{\mu} \qquad (2)$$

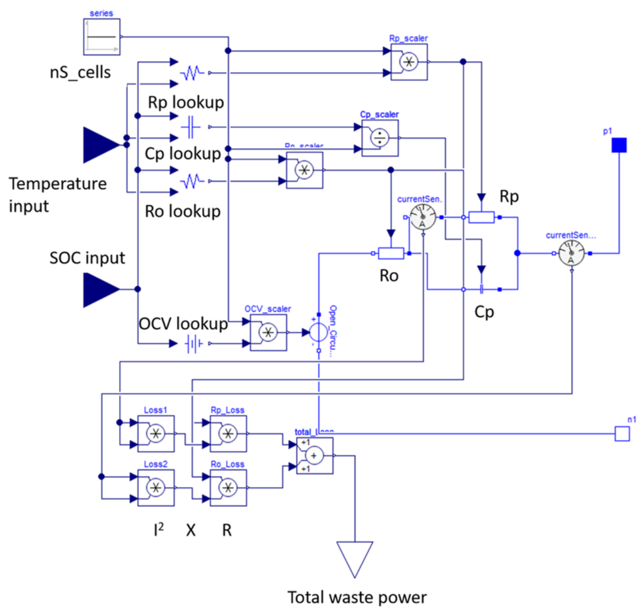


Figure 3. Electric first order RC model for the battery, scaled by number of cells in series using parameter nS_cells, seen in the top right.

$$Pr = \frac{C_p \mu}{k} \tag{3}$$

In Equations 2 and 3 ρ, v, μ and C_p are density, velocity, dynamic viscosity and specific heat capacity of convective fluid, in this case air. The final heat flow to ambient is then given by

$$Q_{ambient} = \bar{G}(T_{plate} - T_{ambient}) \tag{4}$$

or

$$Q_{ambient} = \bar{h} \times W \times D \times L(T_{plate} - T_{ambient}). \tag{5}$$

It should be noted from literature that the capacity of a battery is dependent of its temperature; in research concerning the operation of electric vehicles in low temperatures this should be accounted for. The state of charge model is used to estimate the state of charge of the battery through the drive cycle as a function of temperature. To achieve this, the state of charge model has an additional lookup table which contains information about the percentage of capacity available as a function of temperature. This is then used to scale the Coulomb counting equation by the factor C_{eff} seen in Equation 6. Other examples of this adjustment to Coulomb counting can be seen in (Tripathy et al., 2018) and (Barai et al., 2016).

$$SOC(t) = SOC_{init} - \frac{1}{C_{eff}} \int_0^t I(t)dt \tag{6}$$

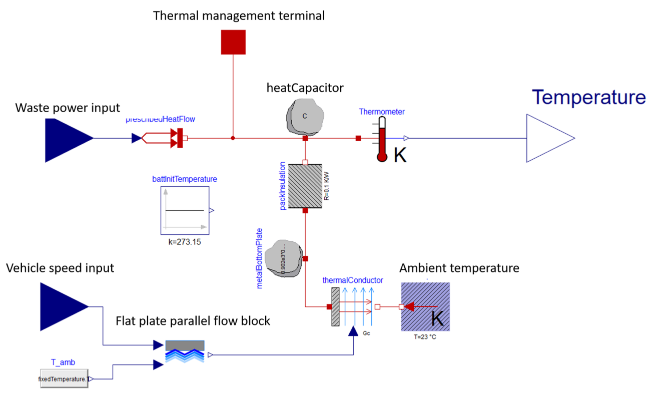


Figure 4. Thermal model of the battery with heat exchanges between ambient through a resistor to an exterior metal plate, and to the thermal management system through heat port seen at the top of the figure.

2.2 Cabin

The cabin has a target temperature of 22°C which can be used to assess the thermal comfort achieved when using the heat pump. The cabin model can be seen in Figure 5. An infinite air source with ambient temperature is taken into the cabin and heated in the eAC component, seen in Figure 5. The heated air is then pumped into the cabin, where a single air volume and heat capacitance are used to measure the cabin temperature. This heat capacitor has four modes of loosing heat; convection to ambient through panels, thermal exchange with soft furnishings, thermal exchange with hard furnishings, and cabin air exhaust. The convection to ambient through the exterior surfaces is modelled using a variable thermal conductance with dependency on vehicle speed. The model uses two heat capacitances for air to hard furnishings (such as dashboard panels, glass etc.) and soft furnishings (such as the seats, carpet etc.), with thermal resistances between the air and these components. Finally there are two volumes representing the air in the cabin; one large and one small. The larger air volume represents the majority of the cabin where the target temperature is imposed. The smaller air volume is used to harvest cabin exhaust waste heat, here up to 30% of the heat can be extracted and used in the heat pump. This amount reflects the claims made by BMW in (Suck and Spengler, 2014).

2.3 Heat Pump

The heat pump has the flexibility to dynamically connect to, and disconnect from, components around the vehicle. There are two places in the heat pump where these connections are controlled. In Figure 6 there is a row of thermal switches at the top of the model, this is the first point of control regarding component connections. Inside these switches there is a thermal conductor which is either set to 0 to thermally isolate the component from the heat pump, or controlled by a PID controller to achieve the desired heat exchange. The desired heat flow is set in the heatDe-

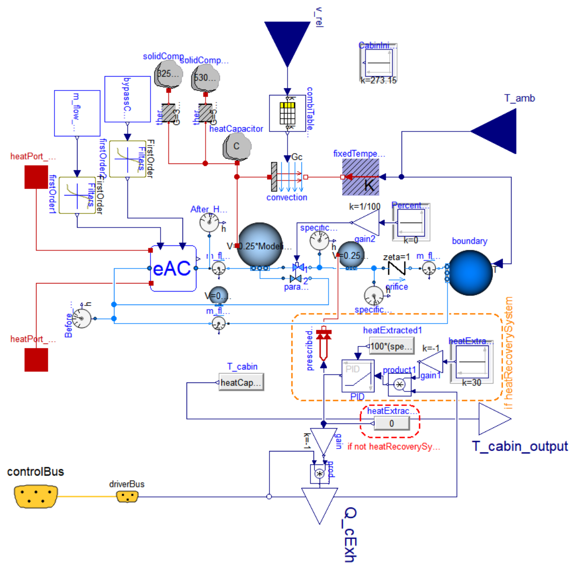


Figure 5. Cabin model

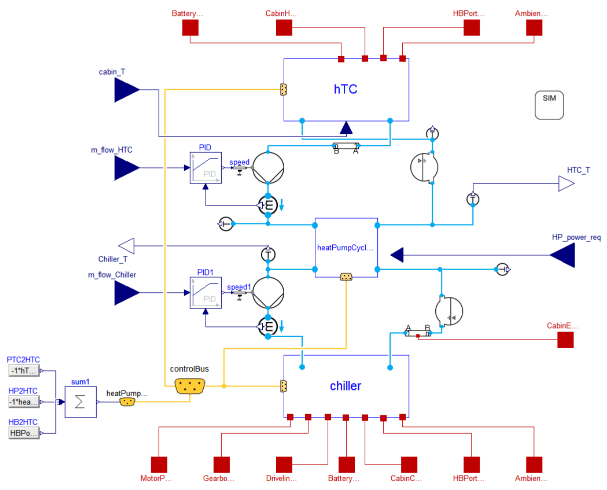


Figure 7. The middle level of the model contains the coolant circuits and the refrigeration loop and is used to control coolant flow between them.

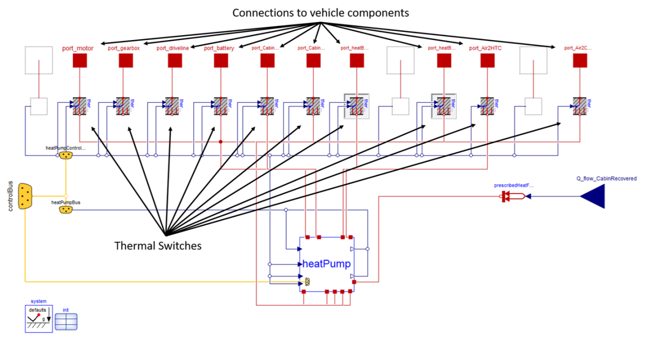


Figure 6. The uppermost level of the heat pump where thermal connections across the vehicle are made and controlled. In the centre is the middle layer of the heat pump which contains the physical models of the coolant and refrigeration circuits.

mandCommander and will be discussed further in section 2.5. This layer acts to control the thermal connection between the heat exchangers in the middle level of the heat pump model and the vehicle components. This is important as it allows for arbitrary heat demands to be requested, which might then be used to guide heat exchanger sizing or coolant control to maintain an optimal temperature in a component. The second point of control for component connections is in the coolant circuits labelled high temperature circuit (HTC) and chiller circuit, found in the middle level of the heat pump, seen in Figure 7.

The middle level of the heat pump model is used to house, pump coolant between, and direct heat flows to, the 3 main models of the heat pump. These are the HTC, chiller circuit and refrigeration loop. Inputs to this level include coolant mass flow rate (which is then controlled by a pump and PID), compressor power demand which is passed onto the refrigerant model, and cabin temperature which is used to shut off the PTC heater in the HTC. The coolant circuits themselves contain heat exchangers which have been sized using examples and estimates taken from components found on existing vehicles. This level also sees the input of heat recuperated from cabin exhaust, which is used to increase the temperature of the chiller loop.

The coolant circuits, HTC and chiller are shown in Figures 8 and 9 respectively. They have operating temperatures of 90°C and -10°C respectively. In Figure 8 the thermal battery and PTC heater are in series before the coolant reaches the cabin, battery and ambient which are in parallel. It can also be seen that all components with the exception of the PTC heater have a bypass option; this is the second control point for component connections which can be used to thermally isolate a component from the system. This arrangement was chosen so that thermal battery could be used to increase the coolant temperature from the condenser output and under certain conditions negate the need for the PTC heater, which is set to turn off if the coolant temperature exceeds 85°C. The hot coolant is then split between the battery and the cabin, giving the user flexibility in selecting how much heat to send to each component; either by using the heatDemandCommander, setting different bypass amounts for each component, or adjusting the heat exchanger sizing. Here it should be mentioned that the PTC heater is controlled by a PID with the objective of getting the cabin to its set point, unless shut off by excessive coolant temperature.

The chiller circuit is arranged in a similar way to the HTC, as seen in Figure 9, with the components capable of contributing heat to the system set in parallel. These heat exchangers allow heat to be extracted from; the motor and inverter (as one unit), the gearbox and driveline (which will be treated as one component and referred to as transmission), the battery, the thermal battery, and ambient. The chiller circuit also uses bypasses to thermally isolate components from the heat pump. The bypass valves are set to values 0.99 for open and 10^{-8} for closed, to

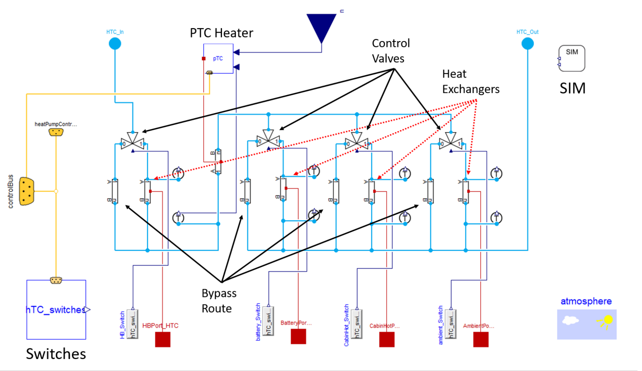


Figure 8. The coolant loop used for high temperature components, i.e. cabin and battery heating, PTC input, heat battery input, and rejecting heat to ambient for air conditioning operation.

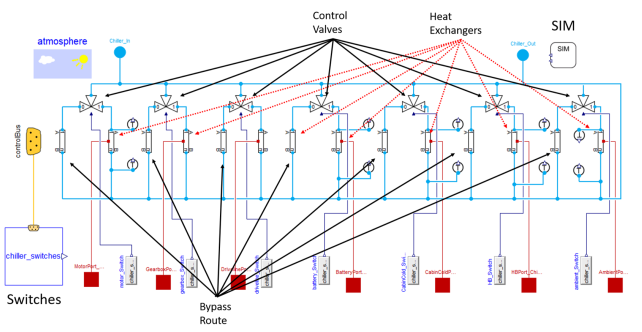


Figure 9. The chiller circuit is used for thermal exchange and extraction, i.e. extracting from the motor and inverter, the gearbox and driveline, ambient, heat battery. There is also the capability to cool the cabin and electric battery if needed.

prevent errors and problems encountered with zero flow. These valves are controlled by the switch blocks in the bottom left of Figures 8 and 9, which is connected to the controlBus and is controlled in the HPCU.

The bottom level of the heat pump contains the physical model of the refrigerant circuit. In this example R134a is used, but the refrigerant circuit may be quickly reconfigured by selecting a different refrigerant in the SIM block and changing pressure settings accordingly. The configuration of the heat circuit is typical of what may be found in literature (Leighton, 2015; Ahn et al., 2014). Since the heat exchanger has liquid coolant heat exchangers on both the condenser and evaporator sides, some care has been taken to create a control system which does not exceed the physical limits of the coolant and refrigerant set by the library. The coolant is prevented from going beyond its temperature limits using the HPCU which controls the compressor demand seen on the right of Figure 10. The refrigerant is protected from exceeding its pressure limit by the PID labelled *PID_pressureControl* seen in Figure 10, which is set to limit the pressure to $30 bar$; chosen corresponding to R134a's pressure-enthalpy diagram. Given two power demands existing at this level, the demand ac-

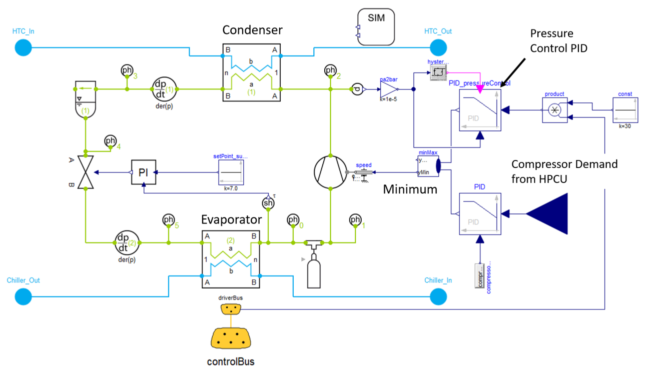


Figure 10. The refrigeration cycle is physically modelled. At this level the speed of the compressor is controlled either by the demand from the heat pump controller, or a PID which stops the system pressure getting too high.

cording to the pressure controller and the demand input from the HPCU, the minimum of these values is used to control the compressor, ensuring that neither refrigerant pressure nor coolant temperature exceed their physical limits.

2.4 Heat Pump Control Unit (HPCU)

The heat pump control unit has three purposes; firstly to set the desired coolant flow for the HTC and chiller circuits, secondly to control which components are connected to, or thermally isolated from, the heat pump, and finally to set the desired power output of the heat pump compressor. The first of these roles is done by putting a constant mass flow demand onto the control bus to be used at the heat pump middle level. For the motor and inverter (single unit), the transmission and the cabin, the thermal switching can be thought of as binary. To ensure a heat flow is created when the component is thermally active, the connection is only allowed to be made if there is a sufficient temperature difference between the component and the coolant loop. For the electric battery, thermal battery and ambient, extra rules exist for logical operation of these sources. Firstly, the electric battery is connected to the HTC (being heated) when its temperature is below 20°C; above this temperature it is thermally isolated from the heat pump. If the electric battery's temperature should rise above 30°C it is connected to the chiller (being cooled). Both these temperatures were chosen for sensible electric battery operation. The heat battery is first connected to the HTC, helping to rapidly increase the HTC temperature during the warm up phase. When it can no longer heat the HTC it is connected to the chiller, where it is fully discharged, then thermally isolated. Finally, ambient is only used as a heat source when its temperature is above -10°C, which, in a normal vehicle, would prevent frost from building up on the ambient heat exchanger, reducing its effectiveness.

The compressor controller, shown in Figure 11, has the purpose of getting the cabin and battery to temperature

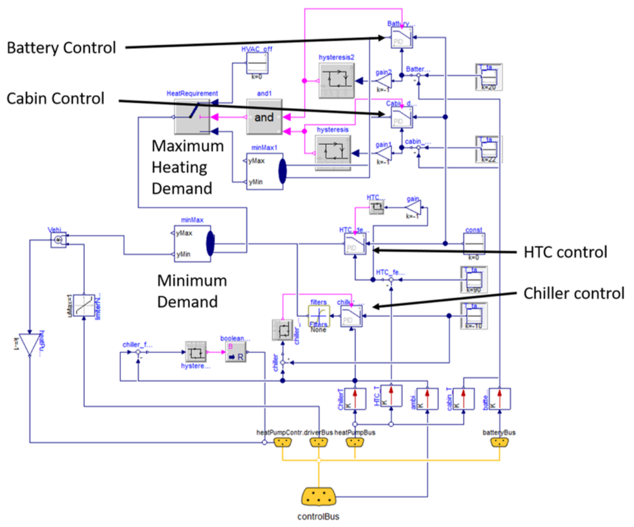


Figure 11. The compressor controller

while ensuring that the HTC and chiller don't exceed their temperature limits. PID controllers are used to create a compressor power demand according to the battery and cabin current and target temperatures, the greatest of these demands is taken to ensure there is enough heat to meet these requirements. Additionally, PID controllers are used to set a power demand needed to bring the chiller and HTC loops to their target temperatures. The minimum power request (from either the cabin and battery, the chiller and the HTC controls) is then passed to the compressor; the minimum is used so that, if a component has reached its set point, it is not pushed beyond that target by the requirements of another component. This logic prevents the battery and cabin from overheating and reduces the chance of the model failing due to the coolant breaching its temperature limits.

2.5 heatDemandCommander

This component is used to set requested heat flows for thermally active components around the vehicle. For this work constant heat flows were requested. The heat flow request is then put on the control bus to be used as the set point for PID controllers in the switches found in the heat pump, described in section 2.3.

2.6 Test cases

Here 2 unique test cases are proposed to demonstrate the flexibility of this work for testing a variety of scenarios. These test cases are:

1. Can the electric battery be used as a heat source when its temperature is above 0°C (chosen to maintain regenerative braking). To explore this case 5 scenarios will be tested and compared; 1. heating the battery (as described in section 2.4), 2. cooling the battery, 3. disconnecting the battery, 4. battery disconnected and PTC off, and 5. battery cooled with PTC off.
2. Is the transmission useful as a thermal source for the heat pump? This is tested by using ambient and transmission as heat sources, then comparing the total vehicle energy consumption and cabin temperature profile. As with the previous case, these 2 scenarios will be retested with the PTC heater off. To make a complete assessment of the transmission as a thermal contributor it will be tested in isolation, i.e. the motor will be disconnected from the heat pump leaving the transmission and ambient as the only contributors.

Both of these test cases are demonstrated at 0°C ambient temperature and using the WLTP drive cycle. While the results of these two cases will be shown in detail, the model has been used in many more scenarios.

3 Results

Here the results of the two cases will be shown and discussed.

3.1 Case 1

In this demonstration, the comparison is made between heating the battery and cooling the battery by adjusting its target temperature in the switches controller. The battery will also be disconnected from the heat pump to provide a baseline. Figure 12 shows the battery temperature through the drive cycle. Here it can be seen that the scenarios have performed as expected. When the battery is heated it quickly reaches its target temperature (20°C), at which point it is isolated from the heat pump. During this heating phase 5kW of heat flow is requested in the heatDemandCommander which is then sustained by the models controllers. Despite being disconnected from the heat pump at target temperature, the battery's temperature continues to rise due to internal resistance and self heating. Should the temperature rise above 30°C then the battery would connect to the chiller. This temperature is chosen somewhat arbitrarily, although some cooling is required to reduce ageing and mitigate against thermal runaway and other safety concerns associated with high temperature cell operation. In the second scenario it can be seen that the cooling keeps the battery temperature much lower throughout the cycle. The battery temperature rises quickly at the end of the cycle, during the high speed section; this reflects the higher losses the battery will experience through maintained lower temperatures. Finally, if the battery is not thermally managed as in the third scenario of case 1, the battery temperature is allowed to rise unaided until it is naturally limited by reduced loss and thermal loss to ambient.

Figure 13 is used to show the impact each scenario has on the cabin temperature. When the battery is heated there is less heating capacity for the cabin and hence the temperature, and therefore comfort, is reduced. When the battery is cooled there is extra thermal capacity and so the cabin is heated faster and thermal comfort is improved. However,

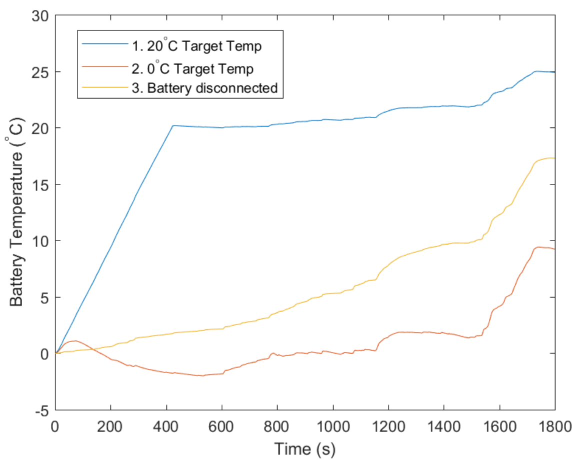


Figure 12. The battery temperature through the WLTP cycle shown for the first 3 scenarios proposed in case 1.

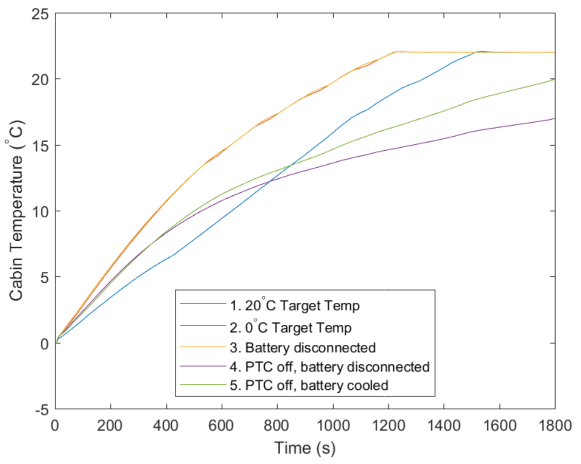


Figure 13. The cabin temperature is shown for the 3 scenarios proposed in case 1.

the extra heat that is extracted from the battery does not appear to make a significant difference to the cabin temperature compared to disconnecting the battery from the heat pump. While the extra heat from the battery should provide extra heat for the cabin, the heat pump is already saturated with heat from ambient and the motor, hence the extra heat only serves to increase the coolant temperature on the chiller. Furthermore with the additional heat from the PTC heater (on in all cases) the HTC quickly reaches temperature.

Scenarios 4 and 5 have been used to demonstrate the benefit that battery cooling can provide to cabin heating when the heat pump is not saturated with heat. Here it can be seen that the cabin temperature suffers in both scenarios where the PTC heater is off. However, the additional heat extracted from the battery now allows the cabin to heat up faster and reach a higher final temperature, compared to no heat extraction and no PTC heater.

Table 1 shows the energy consumption and the final SOC corresponding to the 5 scenarios. Since the rate at which SOC is used is dependant on temperature, the final SOC is not directly proportional to energy consumption, but is also linked to temperature profile. Here it should be noted that scenario 1 (where the battery is heated) has a final SOC higher than scenario 2 (unheated). Operating at a higher average temperature through the cycle reduces the losses through internal resistance, increasing the terminal voltage and reducing the current required to produce the same power, hence the difference in final SOC.

With regards to point 4 of section 1.3, it took 32 minutes and 51 seconds to simulate all the scenarios required for case 1. This gives and average simulation time of 6 minutes and 34.2 seconds. Since WLTP is a 30 minute drive cycle this equates approximately 4.6 times faster than real time, meaning the simulation is adequately fast.

Table 1. The total electrical energy consumption and final SOC for the 5 scenarios proposed in case 1.

Scenario	Energy Consumed	Final SOC
1	8.08 kWh	81.1 %
2	8.24 kWh	78.5 %
3	7.68 kWh	80.8 %
4	6.45 kWh	83.9 %
5	7.68 kWh	79.6 %

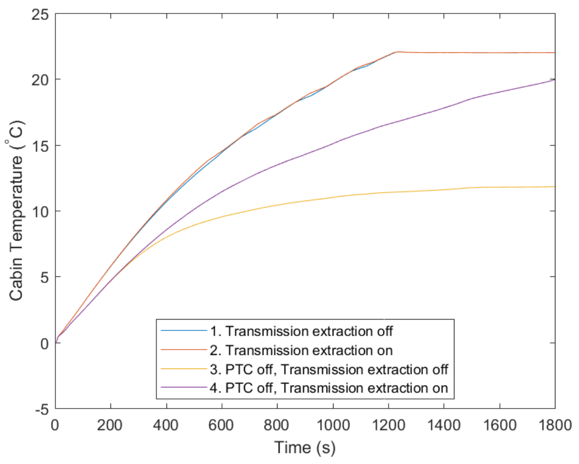


Figure 14. The cabin temperature is shown for the 4 scenarios proposed in case 2.

3.2 Case 2

Here the benefit of including the transmission as a thermal contributor to the heat pump is evaluated. This is done by comparing 4 scenarios, firstly a baseline case with ambient the thermal contributor, and secondly with the transmission as an additional contributor, then repeated with the PTC heater off.

Like in case 1 the heat pump is saturated before the transmission is introduced as a contributor; hence the heat extracted does not make a noticeable difference to the

Table 2. Total electric energy consumption for the 4 scenarios proposed in case 2

Scenario	Energy Consumed
1	7.82 kWh
2	7.68 kWh
3	6.05 kWh
4	6.72 kWh

cabin comfort, as can be seen when comparing scenarios 1 and 2 in Figure 14. As with the battery, when the PTC heater is turned off the extra heat provided to the cabin makes more of a difference, as in scenarios 3 and 4. The compromise of this extra cabin comfort is the cost of extraction and the extra load that is put on the motors due to the transmission being kept at a lower, less efficient temperature. The energy consumption for each scenario is given in Table 2.

In Table 2 it can be seen that when the PTC heater is in use (scenarios 1 & 2) the extra heat from the transmission saves energy. In other words the energy saved from reaching HTC target temperature slightly earlier, which causes the PTC heater to shut off, exceeds the costs of operating the transmission at a lower efficiency and extracting the heat. Although, since the heat pump is operating at capacity before the transmission is introduced, the cost of extraction is negligible. Considering this, in these circumstances the transmission is viable as a thermal contributor, on the condition that the engineering or implementation costs do not outweigh the potential benefits. When the PTC heater is not used, the extra costs of using the transmission do not return an energy saving and so increase total consumption. Given the extra thermal comfort provided, seen in Figure 14, this is probably a worthwhile compromise.

Again for completion, the simulation time for case 2 was 28 minutes and 40 seconds, or 7 minutes and 10 seconds per scenario, or approximately 4.2 times faster than real time. This further evidences that the model runs reliably fast, completing simulations in an adequately short amount of time.

4 Discussion

The results presented above show the variability and potential power of this model for evaluating different thermal management strategies in cold climates. One of the key questions in this area concerns the thermal management of batteries; in case 1 some scenarios are proposed and evaluated. Each scenario evaluated met the expected outcome; for example heating the battery uses more energy and reduces cabin comfort compared to disconnecting the battery from the heat pump, seen by comparing scenarios 1 and 3. It was also shown that the battery may be used as a heat source when cabin heating is not saturated, as seen in scenarios 2, 4 and 5. It is also interesting to note that the reduced battery efficiency and lower effective SOC of extracting heat from the battery means that it is not worth doing, as energy consumption and final SOC of scenario 5 are worse than scenario 3. This difference is marginal and under other circumstances (temperature, battery target temperature, battery size, ambient temperature etc.) the same may not be true. This platform gives the user the ability to easily and quickly explore these spaces.

In case 2 it was shown that the transmission is a viable contributor to the heat pump. This was demonstrated by the energy saving achieved when the PTC heater is in use, and the additional thermal comfort when it was not. Considering this further investigation into its use with a more complex system should be undertaken. While it is seen to provide a benefit on its own, in a system where cabin heat is saturated, its use may not be as valuable as seen with scenarios 1 and 2. The system proposed has the ability to further explore these scenarios and make a more complete recommendation for the use of the transmission as a thermal contributor to the heat pump.

Finally a review of the objectives that were set in section 1.3. Firstly, goal 1. is shown in the case 1, scenario 1 where the battery disconnects itself from the heat pump when it reaches its target temperature. This is an example of passive dynamic connection and disconnection, however a schedule could be implemented to directly control the connection timings. Secondly, in case 1 the model successfully sustained the requested heat flow of 5kW to the battery while it was being heated. Thirdly, in sections 2.4 and 2.3 a control system for the compressor was described which self regulates according to coolant temperatures, cabin and battery target temperatures and refrigerant pressure. This control system ensures the cabin and battery reach their target temperatures without model failing due to breach of physical limits. Fourthly and finally, in the examples shown the case 1 took 32 minutes and 51 seconds to complete, while case 2 took 28 minutes and 40 seconds to complete which are perfectly usable time frames when evaluating thermal management strategies.

5 Conclusion

The model has been demonstrated in a range of cases and scenarios. Its versatility in answering many thermal management problems has been shown and the objectives set to prove this have been met. The results from the tests proposed and evaluated are justifiable and make sense when compared to what is known about electric vehicles operating in low temperatures and the nature of heat pumps. Hence this work shows a valuable, versatile tool in exploring thermal management of complex heat pump systems on electric vehicles in low temperature climates.

Acknowledgement

This project was possible thanks to funding provided by EPSRC, funding and support from Jaguar Land Rover, and software and modelling support from Claytex Services Ltd

References

Jae Hwan Ahn, Hoon Kang, Ho Seong Lee, Hae Won Jung, Changhyun Baek, and Yongchan Kim. Heating performance characteristics of a dual source heat pump using air and waste heat in electric vehicles. *Applied Energy*, 119:1–9, 2014.

Megan Allen. `http://www.fleetcarma.com/electric-car-range-in-bitter-cold/`, 2013. Accessed: 29/11/2018.

Anup Barai, Kotub Uddin, WD Widanalage, Andrew McGordon, and Paul Jennings. The effect of average cycling current on total energy of lithium-ion batteries for electric vehicles. *Journal of Power Sources*, 303:81–85, 2016.

Theodore L Bergman, Frank P Incropera, David P DeWitt, and Adrienne S Lavine. *Fundamentals of heat and mass transfer*. John Wiley & Sons, 2011.

Lionel Broglia, Gabriel Autefage, and Matthieu Ponchant. Impact of passenger thermal comfort and electric devices temperature on range: a system simulation approach. *World Electric Vehicle Journal*, 5(4):1082–1089, 2012.

Kevin Bullis. `https://www.technologyreview.com/s/522496/electric-vehicles-out-in-the-cold/`, 2013. Accessed: 29/11/2018.

Superior Lithium Polymer Cell: Technical Data Sheet. Dow Kokam, 9 2010.

J Jaguemont, L Boulon, Y Dubé, and D Poudrier. Low temperature discharge cycle tests for a lithium ion cell. In *2014 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–6. IEEE, 2014.

Joris Jaguemont, Loïc Boulon, and Yves Dubé. Characterization and modeling of a hybrid-electric-vehicle lithium-ion battery pack at low temperatures. *IEEE Transactions on Vehicular Technology*, 65(1):1–14, 2016.

J. Jeffs, A. McGordon, W. D. Widanage, S. Robinson, and A. Picarelli. Use of a thermal battery with a heat pump for low temperature electric vehicle operation. In *2017 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–5, Dec 2017. doi:10.1109/VPPC.2017.8330932.

James Jeffs, Andrew McGordon, Alessandro Picarelli, Simon Robinson, Yashraj Tripathy, and Widanalage Widanage. Complex heat pump operational mode identification and comparison for use in electric vehicles. *Energies*, 11(8):2000, 2018.

Yan Ji, Yancheng Zhang, and Chao-Yang Wang. Li-ion cell operation at low temperatures. *Journal of The Electrochemical Society*, 160(4):A636–A649, 2013.

Daniel Leighton. Combined fluid loop thermal management for electric drive vehicle range improvement. *SAE International Journal of Passenger Cars-Mechanical Systems*, 8(2015-01-1709), 2015.

Juuso Lindgren and Peter D Lund. Effect of extreme temperatures on battery charging and performance of electric vehicles. *Journal of Power Sources*, 328:37–45, 2016.

Norm Meyer, Ian Whittal, Martha Christenson, and Aaron Loiselle-Lapointe. The impact of the driving cycle and climate on electrical consumption and range of fully electric passengers vehicles. In *Proceedings of EVS*, volume 26, 2012.

Ganesan Nagasubramanian. Electrical characteristics of 18650 li-ion cells at low temperatures. *Journal of applied electrochemistry*, 31(1):99–104, 2001.

Lithium Ion NCR18650. Panosonic, 2012.

J. R. M. Delos Reyes, R. V. Parsons, and R. Hoemsen. Winter happens: The effect of ambient temperature on the travel range of electric vehicles. *IEEE Transactions on Vehicular Technology*, 65(6):4016–4022, June 2016. ISSN 0018-9545. doi:10.1109/TVT.2016.2544178.

Haijun Ruan, Jiuchun Jiang, Bingxiang Sun, Ningning Wu, Wei Shi, and Yanru Zhang. Stepwise segmented charging technique for lithium-ion battery to induce thermal management by low-temperature internal heating. In *Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), 2014 IEEE Conference and Expo*, pages 1–6. IEEE, 2014.

XH Rui, Y Jin, XY Feng, LC Zhang, and CH Chen. A comparative study on the low-temperature performance of lifepo 4/c and li 3 v 2 (po 4) 3/c cathodes for lithium-ion batteries. *Journal of Power Sources*, 196(4):2109–2114, 2011.

Samveg Saxena, Caroline Le Floch, Jason MacDonald, and Scott Moura. Quantifying ev battery end-of-life through analysis of travel needs with vehicle powertrain models. *Journal of Power Sources*, 282:265–276, 2015.

Gerrit Suck and Carsten Spengler. Solutions for the thermal management of electrically driven vehicles. *ATZ worldwide*, 116(7-8):4–9, 2014.

Yashraj Tripathy, Andrew McGordon, and Chee Low. A new consideration for validating battery performance at low ambient temperatures. *Energies*, 11(9):2439, 2018.

SS Zhang, K Xu, and TR Jow. The low temperature performance of li-ion batteries. *Journal of Power Sources*, 115(1):137–140, 2003.

Fangdan Zheng, Jiuchun Jiang, Bingxiang Sun, Weige Zhang, and Michael Pecht. Temperature dependent power capability estimation of lithium-ion batteries for hybrid electric vehicles. *Energy*, 113:64–75, 2016.

Diesel Cooling System Modeling for Electrification Potential

John Batteh[1] Ashok Kumar Ravi[2] Dale Pickelman[3]

[1]Modelon Inc., USA, john.batteh@modelon.com
[2]Modelon Engineering Private Limited, India, ashokkumar.ravi@modelon.com
[3]Hanon Systems, USA, dpickelm@hanonsystems.com

Abstract

Electrification of automotive systems presents significant opportunities for improvements in cooling system efficiency and performance. This paper describes an effort to develop an analytic platform for Hanon Systems to evaluate the electrification potential for powertrain cooling systems. The paper describes the development of a baseline diesel cooling system model based on the Ford 6.7L Power Stroke diesel. A variant of the system with electric pumps is also modeled. Performance of the baseline conventional and electric pump system are compared on a typical automotive drive cycle to quantify potential benefits of the electric pump system and advanced controls.

Keywords: cooling systems, diesel, electrification

1 Introduction

Electrification is a pervasive trend in the auto industry, from fully electric vehicles to hybrids to electrification of individual subsystems and components. For all powertrain systems, thermal management of the components is a critical requirement for the safe and efficient operation of the system. Furthermore, thermal constraints for electric powertrains can limit performance (Stellato, 2017). Significant energy is required to pump cooling fluid for thermal management. Though varying with engine, cycle/operating conditions, fuel type, system design, etc., 1-3% of fuel energy can be consumed by pumps for cooling and lubrication systems (Thiruvengadam, 2014).

In conventional cooling systems, mechanical pumps are driven by the engine. Connected through a fixed drive ratio to the engine, mechanical pumps operate based on engine speed. Since the flowrate is linked to the engine speed, sizing of the pumps for mechanical systems for maximum cooling load can be problematic. In many vehicles, maximum cooling load results from operating conditions with high engine load and potentially low engine speed and vehicle speed/external air flow. To meet this maximum demand, the requirement drives a large pump size. With the linking of pump speed to engine speed, significant inefficiencies can result from mismatch in pump efficiency to typical operating conditions, overflow in the system, and potentially even over cooling under some conditions.

Replacing a mechanical pump with an electric pump can yield significant benefits. With the pump speed decoupled from the engine speed, the electric pump can be sized more appropriately to meet cooling system demand. Decoupling from the engine operation also means that the electric pump speed can be controlled to provide flowrates on demand to better match the cooling load demand. Advanced control strategies can also lead to additional benefits by optimizing warm up for lubricating fluids like engine oil and transmission oil. Previous analytic studies have demonstrated potential benefits of 1.2% for electric pump systems with advanced controls on a vehicle driven by a 1L turbo gas direct injection engine with additional benefits due to optimized transmission thermal conditions (Zheng, 2018). While careful design of the system is required to fully realize these efficiency improvements, there is clearly motivation to pursue given the potential impact on fuel economy or electric range.

System modeling with Modelica has been widely used for vehicle thermal management simulations (Bouvy, 2012; Krüger, 2012; Batteh, 2014; Stellato, 2017). With a powerful and flexible modeling framework and proven commercial libraries, Modelica provides an ideal platform for architectural studies and controls prototyping for advanced vehicle thermal management. This paper describes an effort to develop an analytic platform for Hanon Systems to evaluate the electrification potential for powertrain cooling systems. The goal of this analytic platform is to allow rapid virtual prototyping of different cooling systems for evaluation of the potential of Hanon hardware and controls solutions for system optimization. To demonstrate this platform, a model of a diesel cooling system based on a Ford Power Stroke diesel 6.7L V8 was developed. A baseline model of the system is developed with Liquid Cooling Library (Modelon AB, 2018) and then modified to include electric pumps from Hanon. The results from the simulations are compared on a typical automotive drive cycle to quantify potential benefits of the electric pump system.

2 Diesel Cooling System Model

This section provides an overview of the diesel cooling system model. The model is based on the Ford Power Stroke (code name "Scorpion") 6.7L V8 diesel designed for the North American light commercial truck market (Deraad, 2010). The Power Stroke diesel is used in Ford F series and Super Duty pickup trucks. The Ford Scorpion diesel was designed with an innovative dual loop cooling system. The following sections provide an overview of the full system model and relevant component modeling details.

2.1 System and Model Overview

The Scorpion diesel system has two cooling loops: a high temperature loop and a low temperature loop. The two circuits are completely unmixed and interact with each other via the two stage EGR cooler and via the radiator air flow with the low temperature radiator in front of the high temperature radiator. The high temperature loop provides cooling and coolant flow for the following components:

- Engine block and head
- Turbocharger
- EGR cooler (1st stage)
- Heater core
- Engine oil cooler

The low temperature loop provides cooling and coolant flow for the following components:

- EGR cooler (2nd stage)
- Transmission oil cooler
- Charge air cooler
- Fuel cooler

Figure 1 shows the entire diesel cooling system model. The model was built without substantial system hierarchy per request when getting started with Modelica-based modeling. The model was built based on publicly available information on the system, including the service manual for the engine and cooling system. Characterization of the model is discussed in Section 2.2. To help understand the coolant flow and system operation, a discussion of relevant sections of the model follows.

The high temperature loop operates at typical coolant operating temperatures around 100°C. Starting from the high temperature pump, the coolant flow splits between the EGR cooler and the engine. The flow through the engine goes to the left and right block and head. Some flow from the left side is sent to the turbocharger. Some flow from the right side of the engine is sent to the engine oil cooler. The flow from the EGR cooler, turbocharger, and resulting flow through the engine join downstream before the thermostat. Some flow goes to the heater core while the remaining flow goes through the thermostat. The thermostat controls the balance of flow between the radiator and the bypass. The heater core flow merges downstream of the radiator and then flows to the degas bottle. The resulting flow is mixed with the bypass and flow from the engine oil cooler and then flows to the pump.

The low temperature loop is a fairly complex hydraulic circuit with multiple flow branches and operating modes based on coolant temperature. The low temperature loop also has operation at higher temperatures and at lower temperatures. The higher temperature portion of the system operates at coolant temperatures greater than 45°C and maintained approximately at 60°C. The lower temperature part of the circuit operates at coolant temperatures greater than 20°C and maintained approximately at 45°C. Starting from the pump, some of the flow splits to the upper section of the radiator and the other part of the flow goes to a high temp thermostat that can direct flow to the EGR cooler second stage and the transmission oil cooler. When the coolant temperature is below 45°C, the high temp thermostat is closed, and some of the flow is sent directly to the transmission oil cooler and EGR cooler. When the coolant temperature is above 45°C, the high temp thermostat starts to open and flow to the EGR cooler and transmission oil cooler is extracted after passing through the radiator upper section. The low temperature thermostat is located in the radiator tank. This thermostat controls the flow between the upper and lower sections of the radiator. When the coolant entering the radiator is less than 20°C, the radiator is bypassed altogether, and the flow is directed to the fuel cooler and charge air cooler. When the coolant reaches 20°C, the low temperature thermostat starts to open and allows coolant flow through the upper and low sections of the radiator before flowing to the charge air cooler and fuel cooler. Recall that the flow to the EGR cooler and transmission oil cooler can be extracted after the radiator upper section. Under different operating conditions, it is possible to bypass the radiator altogether, use only the upper section of the radiator, or use both the upper and lower sections of the radiator. To allow this capability, the two sections of the radiator are modeled as separate heat exchangers. The flow from all coolers joins at the degas bottle upstream of the pump.

The Scorpion diesel system provides a nice benchmark for an analytic platform for virtual prototyping of hardware and control strategies given its overall complexity with multiple loops, multiple coolers, and multiple different temperature levels.

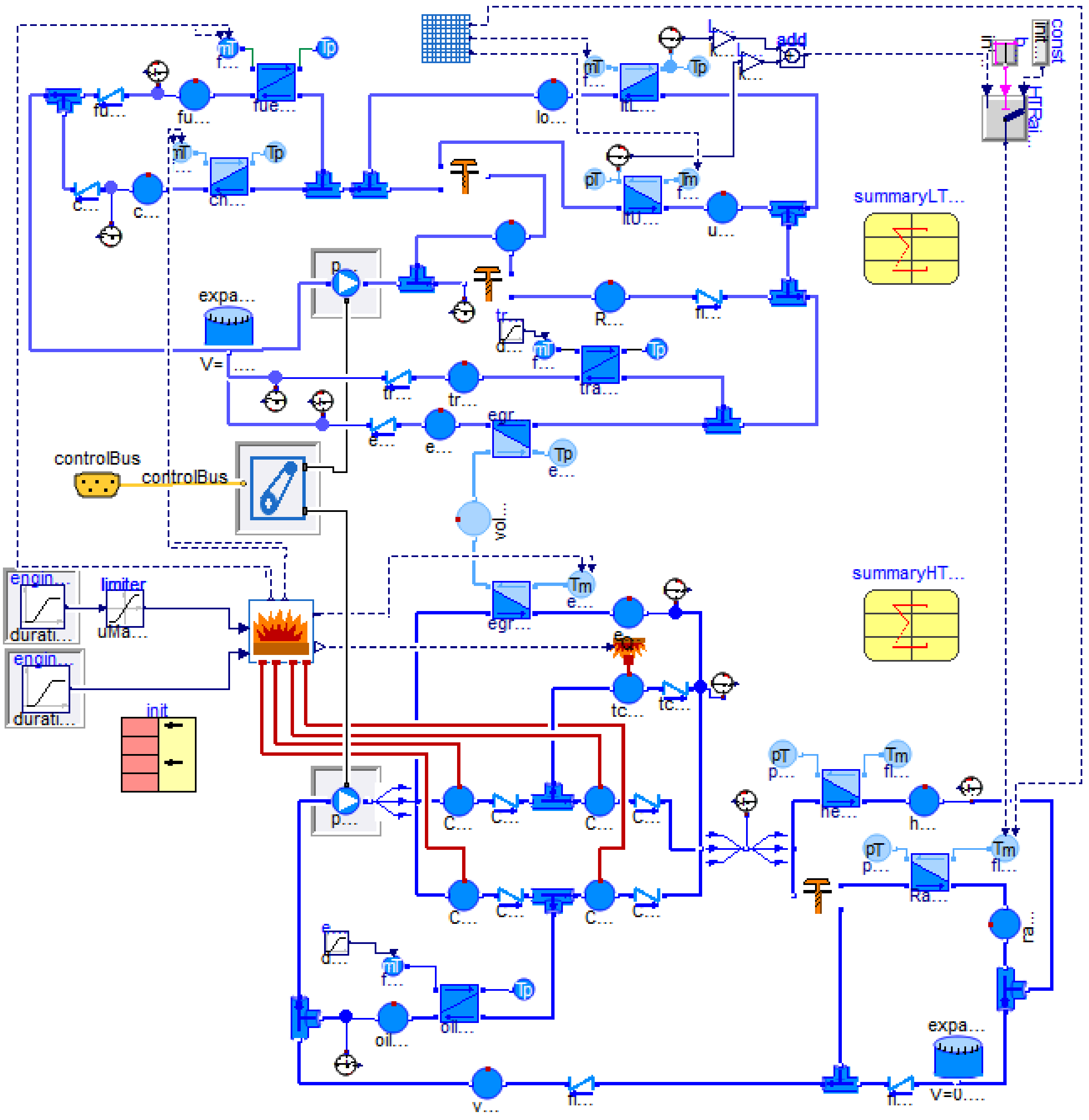


Figure 1. Diesel cooling system model based on the Ford Power Stroke diesel 6.7L V8 with high temperature and low temperature circuits

In addition to the coolant hydraulic circuits, simple hydraulic circuits are modeled for the transmission oil and engine oil. Figure 2 shows this simple circuit for the engine oil cooler and also the implementation of the simple circuit. The inputs to the circuit are the oil flowrate through the cooler and the heat input to the oil and are considered boundary conditions. With this simple circuit, it is possible to simulate dynamic oil temperatures including warmup. While more detailed models can be built if information is available to do so, it is important to have dynamic estimates of oil temperature to ensure that oil temperature limits are respected when considering system variants and also to estimate potential benefits of faster oil temperature warmup for friction reduction.

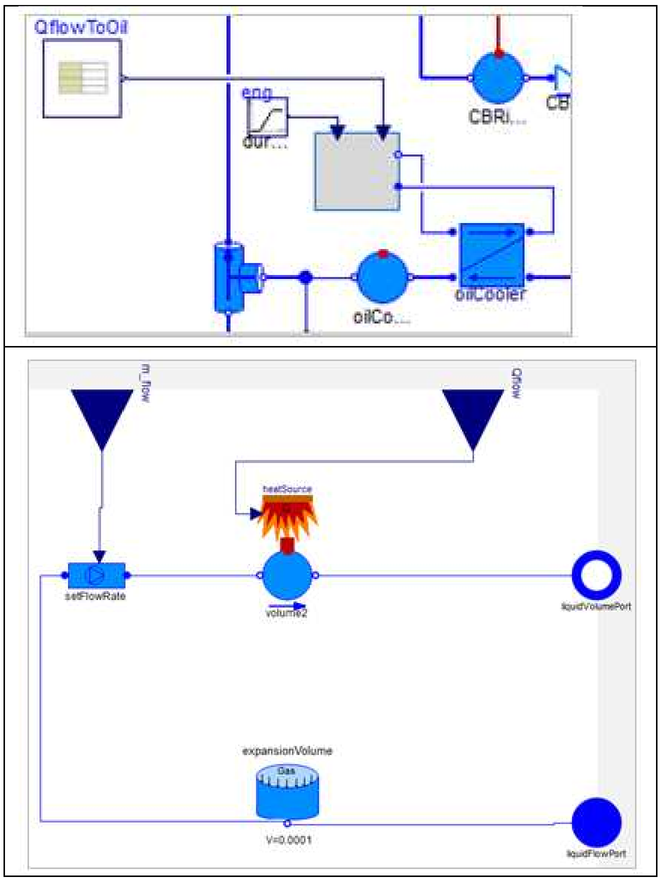


Figure 2. Simple engine oil circuit

Several configurable elements are included in the model to provide the flexibility to switch between the mechanical pump system and the electric pump system. The pump models in the high temperature and low temperature circuits are replaceable models and can be changed individually. A control bus structure is also established. Sensor signals from relevant components are placed onto the control bus. A controller component is then connected to the coolant pumps to specify the pump speed. For the mechanical pump system, the engine speed is passed through to the mechanical pumps offset by a fixed ratio. For the electric pump system, controllers are implemented as discussed in subsequent sections.

The focus of the system model is on the cooling system, but obviously simulation of the cooling system is not possible without the relevant heat inputs and boundary conditions. While a full vehicle simulation can provide some of these inputs, that scope was outside of the focus of the current effort. To support the cooling system modeling effort, a simple map-based engine shown in Figure 3 was developed to estimate heat input from the engine and turbocharger along with operation conditions (flow and temperatures) for the EGR cooler, fuel cooler, and charge air cooler as a function of engine operating conditions. The inputs to the model are the engine brake power and engine speed. The brake specific fuel consumption (BSFC) map was used to calculate fuel flow based on published data (Deraad, 2010). Figure 4 shows the BSFC map as extracted. While highly simplified, this engine model allows simulation of the key inputs from the engine without requiring highly detailed information on the engine and engine operating conditions. Data for this model was input based on some published operating conditions (Deraad, 2010) and then supplemented with nominal information for diesel engines. Predictive capability of this model would obviously be improved with actual engine characterization data, but the basic model does provide a practical computational approach for the engine in lieu of detailed engine mapping data.

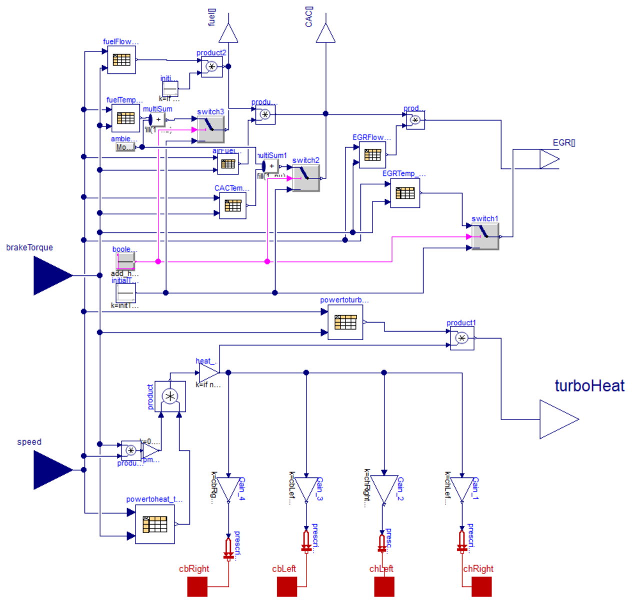


Figure 3. Map based engine model

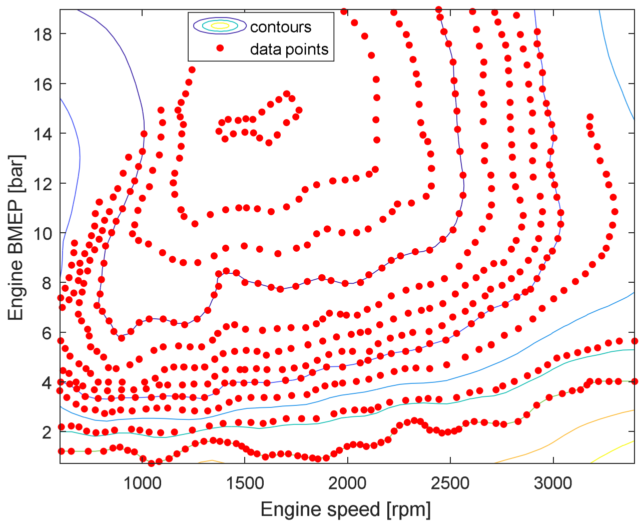


Figure 4. Engine BSFC map as extracted from reference (Deraad, 2010)

In conjunction with the engine model, the boundary conditions for the system are as follows:

- Engine brake power
- Engine speed
- Vehicle speed
- Heat input and flow for engine oil
- Heat input and flow for transmission oil
- Heater core air temperature and flowrate
- Inlet air temperature of low temperature radiator

An initialization component is included with the system model to allow convenient, consistent initialization of the system and specification of the boundary conditions. The initialization component also includes the ability to override the engine calculations to allow isothermal simulations and simulation of the system at specified steady state boundary conditions.

Summary records are included for both the high temperature and low temperature circuits to allow easy access to relevant outputs, including flowrates, temperatures, heat rejection in the coolers, etc.

2.2 System Characterization

One of the main challenges in building the model of the Scorpion diesel cooling 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.

One key piece of characterization data is the flow and efficiency data for the mechanical pump. This data was not readily obtained from published literature. Hanon provided estimates of the flow and system head requirements for the circuits. This data was then used to characterize the mechanical pump model at different operating speeds. No mechanical efficiency data was provided so the pump was assumed to operate at a constant 55% efficiency. This value could be considered on the high side, but a conservative value was chosen so as to not bias results towards the electric pumps. For the calculation of energy to drive the pump, the fuel power required to drive the pump was calculated using the engine BSFC.

For the electric pump system variant, Hanon provided estimates of the pump efficiency and flowrates vs. head characteristics at various speeds based on development and actual hardware. The flow characteristics for the HCP-1KW pump used in the high temperature circuit are shown in Figure 5. The efficiency data provided mapped from hydraulic power to electric power. Since the electric pump system is included in a conventional system and to provide consistent comparisons of energy to drive the pump, the fuel power required to drive the electric pumps was calculated based on the efficiency data provided and an estimate of the alternator efficiency along with the engine BSFC. The alternator efficiency was assumed to be 55% at all operating conditions. With this approach, fuel energy comparisons can be made between the mechanical and electric pump systems.

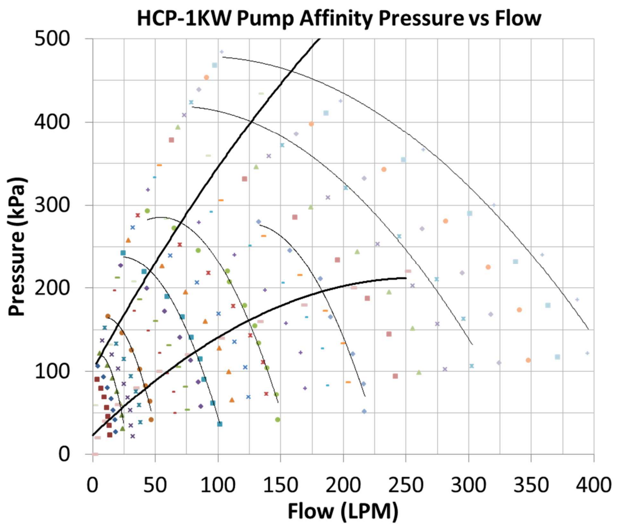


Figure 5. Flow characteristics for Hanon HCP-1KW pump used in high temperature cooling loop

Since no detailed flow information was available for the individual flow branches at different conditions, assumptions were made based on information for a few operating points to establish a flow distribution. The cooling system service manual provided information on the thermostat opening conditions for both the high temperature and low temperature circuits. As can be seen in Figure 1, lumped flow resistances were included in the various branches to allow calibration of the flow distribution. The flow resistances are parameterized with an operating point friction model that takes nominal flow and pressure drop data for a given operating point. Using information for the thermostats, isothermal flow tests were conducted at various operating conditions to calibrate the flow resistances in the model to provide the desired flow distribution. The flow characterization of the system remains fixed and only the pumps are switched to go from the mechanical to electric pump system.

The system volume for the high temperature and low temperature circuits were provided in the cooling system manual. The distribution of the volume within the individual circuit was provided based on rough judgement of the sizes of the various components. The volume of the oil circuits was also obtained based on available data and entered into the simplified circuits.

Following the flow characterization of the system, thermal characterization of the system is required to run reasonable thermal simulations. Typically this sort of characterization is easily provided based on heat exchanger performance characteristics that can be readily entered into the model. The heat exchangers in the system shown in Figure 1 include the high temperature radiator, heater core, engine oil cooler, low temperature radiator broken into upper and lower sections, EGR cooler broken into high temperature and low temperature sections, transmission oil cooler, fuel cooler, and charge air cooler. These heat exchangers are all modeled using the *StaticEffectivenessTable* model in Liquid Cooling Library. This model specifies the heat exchanger performance as a 2D effectiveness table based on the mass flowrates of the individual fluid streams. However, this sort of data was not readily available in public literature. To provide characterization data for the heat exchangers, Hanon provided data for similar types of heat exchangers based on simulated and actual hardware. This data was then scaled as needed to provide the effectiveness maps used in the model. The heater core was not characterized but was set to an inactive state in the model.

To characterize the radiator airflow, the flow areas to the high temperature radiator and the two sections of the low temperature radiator were estimated. A table for the grill factor as a function of vehicle speed was estimated. Using this table, the external air velocity was calculated. This velocity was then converted to a mass flow using the flow area parameters. Since the low temperature radiator is in front of the high temperature radiator, the external air outlet temperature from the low temperature radiator is used as the inlet temperature for the high temperature radiator. While manually considered in this model, detailed heat exchanger models using Heat Exchanger Library (Modelon AB, 2018) handles this stacking effect in a natural, distributed way based on component geometry and stack layout. These models could be integrated into the cooling system circuit but were beyond the scope of this effort given that they require design-oriented geometric data not readily available.

2.3 Electric Pump Control

The conventional mechanical system operates without active control for the coolant flow as the pump speeds are determined by the engine, and thermostats are passive flow control devices based on operational setpoints. For the system retrofit with electric pumps, a controller is implemented to control the pump speed. Figure 6 shows the electric pump controller. The controller operates based on target coolant temperatures for the various coolers. The commanded pump speed increases as coolant temperatures exceed the target temperatures. A minimum pump speed is specified to ensure that there is sufficient flowrate to avoid hotspots in the system. In addition, the pump hardware is designed to only operate in a particular speed range. A maximum pump flowrate is also specified to ensure that the pump stays within operational limits.

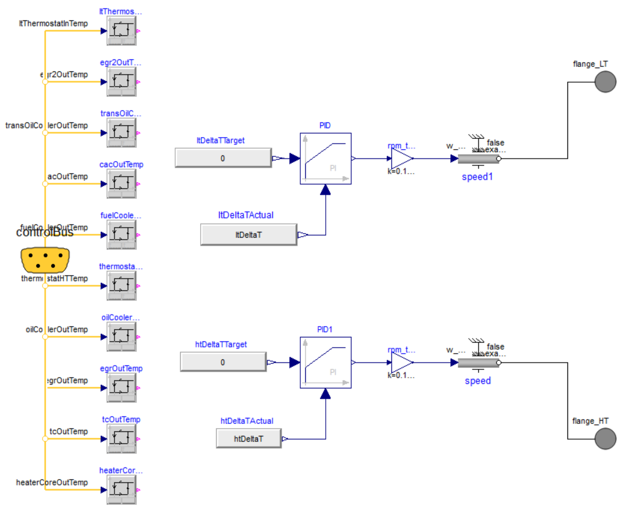


Figure 6. Electric pump controller

3 Simulation Results

Following the characterization of the system, a series of simulations were run to evaluate the potential of the electric pump system as compared to the baseline mechanical pump system. The simulations were conducted on the FTP cycle. Figure 7 shows operating conditions for the FTP cycle used for the simulations including the following plots from top to bottom:

- Vehicle speed [kph]
- Engine speed [rpm]
- Engine power [W]
- Fuel flow rate [g/s]
- High temperature radiator flow rate [kg/s]
- Low temperature radiator flow rate [kg/s]

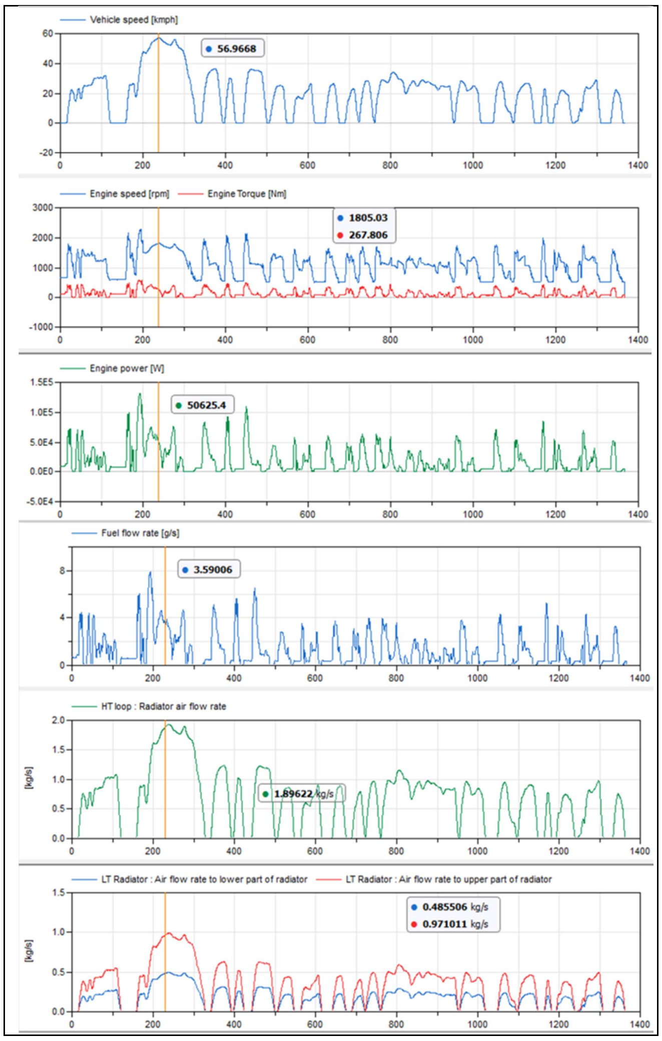


Figure 7. Operating conditions for FTP cycle

3.1 Baseline Electric Pump Controller

The electric pump controller is based on target coolant temperatures for various parts of the system. Unlike the mechanical pump system, the electric pump system can operate at lower pump speeds and flowrates. However, effectiveness of coolers can be low at low flowrates. With low effectiveness, there is minimal heat extracted in the cooler and thus no real effect on the coolant temperature. In this situation, the temperature of the fluid being cooled rises, but the electric pump controller will not respond to increase the flowrate since it is based on coolant temperature targets. Thus, a series of simulations were run to identify the appropriate minimum pump speed to ensure that the engine oil and transmission oil temperatures are kept under control. Figure 8 shows a comparison of the engine oil and transmission oil temperatures for the mechanical pump system and electric pump system at different minimum pump speeds. Based on these simulations, the minimum pump speed is set to 2000 RPM for the high temperature electric pump and 3000 RPM for the low temperature electric pump to provide similar temperatures as in the baseline mechanical pump system.

Figure 9 compares coolant temperatures in and out of the high temperature radiator along with the thermostat opening and mass flowrate through the radiator for the mechanical and electric pump systems. The coolant profiles are very similar.

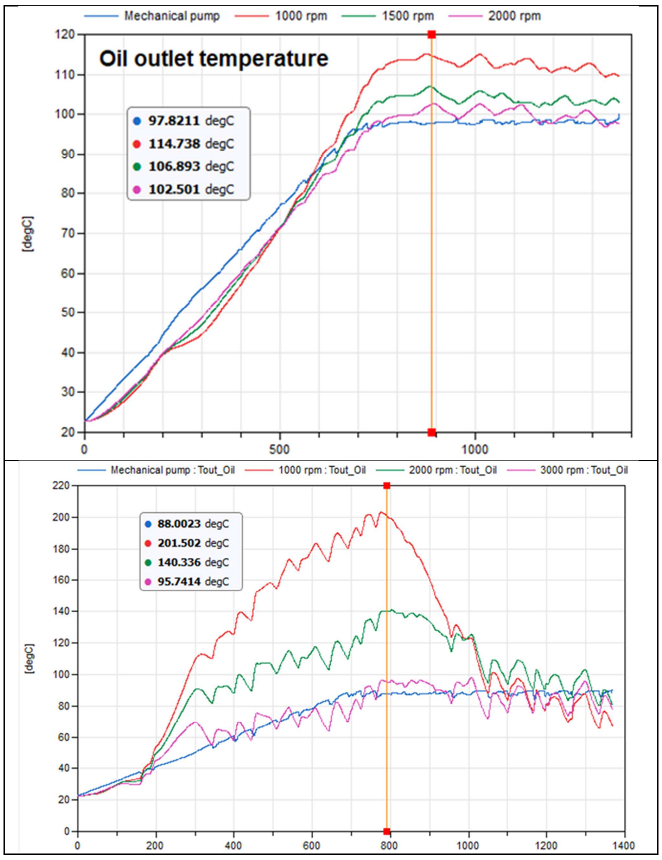


Figure 8. Effect of minimum electric pump speed on engine and transmission oil temperatures

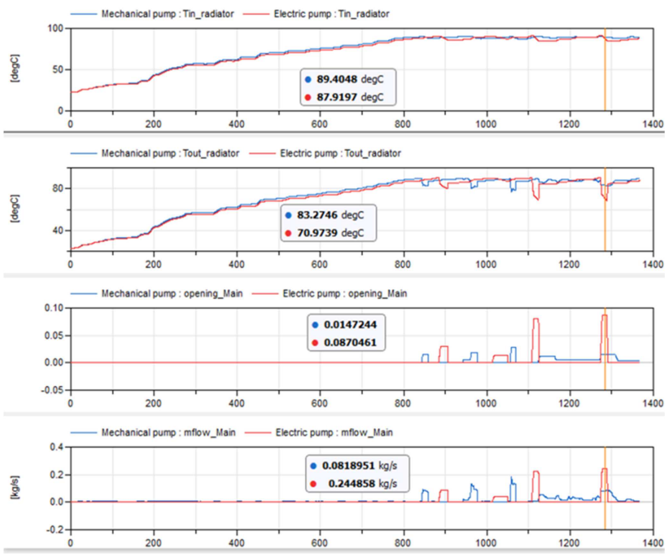


Figure 9. Comparison of coolant temperatures and flowrates at high temperature radiator

Figure 10 compares high temperature pump conditions between the mechanical and electric pump systems. Since the FTP cycle is such a lightly loaded cycle, the electric pump system can operate at the minimum pump speed for the entire cycle. A comparison of the pump flowrates shows how much flow circulates in the mechanical pump system due to the linking of pump speed with engine speed. Even though the overall efficiency of the electric pump system is lower than that of the mechanical pump system, the total pump energy is significantly less in the electric pump system as the hydraulic power requirement is so much lower due to the lower flowrates.

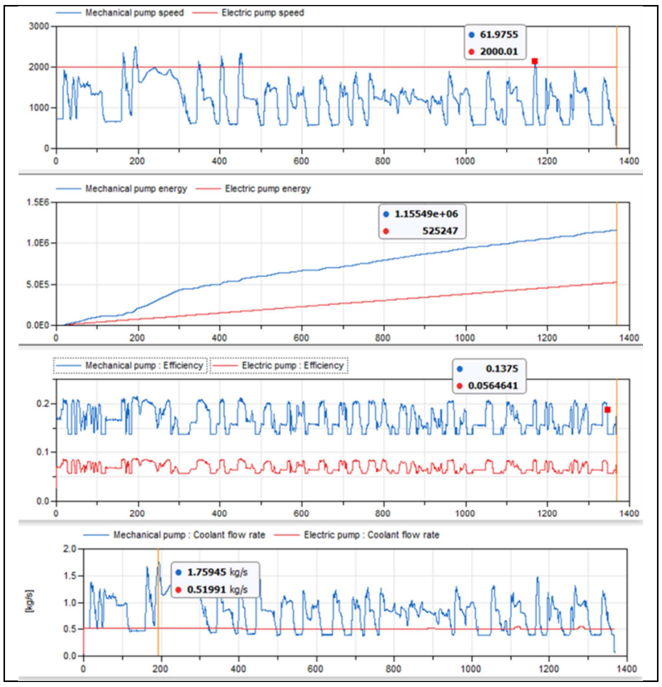


Figure 10. Comparison of high temperature pump conditions for mechanical and electric pump systems

Figure 11 compares temperatures in the low temperature circuit between the mechanical and electric pump systems. Temperatures are similar between the two systems though slightly higher in the electric pump system.

Figure 12 compares low temperature pump conditions between the mechanical and electric pump systems. Again, the electric pump system can run at the minimum pump speed for the entire cycle. As with the high temp pump, the overall flowrates are significantly less in the electric pump system. The overall efficiency of the electric pump system is again lower than the mechanical pump system. For the low temperature circuit, the total pump energy for the electric pump system is larger than the mechanical pump system as the reduction in hydraulic power is not enough to offset the reduced efficiency.

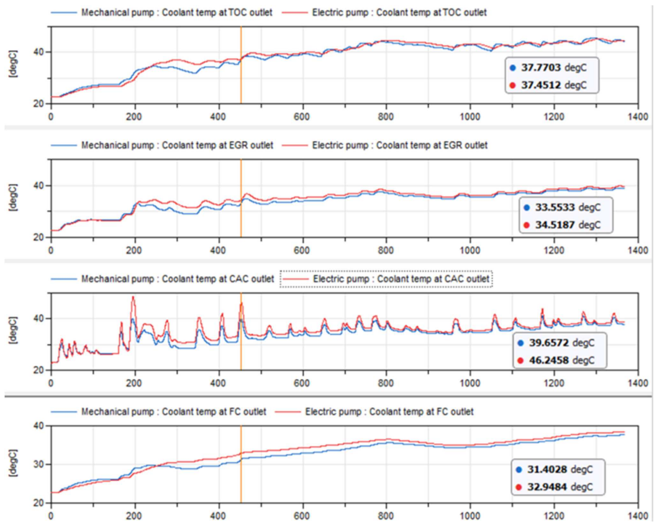


Figure 11. Comparison of temperatures in the low temperature circuit

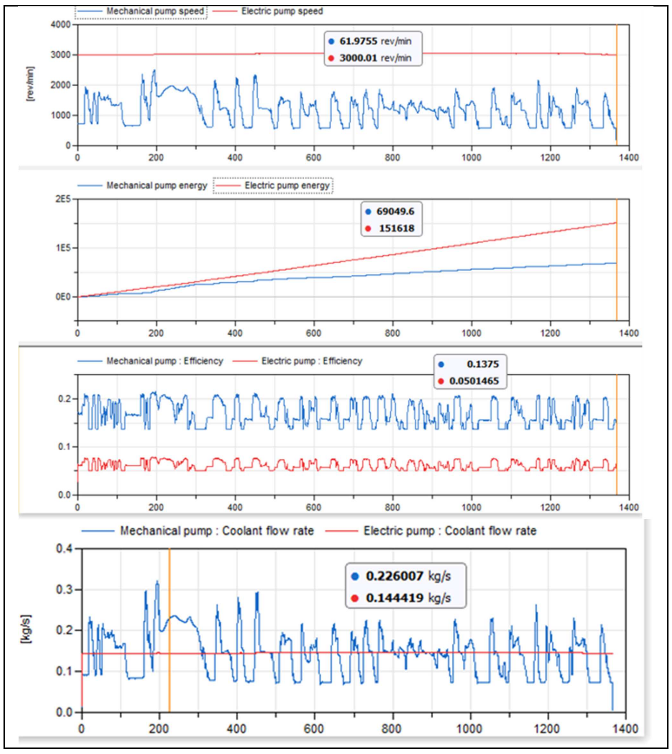


Figure 12. Comparison of low temperature pump conditions for mechanical and electric pump systems

Table 1 shows a comparison of the fuel consumption between the mechanical and electric pump systems. The fuel consumption on the cycle is reduced from 1.6% of the total cycle fuel to approximately 0.9% of the total cycle fuel with the electric pump system. As discussed previously, the electric pump system provides a benefit on the high temperature circuit but not on the low temperature circuit due to the flowrate required to manage the transmission oil temperatures given that the mechanical pump system is inherently more efficient.

Table 1. Fuel consumption comparisons for the mechanical and electric pump systems

	Fuel consumption [g]	% of total fuel consumption
Vehicle	1762.3	
HT Mechanical pump	26.62	1.5
HT Electric pump	12.10	0.69
LT Mechanical pump	1.591	0.090
LT Electric pump	3.493	0.198

3.2 Electric Pump Controller with Oil-Based Control

As described in Section 2.3, the baseline electric pump controller is based on target coolant temperatures. However, this control scheme requires that the minimum pump speed is set to control oil temperatures indirectly. As seen in the previous section, the electric pump system basically operates at minimum pump speed due to the light loads in the FTP cycle.

Another potential benefit of the electric pump system is the ability to accelerate the warmup of the engine and transmission oil by controlling the flow to the oil coolers. This benefit translates into fuel consumption due to reduced losses and friction due to oil temperatures that more rapidly reach the desired operating range. To evaluate this potential, the control algorithm was modified to explicitly control the engine and transmission oil temperatures in addition to target coolant temperatures for the other coolers. Figure 13 shows a comparison between the two different electric pump control strategies for the engine and transmission oil temperatures. By explicitly considering the oil temperatures in the electric pump strategy, the electric pump system can deliver oil temperatures that more quickly reach desired operating range while still managing the maximum temperature constraints.

Though the additional steps to translate these operating temperature benefits into fuel consumption metrics were beyond the scope of this work, they are being considered for further development of the simulation platform for Hanon. Potential future work also includes simulation of different vehicle cycles to evaluate fuel economy potential on a wider range of relevant usage profiles.

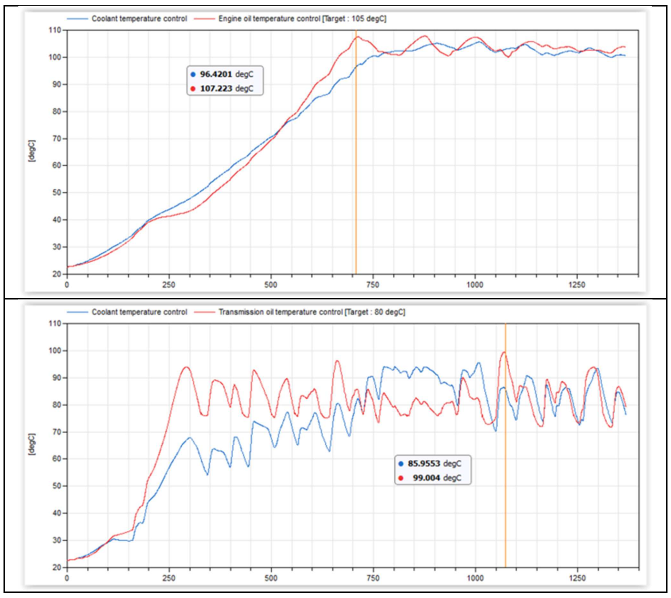


Figure 13. Comparison of electric pump systems with coolant only and coolant + oil temperature control, engine and transmission oil

4 Summary

This paper describes an effort to develop an analytic platform for Hanon Systems to evaluate the electrification potential for powertrain cooling systems. This analytic platform allows rapid virtual prototyping of different cooling systems to evaluate the potential of Hanon hardware and controls solutions for systems optimization. This platform was demonstrated on a model of the diesel cooling system for the Ford Power Stroke diesel 6.7L V8. A baseline model of the mechanical pump system was built and characterized for flow and thermal response. An electric pump variant of the system was built by replacing the mechanical pump with electric pumps from Hanon. Two different electric pump control strategies were implemented. The electric pump system demonstrated a fuel economy benefit when evaluated on the FTP cycle and also showed the potential benefit for more rapid warmup of engine and transmission oil with a modified control algorithm.

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 mechanical pump and heat exchanges would greatly improve model accuracy. Quantifying the benefits of the increased warmup of the oil temperatures for reduction in friction and losses is also a focus of future work. Integrating the cooling system model with a full vehicle simulation would reduce the need for driving the simulations with engine conditions and pick up additional interactions with the vehicle loads. Simulation on different vehicle cycles would also allow the evaluation of fuel economy potential on a wider range of usage profiles.

While this work focused on simply replacing the mechanical pump with electric pumps, future work with this modeling capability includes evaluation of concepts to redesign the system and develop control strategies to take full advantage of the capability of the electric pump system to deliver flow on demand to individual coolers in the high and low temperature circuits.

Acknowledgements

The authors would like to thank Chris Leonello and Ondrej Pavlica from Hanon Systems for providing surrogate heat exchanger data to support thermal characterization of the system.

References

Arvind Thiruvengadam, Saroj Pradhan, Pragalath Thiruvengadam, Marc Besch, Daniel Carder, and Oscar Delgado. Heavy-Duty Vehicle Diesel Engine Efficiency Evaluation and Energy Audit, Final Report, October 2014, https://www.theicct.org/sites/default/files/publications/HDV_engine-efficiency-eval_WVU-rpt_oct2014.pdf

John Batteh, Jesse Gohl, Sureshkumar Chandrasekar. Integrated Vehicle Thermal Management in Modelica: Overview and Appliations. *Proceedings of the 10th International Modelica Conference*, March 10-12, 2014, Lund, Sweden. doi: 10.3384/ECP14096409.

C. Bouvy, P. Jeck, J. Gissing, T. Lichius, L. Ecksterin. Holistic Vehicle Simulation using Modelica - An Application on Thermal Management and Operation Strategy for Electrified Vehicles. *Proceedings of 9th International Modelica Conference*, pp. 263-270, 2012.

Deraad, S., Fulton, B., Gryglak, A., Hallgren, B. et al., "The New Ford 6.7L V-8 Turbocharged Diesel Engine," *SAE Technical Paper 2010-01-1101*, 2010, https://doi.org/10.4271/2010-01-1101.

I. Krüger, A. Mehlhase and G. Schmitz. Energy Consumption of Battery Cooling In Hybrid Electric Vehicles. *Proceedings of 14th International Refrigeration and Air Conditioning Conference*, 2012.

Modelon AB, Lund, Sweden. (2018). *Heat Exchanger Library*. http://www.modelon.com/products/modelon-library-suite/heat-exchanger-library/

Modelon AB, Lund, Sweden. (2018). *Liquid Cooling Library*. http://www.modelon.com/products/modelon-library-suite/liquid-cooling-library/

Massimo Stellato, Luca Bergianti, and John Batteh. Powertrain and Thermal System Simulation Models of a High Performance Electric Road Vehicle. *Proceedings of the 12th International Modelica Conference*, May 15-17, 2017, Prague, Czech Republic. doi: 10.3384/ecp17132171.

Jason Zheng. Multi-Disciplinary Approach to Thermal Systems Design and Optimization. *SAE Thermal Management Systems Symposium*, 18TMSS-0012, October 9-11, 2018. San Diego, CA.

SESSION 2A: BUILDINGS 2

Dynamic Simulation of Residential Buildings Supporting the Development of Flexible Control in District Heating Systems

Nadine Aoun[1,2,3] Roland Bavière[2] Mathieu Vallée[2] Adrien Brun[2] Guillaume Sandou[1]

[1] L2S, CentraleSupélec, Gif-sur-Yvette, France, {Nadine.Aoun, Guillaume.Sandou}@centralesupelec.fr
[2] CEA, LITEN, Grenoble, France, {Nadine.Aoun, Roland.Baviere, Mathieu.Vallee, Adrien.Brun}@cea.fr
[3]ADEME, Angers, France, Nadine.Aoun@ademe.fr

Abstract

Load shifting, peak shaving and night-time setback are key demand-side management measures to make the operation of District Heating Systems (DHSs) more flexible and efficient. These goals can be achieved through appropriate control strategies exploiting the building's and space heating system's thermal inertia. To ease the development of such an advanced controller, we programmed a detailed dynamic Modelica simulator representative of French multi-stories radiator-heated residential buildings. We parametrized the simulator to vary the factors influencing the flexibility potential of a building (e.g. envelope properties, additional internal mass such as partition walls and furniture, the heating system…). This helped us designing a reduced-order building model relevant to our application and setting up a robust identification method for its parameters. We finally used the detailed simulator to test an optimal space-heating controller, thereby allowing many incremental improvements without jeopardizing end-users thermal comfort. This simulation work paves the way to considering the actual implementation of our advanced controller on a real building.

Keywords: District Heating System, Optimal Control, Building Simulation, Reduced-order building model

1 Introduction

1.1 Context of this research

District Heating (DH) has been known for many years as an efficient mode for space heating and domestic hot water preparation in dense urban areas. District Heating Systems (DHSs) have genuinely an important role to play in the future of sustainable energy systems (Lund et al., 2014, 2010) as they allow greater integration of renewable power and recycling of low-temperature excess heat; therefore, a substantial reduction in fossil fuel consumption, CO_2 emissions as well as heat production costs can be achieved by converting from individual to district heating. Yet exploiting the full potential of a DHS relies on advanced management at 3 levels: production, distribution and demand. Demand-Side Management (DSM) of DHSs is a key measure for peak load shaving. It consists in modulating the heat demand for buildings' space heating by using the available thermal inertia for a free short-term heat storage. Relying on this technology at a city scale, DH production load could be reduced at peak hours thus avoiding the start-up of expensive and pollutant fossil fuel generation units.

Our research group is involved in the design of an optimal space-heating controller for residential buildings connected to DHS. Within the FP7 City-Zen project (City-zen, 2018) we will demonstrate the use of this controller on a building located in the city of Grenoble, France. To support our work during the design and validation phases, we developed a detailed building dynamic simulator. This paper reports on the development of the simulator and its use in the research context we have just described.

1.2 Structure of this paper

We organized the remaining part of this paper as follows. Section 2 gives an overview of the programming languages and simulation environments suitable to our application. We then describe the detailed building simulator in section 3. In section 4 we present how we used the simulator to develop and assess an optimal space-heating controller. In section 5, we discuss the obtained results and conclude our study.

2 Simulation environments

This part is focusing on building thermal simulation and more broadly on platforms integrating Building Energy Model (BEM). Figure 1 shows examples of applications and related simulation tools. It also indicates an order of magnitude of the number of buildings modelled for each of the applications and the category to which the building model belongs. BEM could be split into two categories, the classic and the simplified.

The classic approach is originally designed for a stand-alone building: TrnSys, EnergyPlus, Pleiade, IDA-Ice, BuildSysPro, Buildings… The building is broken down into a set of walls and volumes. The geometric description can be very realistic. The main assumptions that are made are the unidirectional conductive transfers, and uniform variables on the air volumes. A detailed description of modelling methods is proposed in (JA Clarke, 2001) and (Bruno Peuportier, 2016). These simulation tools have been the subject of numerous benchmarks (Judkoff and Neymark, 2013), (Brun et al., 2009) and experimental validations.

The simplified approach generally uses analogy between electricity and heat transfer to represent models as an electrical circuit. This type of modelling assumes a linearization of the long wave radiative transfers and constant heat transfer coefficient. (Foucquier et al., 2013) presents and assess other simplifications that are generally made: merging thermal zones, merging walls and reducing walls discretization. This approach is used in building control-command, energy diagnosis but also when the number of building is important (network control, micro-climate, urban energy flow). There is a wide variety of Resistance-Capacitance (RC) scheme and no dedicated benchmark.

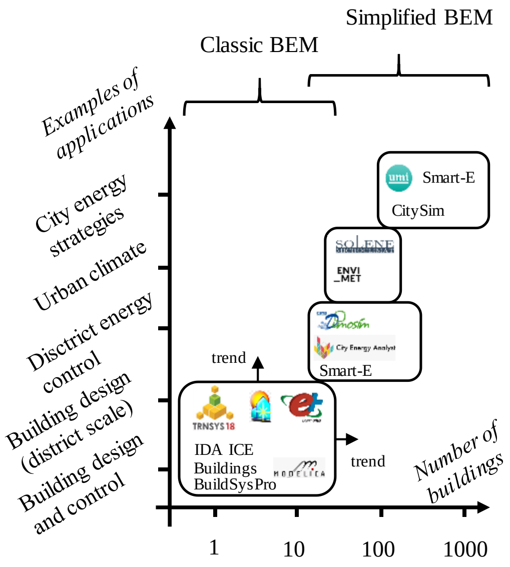


Figure 1. Building simulation environments for various applications and number of buildings.

We can notice that our field of application is the one for which some tools are based on a detailed modelling and others on a simplified modelling. There is currently no consensus on the approach to be used. (Frayssinet et al., 2017) shows that more detail envelope meshing than usual simplified BEM is needed when studying power demand. (Perez et al., 2015) presents the R7C4 mono-zone model developed to consider the major phenomena in the DIMOSIM simulation platform. In their opinion, classic BEM is not appropriated to simulate a lot of building at the same time due to their high computational time and/or required parameters. (Nageler et al., 2018) presents a study with 34 buildings modelling by means of classic BEM and data driven method. They show that this is technically feasible. (Ribault et al., 2017) showed that Energyplus has interesting feature for decision-support tools for urban densification in a 22 buildings model district. The ease of implementation of distributed computing tends to move the boundary and allow the use of detailed models in large numbers.

In view of the above, we used the Modelica language that allowed us to implement both approaches. A classic BEM for the building simulator (see section 3) and a simplified approach for the optimal space-heating controller (see section 4.1).

3 The building simulator

3.1 Generalities

The building simulator is a generic, easily parameterized Modelica model of a multi-storeys building with the main vocation of generating reliable data in replacement of real in-situ measurements. We made some simplifications to obtain representative results while maintaining the parametrization burden tractable. An important design goal of our simulator is the ability to produce numerical results at the expense of reasonable simulation run times.

We built our simulator as a pile of thermally connected identical floors. For simplicity, we considered a rectangular footprint and we discretized each floor into 4 thermal zones, with configurable surface fractions, as shown in Figure 2. The orientation of the building simulator is set using the Θ azimuth angle between the North direction and the building main axis (see Figure 2). The default value of 0 for Θ can be used to represent and "ideal" orientation where equivalent Night, Day, Kitchen and Bathroom zones are facing North, South, West and East, respectively.

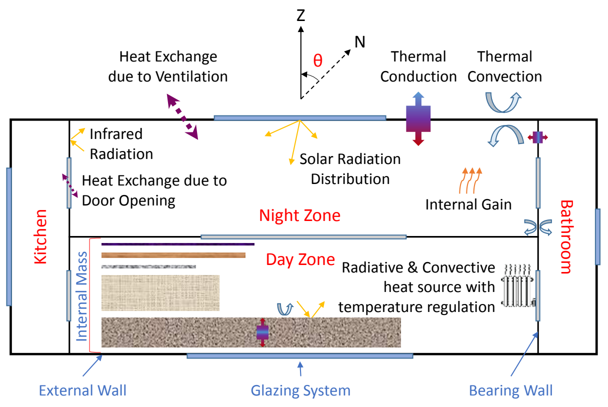

Z
N
θ
Heat Exchange due to Ventilation
Thermal Conduction
Thermal Convection
Infrared Radiation
Solar Radiation Distribution
Heat Exchange due to Door Opening
Internal Gain
Night Zone
Kitchen
Bathroom
Internal Mass
Day Zone
Radiative & Convective heat source with temperature regulation
External Wall
Glazing System
Bearing Wall

Figure 2. Spatial discretization of one floor (top view) showing the modelled elements and the thermal phenomena considered in the simulator.

We represent each thermal zones using a *MixedAir* model from the Modelica *Buildings* library (Wetter et al., 2011). Thus, our simulator considers transient heat conduction through opaque walls, heat transfer through glazed surfaces (with consideration of solar and infrared irradiations), and external/internal convective and radiative heat transfers. Our simulator also includes a hydronic space-heating system composed of a centralized production unit, distribution pipes and radiators each equipped with a thermostatic valve. We used the *RadiatorEN442_2* model from the *Buildings* library and models from our own Modelica *DistrictHeating* library (Giraud et al., 2015) for the distribution pipes, the centralized production unit and the thermostatic valves.

We also implemented a stochastic model of internal gains at the thermal zone level. We statistically model the signal for each zone by combining three heat sources related to occupancy profile, electric appliances and domestic hot water use at a 10 minutes time step. Finally, we expose the simulator to meteorological boundary conditions; we used the *ReaderTMY3*, a weather file reader from *Buildings* in which we can upload Typical Meteorological Year (TMY) data for various cities in France and Europe. An interested reader will find further modelling details in the following sections.

3.2 The envelope and the internal structures

External walls forming the envelope integrate glazing systems of specific height and width, with neither overhangs nor side-fins. Thermal zones are separated by bearing walls. As for the zones' interior, we carried out the modelling of internal partition walls and furniture with special attention since their mass is potentially a significant contributor to short-term storage. In fact, many studies have found that thermal inertia of building's internal mass has the potential, under certain conditions, to maintain a decent comfort level inside the building for hours after cutting off, or reducing, the heating power (Antonopoulos and Koronaki, 2000; Le Dréau and Heiselberg, 2016; Wolisz et al., 2015). Therefore, empty zones would not reflect the correct dynamics of the building. Furnishing elements and light partition walls are modelled as horizontal and vertical slabs, respectively with the properties listed in Table 1. We referred to (Johra and Heiselberg, 2017), a survey on the internal mass and its equivalent heat capacity found in residential and single office buildings in Denmark, to set these material properties, mass and dimensions of furniture equivalent slabs.

Table 1. Properties of the internal mass equivalent slabs: Thermal conductivity (k), Specific heat capacity (c), Density (ρ), Mass per zone area (m) and thickness (ε).

Material	**k** $(\mathbf{W/m\cdot K})$	**c** $(\mathbf{J/kg\cdot K})$	**ρ** $(\mathbf{kg/m^3})$	**m** $(\mathbf{kg/m^2})$	**ε** $(\mathbf{mm})$
Metal	60	450	8000	25	3
Wood / Plastic	0.2	1400	800	25	18
Ceramic / Glass	1.25	950	2000	5	10
Light material	0.03	1400	80	15	120
Light partition walls	0.015	1150	384	25	100

3.3 Thermal phenomena within the zones

This section details the physical modelling of the considered thermal phenomena, symbolically depicted in Figure 2.

Object-oriented Modelica language allows reusability of pre-developed and validated components. In our simulator, we rely on a thermal zone model, called *MixedAir* found in the Modelica *Buildings* library and we use to it model each of the building's zones. *MixedAir* is a volume of homogenous medium, typically ambiance air, with boundary elements including walls, slabs, windows, floor and ceiling. These construction elements, also found in *Buildings* library, may be exposed to external meteorological conditions via a weather bus reading a weather file, or boundary conditions of adjacent thermal zones in the case of a shared wall, or the boundary conditions of the same thermal zone in the case of internal partition walls. Under dynamic simulation and due to temperature differences, the volume of air exchanges heat with its surroundings, thus affecting its thermal states and those of the surroundings elements. Here is a concise description of the thermal phenomena that are modelled within *MixedAir*, further details may be found in (Wetter et al., 2011).

- Convection

Thermal convection on both sides of each construction element. Two options are available, either using a temperature, flow and tilt dependent convection coefficient or one with a fixed value. In our simulator, we selected a fixed coefficient of $3\ W/m^2$ for internal convection and $10\ W/m^2$ for external convection.

- Conduction

Thermal conduction through multi-layers construction elements is assumed to be mono-directional and computed by solving the heat equation after discretization into a number of states. For each layer, the number of states is proportional to the ratio between the layer thickness and the square root of the material's diffusivity:

$$nsta \propto \varepsilon \cdot \sqrt{\frac{c\cdot\rho}{k}} \qquad (1)$$

- Radiation

MixedAir has a complex model for solar radiation thoroughly described in (Wetter et al., 2011) In short, solar radiation that penetrates the unshaded windows is computed. First it strikes the floor construction where part of it is absorbed and the rest is reflected towards the walls and the ceiling. Surfaces then exchange longwave radiation between each other according to the Stephan-Boltzmann law which may optionally be linearized. Additionally, other sources of radiation may be injected, for instance radiative heat from internal gain or from a heating system. An interesting output from the radiation model embedded in *MixedAir* is the room's radiative temperature roughly equal to the average temperature of all the internal surfaces. Note that in our simulator all windows are simulated with no shades for simplicity.

- Mass transfer

Although *MixedAir* is designed with a fluid port for explicit modelling of aeraulic flows, in our simulator direct heat transfer to the outdoor environment and between zones respectively due to ventilation and door opening is modelled as follows:

- Simple-flux ventilation

$$\Phi_{zone \to outdoor} = \rho^{air} \cdot c_p^{air} \cdot V \cdot n \cdot (T_{outdoor} - T_{zone}) \quad (2)$$

where $T_{outdoor}$ and T_{zone} are the $\rho^{air}(kg/m^3)$ and $c_p^{air}\,(J/kg \cdot K)$ respectively stand for the density and specific heat capacity of air, $V(m^3)$ is the air volume of the thermal zone and n is the number of volume changes per second. In reality n is often variable and should be stochastically modelled depending on tenant's behaviour, however in our work we assume it to be constant to a value recommended under European standards. A typical value for n ranges from 0.2 to 0.6 volume changes per hour (*ASHRAE Standard*, 1989). The default value in the simulator is 0.3.

- Door opening

$$\Phi_{i \to j} = \rho^{air} \cdot c_p^{air} \cdot S_{Door} \cdot v_{mix} \cdot \left(T_{zone[i]} - T_{zone[j]}\right) \quad (3)$$

where $S_{Door}(m^2)$ is the open area separating two adjacent zones, $v_{mix}(m{:}s)$ is an equivalent mixing air velocity through the door opening. The default value for v_{mix} is 0.13 m/s (Van Schijndel et al., 2003).

The *MixedAir* model can then be subject to external heat flows connected through two ports: one for convective heat and another for radiative heat. External heat sources are typically the heating system and the internal heat gain due to electric appliances and occupancy. The following sections describe how these latter sources have been modelled.

3.4 The space-heating system

This section describes the modelling of the heating system. The model is composed of radiators fed by a two-pipe distribution network that connects a centralized production unit, located in the building basement, to the heated rooms.

As already stated in section 3.1, we used the *RadiatorEN442_2* model from *Buildings,* which includes computation methods inspired by the EN-442 European standard. To favour numerical efficiency, we limited the discretization level to 3 fluid control volumes.

We developed a model of thermostatic valve to control the hydronic flow through each radiator. The first part of the model is a heat capacity exchanging heat with its environment and representing the sensing bulb of the valve. We defined the heat exchange coefficient using a standard correlation valid for natural convection flows around a vertical cylinder. In most practical situations, this leads to an equivalent thermal time constant (defined as the ratio between inertia and the sensing-bulb to environment thermal conductivity) of approximately 10 minutes. The second part of the model relates the position of the valve to the difference between the sensing bulb temperature and the set-point value. Our model fulfils the specifications of the European standards NF EN 215 to regulate the internal air temperature around a specific set point temperature.

We also used pairs of pre-insulated tubes, available in our Modelica *DistrictHeating* library (Giraud et al., 2015) to model the building's internal space-heating network. These tubes account for the hydraulic head losses and thermal losses occurring in the system. The model also accounts for heat accumulation in the tubes. Figure 3 describes the model we used to represent the centralized production unit of the building's space-heating system. This unit can represent a substation when the building is connected to a DHS. As can be seen in the figure, the thermal power injected in the system at the substation, hereafter denoted Φ_{SST}, is controlled by a cascade of two regulators. "Regulator 1" controls the supply temperature denoted T_s by adapting Φ_{SST}. The set-point value for T_s is traditionally provided by a heating curve, whose output is noted T_{HC}, T_{ext} being the external temperature. In our case, the set-point value for T_s can be lower than T_{HC} ; it is then provided by "Regulator 2" which is fed by a set-point value for Φ_{SST}, denoted $\Phi_{SST}^{Set\ point}$, and an indirect measurement of Φ_{SST} built upon the mass flow-rate ($\dot{m}$ in Figure 3), and the supply and return (T_r in Figure 3) temperatures. The reasons that guided us to design this control strategy are twofold. First, it can be implemented on existing systems and second, its architecture allows shifting between a traditional temperature-driven mode to a more advanced mode where the heating power is planned using a Model Predictive Control (MPC) approach. This aspect is illustrated in section 4 of the present paper.

3.5 Internal heat gain model

Each zone in the building simulator receives a direct internal heat gain flux, half of which is assumed to be convective and the other half is radiative. *MixedAir* handles these heat fluxes and integrates them into the heat balances of the air and the radiative exchange respectively. *MixedAir* can also handle latent heat gain, yet it is not used in our work for simplification.

The original stochastic internal gain signal, which is then divided into the two mentioned halves, is modelled beforehand, separately by combining three heat sources related to occupancy profile, electric appliances and domestic hot water use. Generating this signal requires a database with information concerning the presence schedule of occupants inside the building, and whether or not they are active or not (i.e. sleeping). In both cases, the occupants' presence generates heat due to their metabolism, and furthermore when they are active, their presence triggers the possibility of using electric devices, such as stoves, ovens and laundry equipment. Whereas other appliances are independent of the activity of tenants, such as refrigerators. All appliances dissipate heat as a fraction of their power input and with a certain delay due to their relative thermal inertia. Domestic hot water usage and its temperature level also affect the signal of internal heat gain. Unfortunately, all the data needed to build the internal gain profile is not available for contemporary households in France. Luckily, a survey was carried out in the UK in 2000 and the collected data concerning the occupancy profiles and the electric devices usage is available for the modelling of internal gain signal of our work (Richardson et al., 2010, 2008). We used Markov chains to model a realistic evolution of the signal based on these data. We referred to the work in (Paatero and Lund, 2006; Widén et al., 2009; Yao and Steemers, 2005) for the modelling of the fraction of dissipated heat from the electric appliances and the domestic hot water. The model generates a signal per zone for a year with a 10 minutes step in accordance with the magnitudes found in the French thermal regulation [1]. We then connect the profiles to the building simulator.

3.6 Parametrization

The simulator can be parameterized to describe various types of buildings. Yet there is one particular building of interest in our work for the upcoming experimental demonstration of the advanced control strategy; it is a newly built, 8 stories residential building called *Le Salammbô*, situated in the neighbourhood of Zac Flaubert in Grenoble – France. It has been constructed in accordance with the recommendations of the latest European standards related to buildings thermal consumptions (RT 2012 [2]) and consumes 20% less than the threshold set by the standards. It is connected to a low-pressure district heating loop and serves as a demonstrator in the European project City-Zen (City-zen, 2018).

However, according to Tabula [3], a statistical study of the French residential buildings from a thermal point of view, *Le Salammbô* (constructed in 2016) is not representative of buildings of its category (multifamily house) in France (see Figure 4).

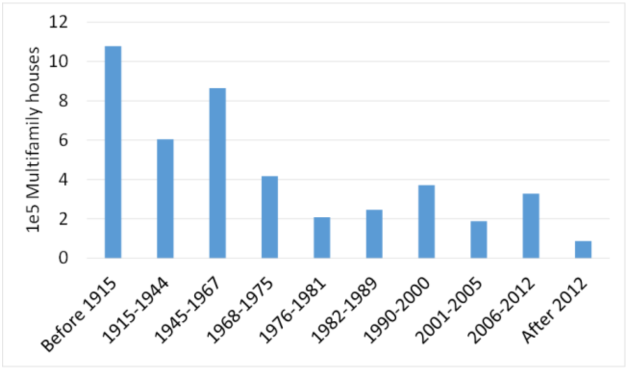


Figure 4. Number of multifamily houses per construction period (Source Tabula [3]).

In order to carry out a more inclusive and representative research on advanced control strategies of space-heating demand in DHSs, we decided to parameterize 3 different simulators with the same

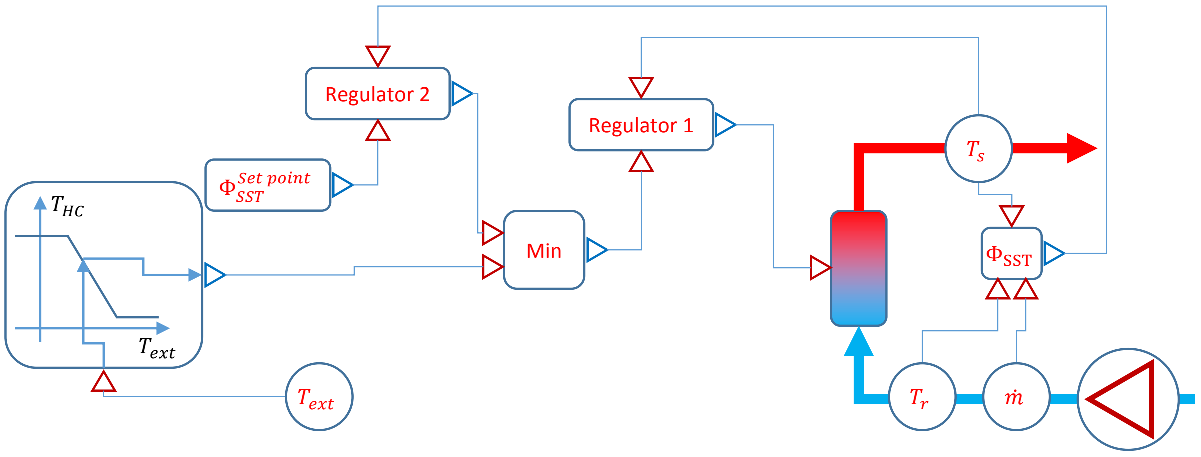


Figure 3. Schematic view of the centralized space-heating production unit composed of hydraulic connections, a circulation pump, the heat generator (right) and the controllers (left).

[1] Arrêté Du 30 Avril 2013 Portant Approbation de La Méthode de Calcul Th-BCE 2012 Prévue Aux Articles 4, 5 et 6 de L'arrêté Du 26 Octobre 2010 Relatif Aux Caractéristiques Thermiques et Aux Exigences de Performance Énergétique Des Bâtiments Nouveaux et Des Parties Nouvelles de Bâtiments. Annexe Détaillant La Méthode de Calcul Th-BCE 2012. 2017

[2] http://www.gbpn.org/databases-tools/bc-detail-pages/france#General%20Information

[3] http://episcope.eu/fileadmin/tabula/public/docs/brochure/FR_TABULA_TypologyBrochure_Pouget.pdf

geometric parameters as *Le Salammbô* but with different construction materials found in Tabula [3] representing:

- A recent building constructed after 2012 because it is representative of *Le Salammbô* itself.
- A building constructed between 1976 and 1981; this category is thermally interesting because it comes just after the year where the first European standards concerning buildings thermal performance (RT 1974) have been introduced.
- A building constructed before 1915, since it is the most commonly found in France.

Table 2. Main thermal characteristics for 3 building simulators.

Simulator	**Envelope**	**Glazing system**	**Number of air renewal per hour**	**Sizing heating power (kW)**
After 2012	Concrete, exteriorly insulated with 16 cm of expanded polystyrene	double-glazed with 16 cm of argon	0.3	56.2
Between 1976 and 1981	Cinderblock, exteriorly insulated with 4 cm of expanded polystyrene	double-glazed with 6 cm of air	0.4	88.6
Before 1915	Stone (40 cm-thick), uninsulated	single-glazed	0.5	134.8

The main thermal characteristics of the simulators are reported in Table 2. Note that the sizing power is estimated as the building's thermal losses under extreme conditions of -11°C external temperature (the sizing temperature used in the city of Grenoble) with no solar radiations of internal heat gain.

3.7 Numerical performance

In this section, we report on the numerical performance of the simulator. We monitored the numerical efficiency of a 8 storeys building simulator, amounting 32 thermal zones and radiators. The corresponding computational problem weighs $79\,k$ non-trivial equations. We translated the model using DYMOLA 2019, compiled it with Microsoft visual C++ build tools 2015, and executed it on a Dell Power Edge $R640$ server, operated by Windows Server 2016 equipped with two Intel Xeon Gold 6154 3 GHz processors of 18 cores each. For numerical efficiency reasons, we disabled multi-threading. We also decided to enable the Node Interleaving option thereby configuring the server as a Symmetric Shared Memory Multiprocessing (SMP) computer. These settings where found to be optimal for the parallel execution of a simple "for" loop with reduction using OpenMP.

Our test consists in executing simulation runs using various solvers and computing options. The simulations are run for typical winter meteorological conditions. The execution time is expressed under the form of an acceleration factor Acc_f defined as the ratio between the simulated period to the execution time. Thus, $Acc_f = 168$ means that a simulation covering a period of 1 week lasts 1 hour since a week is composed of 168 hours. We limited the maximal time step used by the integrator to $900\ s$ by generating time events.

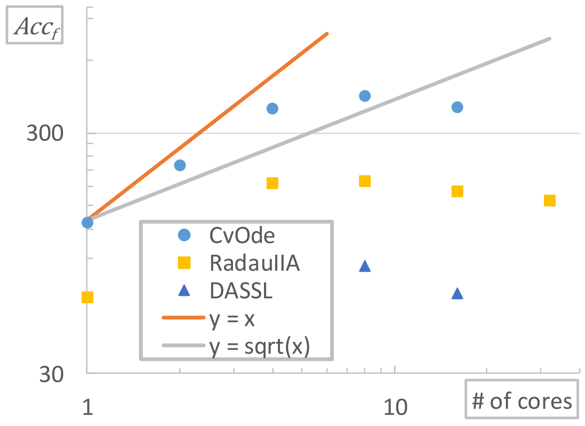


Figure 5. Execution time, expressed as an acceleration factor with respect to real time, as a function of the number of cores and type of solver for a **8** storeys building simulator.

Figure 5 presents the obtained results. The simulation runs are generally shorter when the number of cores used for the calculation increases. Figure 5 shows a quasi-linear trend when the number of cores involved in the calculation is low. However, above 8 cores, there are no more benefits in parallelization. A second observation is that when appropriate solver settings are used, the model execution time can reach $Acc_f = 420$. Such execution speed is well suited for control applications implying simulation periods in the day to week range. A reduction in the number of thermal zones and/or number of storeys would be necessary to perform annual simulations.

4 Using the simulator to assess an optimal space-heating controller

In this section, we describe how we used the simulator to design and test an optimal space heating controller enabling load shedding for DH network. More specifically, we used the simulator to perform three essential steps:

1. Designing a reduced-order building model relevant to our application (section 4.1)
2. Setting up a robust identification method for this model (section 4.2)
3. Validating the obtained optimal space-heating controller, within our simulation tool Pegase (section 4.3)

One important aspect to underline is that a key issue in the context of our application is the need to perform the building model identification as well as the optimal control with very limited measurements inside the building itself. In particular, optimal space heating control would be very beneficial to DH operators, however they typically have no access to detailed information or measurements inside the building they are heating. In this context, the detailed and versatile building simulator described in the previous section 3 provided a perfect environment for validating the proposed strategy on a range of different buildings, so that to ensure its reproducibility using real-world data.

4.1 Reduced-order building model design

A first step in the proposed optimal control approach is the design of a reduced-order building model (ROM). In our case, we are specifically considering linear or linear-saturated models, which are suitable for a mixed-integer linear programming (MILP) optimization. Although various optimization techniques could be used to perform optimal control, the MILP formulation provides many advantages (proven optimality, short resolution times, easy deployment). It was also shown in previous works that these advantages greatly compensate for the small loss of precision compared to more detailed non-linear models (Ommen et al., 2014; Schütz et al., 2017).

Simple building modelling is a well-researched topic (see section 2). RC modelling starts by defining the structure of the ROM; i.e. the number of elements we wish to represent for a belief that they might have a considerable influence in the desired application. The simulator comes in handy to test the influence of certain elements. In this section, a parametric study performed using the simulator and which has helped setting the ROM structure is described.

We shall first recall that the ROM will be used to apply and assess DSM measures. For instance, during load shifting, we want to rely on this model to optimally plan the heat delivered to the building without jeopardizing the thermal comfort. We expect the real building inertia to delay the mean internal temperature drop, thus offering heat demand flexibility. Therefore, a reliable ROM structure should well predict these delays. In simplified RC models, thermal delays are created by introducing thermal capacitances (C). The simulator will help us determine, per building class, which element in the building is worth being represented by a "C" in the RC model for MPC applications.

To answer this question we considered the 3 building simulators already presented in section 3.6 and Table 2. For each of these buildings we considered 3 levels of internal mass by simply modifying the parameter of the mass density per m² of furniture-equivalent slabs:

- Empty zones with no internal mass.
- Light internal mass of a total of 70 kg/m².
- Heavy internal mass of a total of 140 kg/m².

For each case we considered 3 simulations to assess the influence of the heating circuit inertia:

- A simulation where the heating system model is omitted and heat is directly injected into the zones through the internal air node.
- A simulation with low temperature radiator system having a supply water at 50°C
- A simulation with high temperature radiator system having a supply water at 70°C

Using the thus derived versions of the simulator, we could record the effect of shutting down the heating power to characterize the building time constants, using the following simulation protocol:

1. Reaching steady state conditions, for an internal temperature set-point of 20°C.
2. Cutting out the power supply at the substation.
3. Recording the internal temperature drop, up to a 1°C drop from the set point.
4. Comparing the time constants sensibility obtained for each insulation class to the addition of internal mass and the heating system's inertia.

From this study we could conclude that taking into account both the space-heating circuit and the internal thermal mass were of prime importance for load shedding for all building classes. The obtained detailed results from the simulator can also be compared to the experimental study in (Kensby et al., 2015), in which conclusions could only be reached about the need for considering several time constants.

Therefore, we propose the structure depicted in Figure 6 for MPC applications. The building model features 3 thermal capacitances, 4 thermal resistances and 3 solar gain coefficients. The building model is linear and can be easily derived from the analogical network. The heating circuit model features 1 thermal capacitance to delay the heat delivered at the substation level Φ_{SST} from that injected into the air node Φ_{rad}. It is symbolically represented in Figure 6. Equations of this model include a saturation to limit the substation power to its maximum sizing value. These aspects are rarely modelled in other studies, but the flexibility of Modelica enabled us to model them and assess their importance in a single tool.

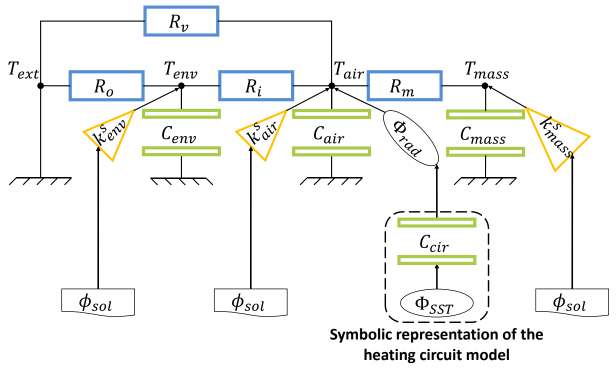


Figure 6. Structure of ROM proposed in light of the parametric study performed using the building simulator.

Table 3. Nomenclature of the ROM elements.

Symbol	Meaning	Subscript	Relative to
T	Temperature [K]	$_{ext}$	The external environment
$\boldsymbol{C}$	Thermal capacitance [J/K]	$_{air}$	The building indoor air
$\boldsymbol{R_v}$	Thermal resistance of ventilation [K/W]	$_{env}$	The building envelope
$\boldsymbol{R_O}$	Thermal resistance between the envelope and the outdoors [K/W]	$_{mass}$	The building internal mass
$\boldsymbol{R_i}$	Thermal resistance between the indoor air and the envelope [K/W]	$_{cir}$	The heating circuit
$\boldsymbol{R_m}$	Thermal resistance between the indoor air and the internal mass [K/W]		
$\boldsymbol{k^s}$	Solar gain coefficient [m²]		
Φ_{SST}	Space-heating power at the substation [W]		
Φ_{rad}	Space-heating power at the radiators [W]		
ϕ_{sol}	Global horizontal solar radiation flux [W/m²]		

4.2 Reduced-order model identification

After the structure definition comes the parameters' identification. This step requires historical data to tune the parameters of the ROM (Figure 6, marked in a bold font in Table 3), with the goal of obtaining an optimal set of parameters that best fits that historical data. In our work, historical data is replaced by data generated by the building simulator, and we restrain it to real-world data accessible to DH operators which is mostly found at the substation level, i.e. outside the building. This means we need to identify the parameters based on the space-heating load power at the substation, and without continuous and intrusive internal temperature measurements.

The detailed building simulator was again of great help for this task, as it enabled us to test several identification methods as well as to verify the results on the internal temperature behaviour (which would not be available in the real world).

The identification method itself is based on the GenOpt optimisation toolkit, and will be described in a future publication. Although more detailed results are out of scope of this paper, we could in particular highlight the following two results:

- The chosen ROM performs better at describing the internal temperature behaviour when identified with only load power at substation, compared to other model structures. Interestingly, most previous work considering identification using internal temperature measurements had different findings, which are not exploitable in our case.
- The correlation between the load power error and the internal temperature error only appears for very low load power errors, meaning that a set of identified parameters may seem to perform correctly when looking at load power error, but may be performing poorly when looking at internal temperature error. A more specific characterization of this result is under study.

4.3 Validation of an optimal space heating controller

Based on the previous steps and on other work, we were able to design an optimal space heating controller. The space-heating controller is designed to act at the district heating substation level.

As explained in section 4.1, this controller is based on a MILP problem formulation, as well as on a receding horizon principle. More specifically, it performs the following operations at regular time intervals (here 15 minutes):

1. Collect data available at substation level (esp. load power), as well as weather and energy cost predictions. The hypothesis here is that energy provision costs are variable over the day, either because of renewable energy usage or by taking into account variations in the global network load leading to various generator use.
2. Formulate a MILP optimization problem, aiming especially at controlling the power injected at substation level while minimizing energy provision costs and over/under-heating inside the building. Part of this MILP problem is obtained from the reduced-order building model, which describes the expected thermal behaviour of the building.
3. Solve the MILP optimization problem over the a given horizon (typically 24h), in order to define the optimal trajectory of the control variable (here the power injected at substation level)
4. Apply the obtained set point for the next time interval (here 15 min), before performing the optimization again to adjust for prediction changes and real system behaviour.

At the validation step, the obtained set points are not applied to the real building, but instead to the building simulator. This particularly enables us to validate that the internal temperature constraints and thermal comfort are well respected.

We used our optimal control tool called Pegase. Pegase is based on the Functional Bloc Simulation Framework (FBSF, 2018) which provides a very efficient C++ co-simulation master fully compatible with the FMI 2.0 standard. Pegase also embeds MILP formulation capacities based on the Eigen linear algebra library (Eigen, 2018), and is integrated with numerous MILP solvers. Using this tool, each iteration step (problem formulation, resolution and building simulation) is performed in a few seconds on a standard PC, using the GLPK open source solver in this case (GLPK, 2018). A real-world deployment of the controller is also available, in which case the building simulator FMU is simply replaced with communication to the real system in place.

Figure 7 presents some first results obtained with the optimal space-heating controller. In Figure 7, the predictive quantities, the outputs of the controller and the results of the detailed building simulator are respectively plotted with dot-dashed lines, dashed lines, and solid lines. From top to bottom, the first 2 graphs present the evolutions of the external temperature, the total solar irradiation and the energy costs respectively. We then present the planned substation power and the internal building mean temperature. The bottom plot shows the evolutions of the supply and return temperatures of the space-heating system. To enable the comparison with a standard control strategy, we also plotted the supply temperature provided by the buildings' heating curve (see section 3.4).

Figure 7 shows that our controller is able to adapt the substation heating power to the actual heating needs of the building. This leads to a decrease of the supply space-heating temperature when solar gains contribute to space heating. Another interesting feature is that the controller is able to decrease the internal building temperature when energy prices are high thereby demonstrating that the space-heating strategy considers a balance between the energy purchase costs and the end-users' thermal comfort.

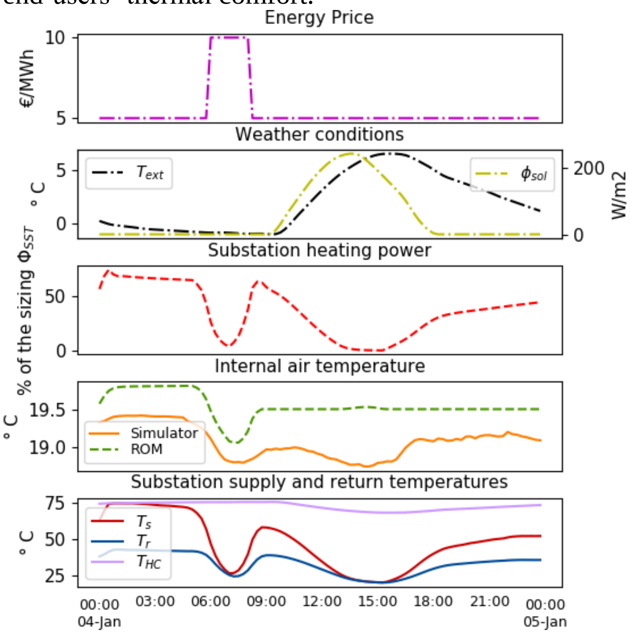


Figure 7. Sample results obtained with the space-heating optimal controller of the model predictive control type.

5 Conclusion

In this paper, we present a Modelica-based building simulator and show how it can efficiently support the development of demand-side management control strategies. Based on the *MixedAir* model available in the *Buildings* library and on component models developed in the *DistrictHeating* Library, we designed a customizable simulator for multi-stories radiator-heated residential buildings, representative of the French district heating sector. By using Modelica as a support language, we were able to model not only the building envelope, but also to other relevant elements such as the internal mass and the radiator heating system, which play an important role when considering load shifting.

We also assessed the numerical performance of this building simulator, especially the parallelization features of the Dymola simulation tool. Although the results show some limitation in the parallelization, we could reach an acceleration factor of 420, meaning that 420 hours can be simulated in 1 hour, in our case using 8 cores in parallel. Such execution speed is well suited for control applications implying simulation periods in the day to week range. A reduction in the number of thermal zones and/or number of storeys would be necessary to perform annual simulations.

Finally, we illustrate how we used the building simulator to design and assess an optimal space-heating controller. In particular, we used the simulator to perform three essential steps: designing a reduced-order building model; setting up a robust identification method for this model; validating the obtained optimal space-heating controller. In all these tasks, being able to define various building models with different parameters, as well as to check the results of the optimal space-heating controller on the simulator were essential.

Acknowledgements

We gratefully acknowledge ADEME (Agence de l'Environnement et de la Maîtrise de l'Énergie) and the City-Zen project for their support and financial contribution in funding the Ph.D. thesis of Nadine Aoun.

References

Antonopoulos, K.A., Koronaki, E.P., 2000. Effect of indoor mass on the time constant and thermal delay of buildings. Int. J. Energy Res. 24, 391–402. https://doi.org/10.1002/(SICI)1099-114X(200004)24:5<391::AID-ER585>3.0.CO;2-L

ASHRAE Standard: Ventilation for Acceptable Indoor Air Quality, 1989. . American Society of Heating, Refrigerating and Air-Conditioning Engineers, Incorporated.

Brun, A., Spitz, C., Wurtz, E., Mora, L., 2009. Behavioural comparison of some predictive tools used in a low-energy building, in: Eleventh International IBPSA Conference. pp. 27–30.

Bruno Peuportier, 2016. Energétique des bâtiments et simulation thermique, Eyrolles. ed.

City-zen, 2018. City-zen Project [WWW Document]. Cityzen-Smartcity. URL http://www.cityzen-smartcity.eu/

Foucquier, A., Brun, A., Faggianelli, G.A., Suard, F., 2013. Effect of wall merging on a simplified building energy model: accuracy vs number of equations, in: 13th International Building Performance Simulation

Association (Building Simulation 2013), Chambéry, France, 25-28 August 2013.

Frayssinet, L., Kuznik, F., Hubert, J.-L., Milliez, M., Roux, J.-J., 2017. Adaptation of building envelope models for energy simulation at district scale. Energy Procedia, CISBAT 2017 International ConferenceFuture Buildings & Districts – Energy Efficiency from Nano to Urban Scale 122, 307–312. https://doi.org/10.1016/j.egypro.2017.07.327

Giraud, L., Bavière, R., Vallée, M., Paulus, C., 2015. Presentation, validation and application of the DistrictHeating Modelica library, in: 11th International Modelica Conference. Presented at the 11th International Modelica Conference, Versailles.

JA Clarke, 2001. Energy Simulation in Building Design.

Johra, H., Heiselberg, P., 2017. Influence of internal thermal mass on the indoor thermal dynamics and integration of phase change materials in furniture for building energy storage: A review. Renew. Sustain. Energy Rev. 69, 19–32. https://doi.org/10.1016/j.rser.2016.11.145

Judkoff, R., Neymark, J., 2013. Twenty years on!: updating the IEA BESTEST building thermal fabric test cases for ASHRAE standard 140. Proc. BS2013.

Kensby, J., Trüschel, A., Dalenbäck, J.-O., 2015. Potential of residential buildings as thermal energy storage in district heating systems – Results from a pilot test. Appl. Energy 137, 773–781. https://doi.org/10.1016/j.apenergy.2014.07.026

Le Dréau, J., Heiselberg, P., 2016. Energy flexibility of residential buildings using short term heat storage in the thermal mass. Energy 111, 991–1002. https://doi.org/10.1016/j.energy.2016.05.076

Lund, H., Möller, B., Mathiesen, B.V., Dyrelund, A., 2010. The role of district heating in future renewable energy systems. Energy 35, 1381–1390. https://doi.org/10.1016/j.energy.2009.11.023

Lund, H., Werner, S., Wiltshire, R., Svendsen, S., Thorsen, J.E., Hvelplund, F., Mathiesen, B.V., 2014. 4th Generation District Heating (4GDH). Energy 68, 1–11. https://doi.org/10.1016/j.energy.2014.02.089

Nageler, P., Koch, A., Mauthner, F., Leusbrock, I., Mach, T., Hochenauer, C., Heimrath, R., 2018. Comparison of dynamic urban building energy models (UBEM): Sigmoid energy signature and physical modelling approach. Energy Build. 179, 333–343. https://doi.org/10.1016/j.enbuild.2018.09.034

Ommen, T., Markussen, W.B., Elmegaard, B., 2014. Comparison of linear, mixed integer and non-linear programming methods in energy system dispatch modelling. Energy 74, 109–118. https://doi.org/10.1016/j.energy.2014.04.023

Paatero, J.V., Lund, P.D., 2006. A model for generating household electricity load profiles. Int. J. Energy Res. 30, 273–290. https://doi.org/10.1002/er.1136

Perez, N., Riederer, P., Inard, C., Partenay, V., 2015. Thermal building modeling adapted to district energy simulation, in: Building Simulation.

Ribault, C., Bouquerel, M., Brun, A., Schumannb, M., Rusaouën, G., Wurtz, E., 2017. Assessing tools relevance for energy simulation at the urban scale: towards decision-support tools for urban design and densification. Energy Procedia, CISBAT 2017 International ConferenceFuture Buildings & Districts – Energy Efficiency from Nano to Urban Scale 122, 871–876. https://doi.org/10.1016/j.egypro.2017.07.452

Richardson, I., Thomson, M., Infield, D., 2008. A high-resolution domestic building occupancy model for energy demand simulations. Energy Build. 40, 1560–1566. https://doi.org/10.1016/j.enbuild.2008.02.006

Richardson, I., Thomson, M., Infield, D., Clifford, C., 2010. Domestic electricity use: A high-resolution energy demand model. Energy Build. 42, 1878–1887. https://doi.org/10.1016/j.enbuild.2010.05.023

Schütz, T., Schiffer, L., Harb, H., Fuchs, M., Müller, D., 2017. Optimal design of energy conversion units and envelopes for residential building retrofits using a comprehensive MILP model. Appl. Energy 185, 1–15. https://doi.org/10.1016/j.apenergy.2016.10.049

Van Schijndel, H., Zmeureanu, R., Stathopoulos, T., 2003. Simulation of air infiltration through revolving doors, in: Eighth International IBPSA Conference, Eindhoven, Netherlands.

Wetter, M., Zuo, W., Nouidui, T.S., 2011. Modeling of heat transfer in rooms in the modelica “buildings” library, in: Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association. Presented at the 12th Conference of International Building Performance Simulation Association Building Simulation 2011, BS 2011, pp. 1096–1103.

Widén, J., Lundh, M., Vassileva, I., Dahlquist, E., Ellegård, K., Wäckelgård, E., 2009. Constructing load profiles for household electricity and hot water from time-use data—Modelling approach and validation. Energy Build. 41, 753–768. https://doi.org/10.1016/j.enbuild.2009.02.013

Wolisz, H., Kull, T.M., Streblow, R., Müller, D., 2015. The effect of furniture and floor covering upon dynamic thermal building simulations. Presented at the Energy Procedia, pp. 2154–2159. https://doi.org/10.1016/j.egypro.2015.11.304

Yao, R., Steemers, K., 2005. A method of formulating energy load profile for domestic buildings in the UK. Energy Build. 37, 663–671. https://doi.org/10.1016/j.enbuild.2004.09.007

Integrated Modelica Model and Model Predictive Control of a Terraced House Using IDEAS

Filip Jorissen[1] Lieve Helsen[1,2]

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

Abstract

Modelica has been used extensively within the Thermal System Simulation (The SySi) research group at KU Leuven to simulate and optimize the control and design of building energy systems. Within this scope, the open source Modelica library IDEAS has been developed and papers have been published that explain how IDEAS can be used to develop fast simulation models and MPC. This paper presents an open-source simulation model of a terraced house for which these earlier presented guidelines are applied and for which MPC results are made available. A full-year simulation of the nine-zones model takes four minutes and energy savings of 12.8 % are reported compared to a current-practice rule-based controller, although MPC has thermal comfort violations of up to 0.4 K.

Keywords: IDEAS, Model Predictive Control, TACO, Building Energy Simulation

1 Introduction

Building space heating and HVAC account for 15 % of the world final energy use (International Energy Agency, 2015). Therefore, according to the European Union's Directive 2010/31/EN (European Parliament, 2010), an increasing effort is spent at increasing buildings efficiency by improving their insulation level, by installing HVAC systems with a high primary energy efficiency (e.g. a combination of heat pump and floor heating or concrete core activation), and by increasing the share of renewable energy sources in buildings. These systems increase the thermal time constants of the building and introduce a rapidly changing collection of devices into the built environment. Efficient design and operation of buildings therefore calls for a dynamic simulation tool that can follow rapidly changing trends. Furthermore, further system integration with district heating systems and electrical distribution networks calls for a tool that is not limited to a fixed set of disciplines. 'Traditional' Building Energy Simulation (BES) tools often only offer a limited set of models and can be difficult to extend. Furthermore, their algorithms often tightly integrate equations and algorithms for solving them into the same code (Wetter et al., 2016; Wetter, 2009; Wetter et al., 2015), which complicates the maintenance of such codes. Furthermore, extracting model equations is usually not supported.

Modelica thus has some clear advantages over these BES tools. Firstly, Modelica is not limited in terms of what models can be integrated. Secondly, Modelica treats the model equations and algorithms for solving them separately. The same set of equations can thus be coupled to different solvers. This includes solvers both for simulation and optimization. Since the model equations are available, they can be differentiated automatically, which enables the use of highly efficient derivative-based optimization algorithms.

For these reasons, the Thermal Systems Simulation (The SySi) research group at KU Leuven has co-developed the open-source Modelica library IDEAS (Jorissen et al., 2018c) together with the Building physics department and the company 3E since 2010. One of the main motivations for our continued development of IDEAS, is to use the building models for optimal design and control applications, where the availability of the model equations poses a crucial advantage. The development of IDEAS is part of the IBPSA project 1, the successor of the IEA EBC Annex 60 (Wetter and van Treeck, 2017), which coordinates the development of IDEAS with the Modelica libraries Buildings (Wetter et al., 2014), AixLib (Müller et al., 2016) and BuildingSystems (Nytsch-Geusen et al., 2013). Our research focusses in particular on Model Predictive Control (MPC). Based on IDEAS, two main white-box approaches for MPC have been developed so far.

Firstly, the model can be implemented using component models from the package `IDEAS.LIDEAS`, which contains component models that support linearisation. This way the building *envelope* can be exported in the state space form

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = Ax(t) + Bu(t). \qquad (1)$$

Where variables $x(t)$ are state variables and variables $u(t)$ are the boundary conditions of the building such as the solar irradiation that enters through each window. Time series data for the vector $u(t)$ is also generated using IDEAS. The model can thus be exported to other frameworks such as python or Matlab. The component models, templates and a minimum working example required for using LIDEAS have recently been made open source and can now be found in `IDEAS.LIDEAS`. For more details with respect to this linearisation toolchain we refer the reader to Picard et al. (2015). This methodology can only be used to export linear models and as such its functionality is limited to linear problems, although non-linear equations can be added to the exported linear model manually.

Secondly, to overcome these hurdles, a non-linear Toolchain for Automated Control and Optimization (TACO) has recently been developed by Jorissen et al. (2018b). TACO interprets a Modelica model using the JModelica framework (Åkesson et al., 2010) and automatically translates the model into an efficient optimization code that is implemented using CasADi (Andersson et al., In Press, 2018). The goal of TACO is to significantly reduce the engineering overhead required for developing MPCs for building applications by leveraging object oriented Modelica models. Jorissen et al. (2018a) describe the detailed Modelica model of an office building and a comparison with measured data. An MPC has been developed for this model by Jorissen (2018). In simulations the operational cost of the building was reduced by more than 50 %, however thermal depletion of the ground by passive cooling was not accounted for.

These applications demonstrate the potential of Modelica and MPC for building applications. Since Modelica is a general-purpose modelling language, multiple problem formulations are possible and multiple solvers can be chosen by the user. Both affect the model computation time and robustness for simulations and optimizations. Our models combine many state variables in the building envelope model with non-linear equations and algebraic loops in the Heating, Ventilation and Air Conditioning (HVAC) models, and discrete equations in the building control. For the simulation of large models this combination can lead to long computation times. Jorissen et al. (2015) and Jorissen et al. (2018d) describe how models can be manipulated to speed up computations. Furthermore, we propose to use explicit time integrators such as Euler integration since the computation time of Euler integration scales better with the number of state variables than for implicit solvers. These tips are applied by Jorissen et al. (2018a) on the model of a 32-zones office building.

These guidelines for implementing computationally efficient building models using Modelica may however be difficult to apply by IDEAS users since a clear example model is not available. Similarly, unexperienced Modelica users may not understand how these integrated building models can be structured. This limits the usability of IDEAS. This paper therefore presents a Modelica example model of a 9 zones terraced house building in Modelica. The guidelines for increasing the simulation speed are applied and explained. Furthermore, the performance of an MPC controller for this model is compared to a rule-based controller (RBC). This comparison can be extended in the future within the scope of the IBPSA project 1, which aims to develop BOPTEST, a set of benchmarks for comparing advanced building controllers such as MPC. Both the model and the optimal control trajectories from the MPC are included in the IDEAS library as of release 2.1 such that they can serve as a case study model or benchmark for other research applications.

This paper is structured as follows. Section 2 describes the building that is modelled and Section 3 explains how this building is modelled using components from the IDEAS library. Section 4 then explains how the guidelines from earlier work are applied to this example model and presents the resulting computation speed. Section 5 compares an MPC based on this model to the RBC implementation that is included with the model. Conclusions are presented in Section 6.

2 Building Description

This paper presents the IDEAS model of a real terraced house that consists of three floors and an attic. The building layout is sketched in Figure 1. The ground floor consists of 1) a living room and a hallway with a combined surface area of 4.6 m x 8 m in the front of the building and 2) a dining room, kitchen and glazed veranda behind the main building, which have a combined ground surface area of 4.6 m x 5 m. The first floor consists of a hallway (1.6 m x 4 m), bathroom (3 m x 4 m) and bedroom (4.6 m x 4 m). The top floor consists of a bedroom (4.6 m x 4 m) underneath a mansard roof, and an office (4.6 m x 4 m) underneath a gabble roof, which contains a Velux window of 1.2 m^2. Both parts of the roof are insulated with 12 cm of polyurethane, 5 cm of glass wool and are finished with plasterboard of 12.5 mm.

The building was constructed around 1926 and its façade consists of uninsulated brick walls without cavity and a thickness of approximately 28 cm. The front 4 m of the building has a cellar, whose ceiling is uninsulated. The ground floor consists of parquet on tiles on solid ground, while the other floors consist of fiberboard or wooden floor board on wooden beams. The ceilings below the floors are finished using plasterboard, which is attached to a metal frame. An acoustic insulation layer of 5 cm lies on top of this frame. The building windows consist of PVC frames with triple glazing. A vertical technical shaft runs through center of the building, which contains the heating system pipes and collectors and the ventilation ducts.

The building is heated using a gas boiler of 30 kW (Bulex Thermomaster T30/35). Radiators supply heat to each of the rooms except the dining room, porch and kitchen. The radiators are connected to a common collector for each floor using alupex tubes. The collectors are connected in series to the boiler. The boiler is on-off controlled using a Bulex Exacontrol E7C thermostat in the living room. A ventilation unit (Brink Renovent Excellent 300 +) supplies air to the bedrooms, office and living room

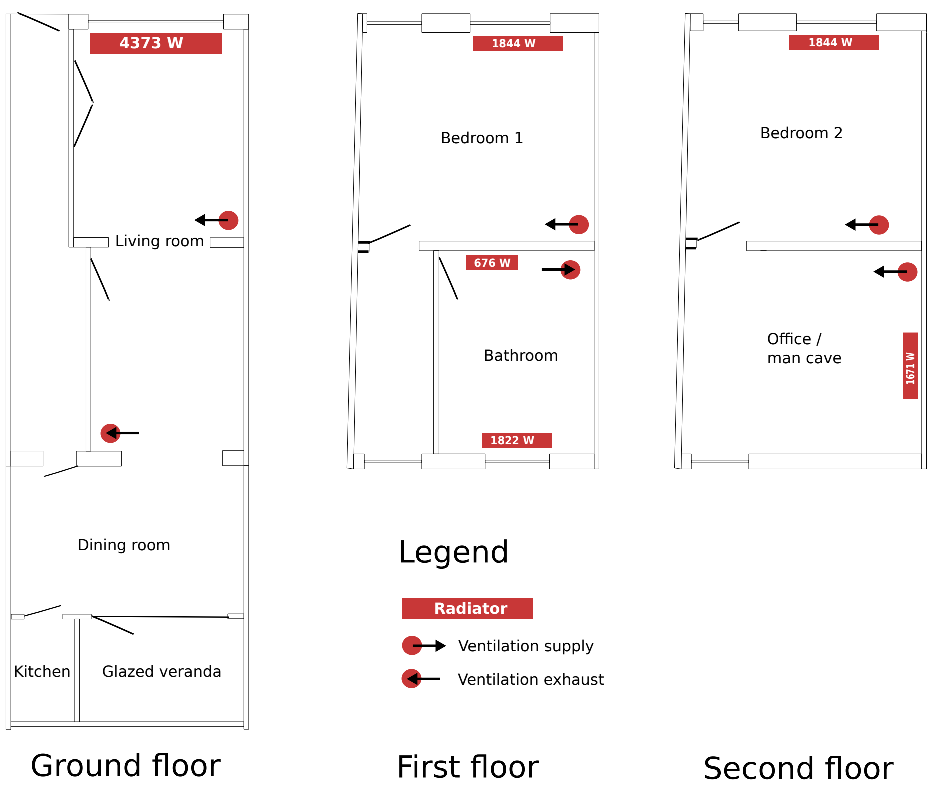


Figure 1. Sketch of the building layout and the location of radiators and vents.

and extracts air in the bathroom and living room. The nominal air flow rate of the unit is 300 m^3/h, for which heat is recovered with an efficiency of approximately 75 %. The heat exchanger is automatically bypassed during summer, which is not modelled yet. The ventilation unit operates at a fixed flow rate of 70 m^3/h.

3 Model Description

An integrated Modelica model of this building is created using the IDEAS library (Jorissen et al., 2018c). The model consists of four parts:

1. The building envelope, which consists of zones, walls and windows.
2. The heating system, which consists of a pump, heater, pipes, radiators and thermostatic valves.
3. The ventilation system, which consists of fans, ducts, two bypass valves and a heat exchanger.
4. The thermostat.

Each of these parts is illustrated in Figure 2. The model is part of the IDEAS library and can be found in the package `IDEAS.Examples.PPD12`.

3.1 Ventilation System

The top of Figure 2 illustrates the ventilation system, which consists of two fans, two bypasses, a heat exchanger and ducts that are connected to the zones. Ducts are modelled using the component `IDEAS.Fluid.FixedResistances.Junction`. The pressure drop of each branch of the junction is estimated from the design flow rate of the system and the diameter and free area fraction of the vents. The pressure drop of the ducts is neglected. This results in nominal pressure drops in the order of 500 Pa for a flow rate of 0.1 kg/s.

The model `IDEAS.Fluid.Movers.FlowControlled_m_flow` from the IBPSA library (Wetter, 2013) prescribes the flow rate of the fans. The total pressure drop is thus computed from the fluid flow network, which is in turn used to compute the fan electrical power use. A fixed fan efficiency of 23.75 % is assumed, which coincides with the measured efficiency at nominal flow rate.

The model `IDEAS.Fluid.HeatExchangers.ConstantEffectiveness` computes how much heat is recovered. It assumes a fixed heat exchange effectivity. Ideal dampers, without pressure drop, are used to model the bypasses.

As indicated in Figure 1, air is injected in some rooms, while air is extracted from other rooms. Manual connections are added to allow air exchange between rooms such that mass is conserved. In the future this functionality will be integrated into IDEAS such that these connections need not be added by the user.

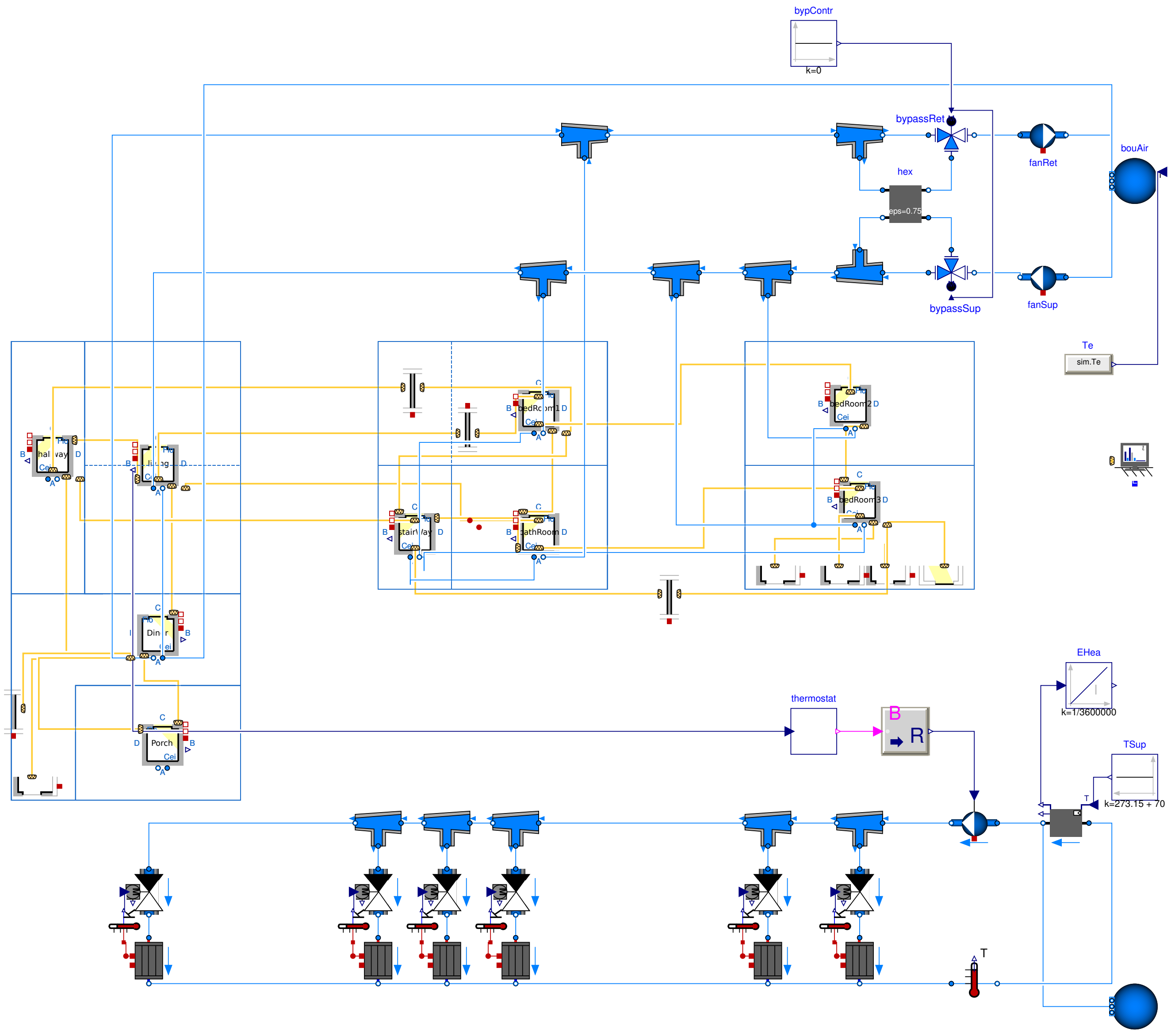


Figure 2. Illustration of the Modelica model of the building. The top includes the ventilation, which consists of two fans, a heat exchanger, ducts and two bypasses. The middle contains the building envelope model, which consists of walls, windows and zone templates, which integrate more walls and windows for a rectangular zone. The bottom shows the heating system, which consists of a heater, pump, pipes, radiators and thermostatic valves. In between middle and bottom the thermostat can be found.

3.2 Building Envelope

The middle of Figure 2 represents the building envelope, which is modelled using components from `IDEAS.Buildings`. The main functionality of these components was described by (Jorissen et al., 2018c). Most notably, the model `IDEAS.Buildings.Components.RectangularZoneTemplate` is used to define the zones, their parameters and interconnections. Interior walls, external walls, roof and windows are included in the zone template for each orientation. When the zone geometry is not simply rectangular, or when a floor has to be split into two parts, additional surfaces are added manually. See e.g. two separate floor models are used to model the floor of bedroom 1 since it is both above the living room and the hallway. The radiators and ventilation system are also connected to the zones but these connections are hidden in Figure 2. The surface (walls, windows, etc.) parameters consist of the surface dimensions, orientations and its structure, which is defined using `records` such as illustrated in listing 1.

Listing 1. Example use of a record to define the material layers of a ceiling

```
record Floor "Ppd12 floor with suspended gypsum ceiling"
  extends IDEAS.Buildings.Data.Interfaces.Construction(
    incLastLay = IDEAS.Types.Tilt.Floor,
    final mats={
      IDEAS.Buildings.Data.Materials.Gypsum(d=0.0125),
      IDEAS.Buildings.Data.Materials.Air(d=0.075),
      IDEAS.Buildings.Data.Insulation.Glasswool(d=0.05),
      IDEAS.Buildings.Data.Materials.Air(d=0.15),
      IDEAS.Buildings.Data.Materials.Timber(d=0.022)});
end Floor;
```

The wall opening between the living room and the dining room is modelled using the recently introduced option in the `InternalWall` model to model a 'cavity' or door. This model approximates the buoyancy-driven advection between zones using the zone temperature differences, the ideal gas law and Bernoulli's principle. We assume that all doors are closed, i.e. they are not modelled using the door model option. Furthermore, we assume that the common walls with the neighbours are adiabatic.

3.3 Heating System

The bottom of Figure 2 illustrates the heating system model. It consists of six radiators, a pump and an ideal heater. Radiator thermal powers are indicated in Figure 1. The pump head is unknown and is assumed to be 100 kPa. The ideal heater model `IDEAS.Fluid.HeatExchangers.PrescribedOutlet` is used, which supplies a prescribed temperature set point unless this implies that a thermal power of more than 30 kW or less than 0 kW is supplied. The boiler efficiency is computed as a function of the return water temperature using the polynomial fit to the data that is provided in a temperature range of 25 °C to 65 °C. The slope at the end of the range is extrapolated up to 75 °C. The resulting efficiency is implemented as in Listing 2.

Listing 2. Efficiency implementation of the boiler

```
Modelica.SIunits.Efficiency eta = {-6.017763e-11, 2.130271e-8,
    -3.058709e-6, 2.266453e-4, -9.048470e-3, 1.805752e-1,
    -4.540036e-1}*{TRet^(6-i) for i in 0:6};
```

Radiators are modelled using the model `RadiatorEN442_2`. Each radiator has a thermostatic valve, which is modelled using the model `TwoWayTRV`, which consists of a valve with a linear opening characteristic that is controlled by a proportional controller with a fixed temperature set point and a proportional band of 2 K. The proportional controller is implemented such that the valve is closed when the temperature set point is reached. The set point equals 30 °C for the radiator in the living room, since the thermostat is mounted in this room. The remaining radiators have a set point of 21 °C. The pressure drop of the pipes is computed using their dimensions.

3.4 Thermostat

The thermostat is a custom implementation, which turns the boiler on or off. Since the supply temperature is not actively controlled, the heater supply water temperature is fixed to 70 °C. The thermostat operates on a schedule. The temperature set point is 21 °C from 7:00 to 9:00 and from 18:00 to 23:00 on week days and from 7:00 to 23:00 on weekend days and 16 °C otherwise. Note however that the thermostat is activated one hour earlier to ensure that the building has heated up in time. Since the thermostat is mounted in the living room, the thermostat uses the living room temperature as an input. Furthermore, it implements a hysteresis controller with a hysteresis band of 1 K above the set point temperature. When enabled, the boiler pump head is set to 1 bar.

4 Computational Aspects

We now discuss some Modelica-related implementation details that affect the computation time.

4.1 Computation Time

As described by Jorissen et al. (2015), implicit integrators are not well suited for simulating models with many states and, depending on the model size, explicit integrators such as Euler integration may be more suited. Our model has about 330 state variables. Therefore, the model parameters have been chosen to avoid small time constants by considering fast processes to be steady state, such that the model can be simulated using (explicit) Euler integration. We now explain for each part of the model how this was done.

4.2 Envelope Model Configuration

Jorissen et al. (2018c) describe in detail how the models in `IDEAS.Buildings` have been adjusted to speed up computations. I.e. fast dynamics in the window glazing are lumped into a single thermal capacitor and non-linear algebraic loops are avoided by choosing an appropriate discretisation scheme of the heat conduction equations.

Furthermore, we configure the massDynamics of the zone models to SteadyState, which implements an incompressible air model, which thus avoids a state variable for the zone air pressure.

4.3 Ventilation System Configuration

In the ventilation system we avoid the generation of algebraic loops that solve for enthalpy by using the portFlowDirection_* variables. Furthermore, the energyDynamics and massDynamics of the junctions, fans and bypasses are set to SteadyState, which avoids more states with a small time constant. Similarly, the input filter of the fans is removed.

4.4 Heating System Configuration

In the heating system, the used water medium is already incompressible by default, such that we need not set massDynamics. The valve filters are not removed since they react already sufficiently slowly. However, the junctions energyDynamics are set to SteadyState. The radiator energy dynamics are not neglected. However, the number of elements of the radiator is reduced from the default value of 5 to 3, such that the radiator segment volumes are sufficiently large to avoid instabilities when the radiator mass flow rate is large. Furthermore, the series pressure drops of the thermostatic valve, the radiator and the pipes are merged into a single pressure drop equation by using the parameter dpFixed_nominal of the valve model. The pressure drop of the pipes between the collectors of each floor are neglected since their internal diameter is 20 mm instead of 12 mm, which causes pressure drops that are 7.7 times smaller for the same mass flow rate, moreover these pipes are shorter. This causes the earlier mentioned series connections to be in parallel with respect to each other, meaning that they have the same pressure drop. This can be exploited by setting from_dp=True in the valve models, since then a single iteration variable can be used to solve the resulting algebraic loop (Jorissen et al., 2018d). For a more detailed discussion and motivation for these configurations and simplifications we refer the reader to Jorissen et al. (2018d) and Jorissen et al. (2015).

4.5 Computation Time Results

Using this configuration, which also speeds up Dassl, the model can be evaluated at a speed 156 000 times faster than real time using explicit Euler integration with a fixed step size of 15 s and 44 000 times faster than real time using Dassl with a tolerance of 10^{-4}.[1] Simulating a full year thus requires 4 minutes. Euler has an error of 0.14 % on the total computed energy use when comparing to LSodar with a tolerance of 10^{-6}, while Dassl has an error of 0.017 %.

[1]On a Macbook pro with 2.7 GHz i7-6820HQ processor, using Dymola 2019 with option Evaluate=true and virtual machine software Parallels 11 running Ubuntu 14.04.

5 Application of Model Predictive Control

While IDEAS was originally designed as a Modelica library for building and district energy *simulation*, the models are now also suited for *optimization* applications such as MPC. Using TACO, the presented model can be translated into an MPC controller by performing only minor modifications such as linearising the building envelope heat transfer equations (Jorissen, 2018, Appendix A).

As a demonstration, an MPC controller was developed that minimizes

$$J(t) = P_{fan,sup}(t) + P_{fan,ret}(t) + 0.25\dot{Q}(t) \qquad (2)$$

where $P_{fan}(t)$ is the electrical power use of the fans and $\dot{Q}(t)$ is the thermal power use of the heater. A weighting factor of 0.25 is used since gas is about four times less expensive than electricity in Belgium. Furthermore, the living room operative temperature is lower bounded to 21 °C during the schedule indicated in Section 3.4. We optimize the heater supply water temperature, which is upper bounded to 75 °C and the fan mass flow rates, which are lower bounded to their set point value of 70 m^3/h and upper bounded to the nominal value of 300 m^3/h. The bypasses are closed and the pump is always enabled. The resulting controller is coupled to the simulation model and is operated for a full year in a closed loop simulation. The optimal control results are stored in a csv file and are included as a benchmark in the model IDEAS.Examples.PPD12.VentilationMPC.

A comparison with RBC (see IDEAS.Examples.PPD12.VentilationRBC), which operates on the same temperature set point and using the same schedule, is shown in Figure 3. The figure shows that MPC is unable to satisfy the comfort constraint, which is caused by model mismatch between the MPC model and the simulation model. For instance, the building envelope convective heat transfer equations are linearized for the MPC, which causes the heat convection coefficients to be overestimated. The average comfort violation of MPC for the living room peaks at about 0.4 K.

MPC often uses much lower supply water temperatures than RBC, of about 45 °C, which is reflected in a smoother heating profile. This is clearly visible in the thermal power of the heater. Furthermore, Figure 5 shows smoother temperature profiles for zones that are controlled using a thermostatic radiator valve, which heat the rooms more than required due to the thermal inertia of the radiators and the zone air temperature. The total energy use of RBC is 6590 kWh while MPC uses only 5749 kWh. This implies a final energy use reduction of 12.8 %. While these are modest energy savings, this is to be expected for badly insulated buildings with simple heating systems,

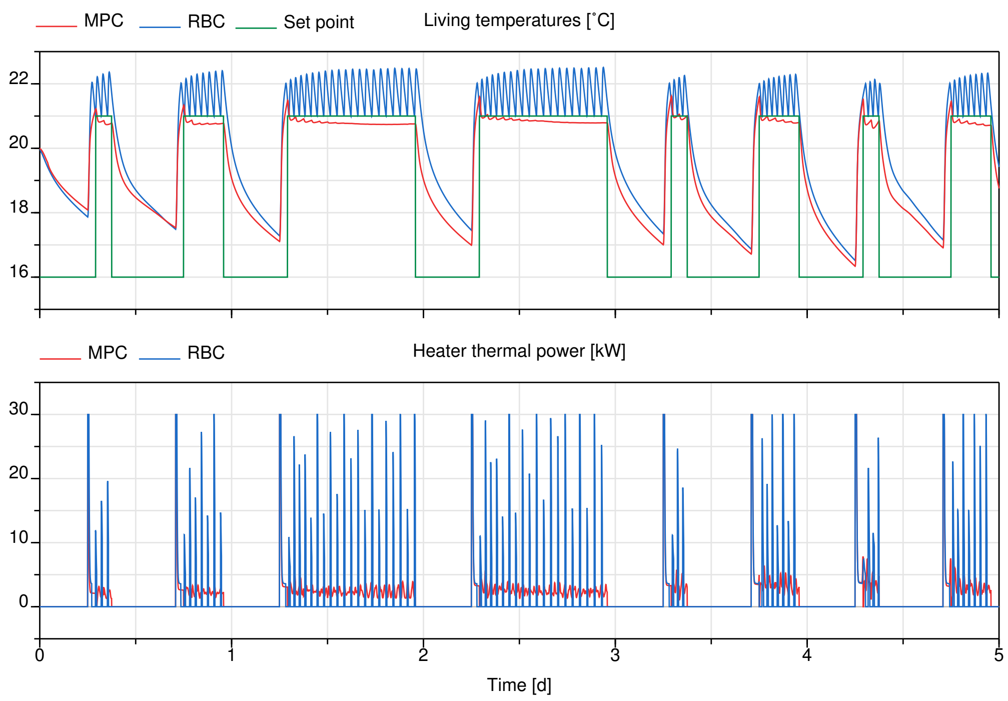


Figure 3. Comparison of zone temperature and thermal power results for RBC and MPC

which have only limited flexibility that can be exploited by MPC. These energy savings are only partly explained by the comfort violations of MPC, since lowering the RBC temperature set point by 1 K for the whole year results in an energy use of 6212 kWh.

The MPC uses control intervals of one hour such that it is forced to start heating one hour in advance in order to satisfy the comfort constraint. If smaller control intervals were used, the pre-heating could be postponed which would lead to larger energy savings.

The model also computes the return water temperature dependent gas energy use. The MPC has a lower supply water temperature so it is unexpected that the average heat generation efficiency of MPC is in fact lower than that of RBC. The gas energy use for MPC is 6317 kWh instead of 7218 kWh (only 12.5 % reduction). This is at least partly caused by the fact that the pulsed behaviour of RBC causes it to have relatively low return water temperatures for the short period of time when the boiler is active, which increases the efficiency. During a winter period, for RBC the return water temperatures rises from about 30 °C to 50 °C, while the MPC has a more constant return water temperature of about 40 °C, as illustrated in Figure 4. Note that the largest heat production for RBC occurs at the lower range of the return water temperatures. This shows the importance of integrated system simulations for making detailed analyses. Note however, that these results may no longer hold when more detailed convection correlations and a more detailed boiler efficiency computation are used.

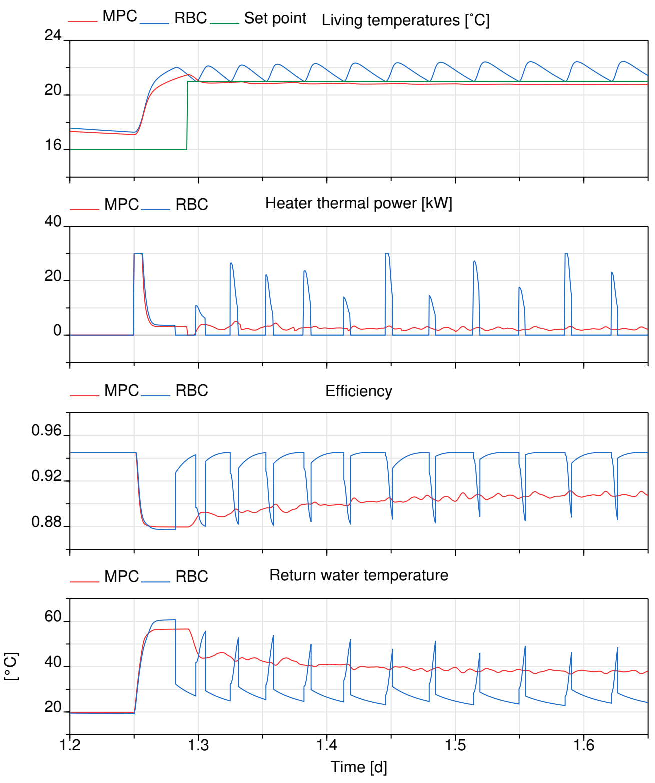


Figure 4. Comparison of heater efficiency for RBC and MPC

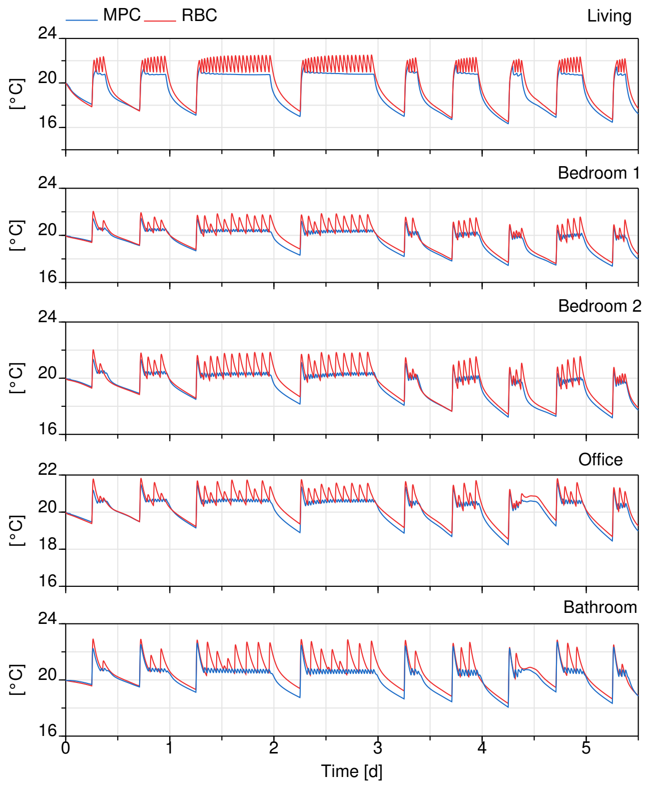


Figure 5. All zone temperatures for RBC and MPC

6 Conclusion

Modelica and open-source libraries such as IDEAS can have large added value in the building energy simulation and optimization community. This paper applies formerly published guidelines for efficient model development using IDEAS and implementation details to reduce computation time to the example model of a terraced house. The example is provided open-source and thus provides Modelica and IDEAS users with a clear example of how to use IDEAS for integrated building energy simulations. The building is first described, after which the model implementation and computational aspects are explained. The model simulation time is 4 minutes for one year. Furthermore, a white-box Model Predictive Controller (MPC) is generated for the model using TACO, a Toolchain for Automated Control and Optimization. MPC uses 12.8 % less energy than rule-based control, which is partly caused by the lower supply water temperatures that generate a smoother heating profile. While these are modest energy savings, larger energy savings are typically obtained for more complex buildings with longer time constants than this uninsulated building with a simple heating system. The model and MPC results are available in the IDEAS library in the package `IDEAS.Examples.PPD12` and can thus serve as a starting point for other research. Future work will focus on validating the model and on expanding the heating system complexity and thus its control options. More complex benchmarks will be developed within the frame of BOPTEST development in IBPSA project 1.

7 Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 723649. The original project acronym is "MPC-.GT".

Co-funded by the Horizon 2020 programme of the European Union

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

J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit. Modeling and optimization with Optimica and JModelica.org – Languages and tools for solving large-scale dynamic optimization problems. *Computers & Chemical Engineering*, 34(11):1737–1749, 2010. doi:10.1016/j.compchemeng.2009.11.011.

J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, In Press, 2018.

European Parliament. Directive 2010/31/EU of the european parliament and of the council of 19 may 2010 on the energy performance of buildings (recast). *Official Journal of the European Union*, 18(06), 2010.

International Energy Agency. World energy outlook 2015. Technical report, 2015.

F. Jorissen. *Toolchain for Optimal Control and Design of Energy Systems in Buildings*. Phd thesis, Arenberg Doctoral School, KU Leuven, April 2018.

F. Jorissen, M. Wetter, and L. Helsen. Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation. In *11th International Modelica Conference*, pages 59–69, Paris, 2015. doi:10.3384/ecp1511859.

F. Jorissen, W. Boydens, and L. Helsen. Implementation and Verification of the Integrated Envelope, HVAC and Controller Model of the Solarwind Office Building in Modelica. *Journal of Building Performance Simulation*, 2018a. doi:10.1080/19401493.2018.1544277. Published on line.

F. Jorissen, W. Boydens, and L. Helsen. TACO, an Automated Toolchain for Model Predictive Control of Building Systems: Implementation and Verification. *Journal of Building Performance Simulation*, 12(2):180–192, 2018b. doi:10.1080/19401493.2018.1498537.

F. Jorissen, G. Reynders, R. Baetens, D. Picard, D. Saelens, and L. Helsen. Implementation and Verification of the IDEAS Building Energy Simulation Library. *Journal* *of Building Performance Simulation*, 11(6):669–688, 2018c. doi:10.1080/19401493.2018.1428361.

F. Jorissen, M. Wetter, and L. Helsen. Simplifications for Hydronic System Models in Modelica. *Journal of Building Performance Simulation*, 11(6):639–654, 2018d. doi:10.1080/19401493.2017.1421263.

D. Müller, M. Lauster, A. Constantin, M. Fuchs, and P. Remmen. AIXLIB – An Open-Source Modelica Library Within the IEA-EBC Annex 60 Framework. In J. Grunewald, C. Felsmann, A. Nicolai, and J. Seifert, editors, *BauSIM*, pages 3–9, Dresden, 2016. Fraunhofer IRB Verlag, Stuttgart.

C. Nytsch-Geusen, J. Huber, M. Ljubijankic, and J. Rädler. Modelica buildingsystems- eine modellbibliothek zur simulation komplexer energietechnischer gebäudesysteme. *Bauphysik*, 35(1):21–29, 2013.

D. Picard, F. Jorissen, and L. Helsen. Methodology for Obtaining Linear State Space Building Energy Simulation Models. In *11th International Modelica Conference*, pages 51–58, Paris, France, 2015. doi:10.3384/ecp1511851.

M. Wetter. Modelica-based Modeling and Simulation to Support Research and Development in Building. *Journal of Building Performance Simulation*, 2(2):143–161, 2009.

M. Wetter. Fan And Pump Model That Has A Unique Solution For Any Boundary Condition And Control Signal. In *13th Conference of International Building Performance Simulation Association*, pages 3505–3512, Chambéry, France, 2013.

M. Wetter and C. van Treeck. *IEA EBC Annex 60: New Generation Computing Tools for Building and Community Energy Systems*. The Regents of the University of California and RWTH Aachen University, 2017. ISBN 978-0-692-89748-5. URL `http://www.iea-annex60.org/pubs.html`.

M. Wetter, W. Zuo, T. S. Nouidui, and X. Pang. Modelica buildings library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014. doi:DOI:10.1080/19401493.2013.765506.

M. Wetter, T. S. Nouidui, D. Lorenzetti, E. A. Lee, and A. Roth. Prototyping the Next Generation EnergyPlus Simulation Engine. In J. Mathur and V. Garg, editors, *14th Conference of International Building Performance Simulation Association*, pages 403–410, Hyderabad, 2015. International Building Performance Simulation Association.

M. Wetter, M. Bonvini, and T. S. Nouidui. Equation-based languages - A new paradigm for building energy modeling, simulation and optimization. *Energy & Buildings*, 117:290–300, 2016. doi:10.1016/j.enbuild.2015.10.017.

An Extended Luenberger Observer for HVAC Application using FMI

Scott A. Bortoff[1] Christopher R. Laughman[1]

[1]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA, {bortoff, laughman}@merl.com

Abstract

In this paper we show how a Functional Mockup Unit (FMU) may be used for the realization of an Extended Luenberger Observer (ELO), which may be considered the deterministic version of an Extended Kalman Filter (EKF). The ELO has advantages over an EKF in some situations, such as lower computational burden and improved convergence. Nonlinear observers, such as those that make use of changes of coordinates to linearize, or approximately linearize the estimate error, are continuous-time dynamical systems that use so-called output injection to modify the dynamics of a model. Output injection provides a similar feedback effect as the correction step of an EKF. However, nonlinear output injection is a slightly FMU different use case because the ELO is a continuous time object. It is realized by feedback around a model-sharing type of continuous time FMU, in contrast with the algorithmic realization of a discrete-time EKF, which uses the co-simulation form of FMU. We illustrate the design and realization of an ELO for a building HVAC example, in which we estimate unmeasured heat flows and unmeasured boundary conditions for use in a building “digital twin.” We also make some remarks about model reduction and the challenges in realizing a conventional EKF for these types of models.

Keywords: Estimation, Buildings, HVAC, FMI, FMU

1 Introduction

State estimation is one of the important use cases for the Functional Mockup Interface (FMI). For example, states of a nonlinear continuous-time model can be estimated from discrete-time measurements of the input and output of a plant using a continuous-discrete Extended Kalman Filter (EKF), realized using the co-simulation form of a Functional Mockup Unit (FMU) of the plant (Brembeck et al., 2014, 2011). Fundamentally, the EKF, and its various extensions estimate the state in a two-step process. In the prediction step, the EKF computes the predicted state estimate using a discretized plant model. Then in the correction step, the covariance and gain are computed as a function of the predicted state estimate, and the predicted state estimate state is corrected. The discrete-time prediction model is then initialized using the corrected state, and the process is repeated. Importantly, the two steps are coupled in a causal manner: The prediction step at time $(k+1)$ depends only upon the correction step at time k, and the correction step at time k depends only on the prediction step at time k. This fact allows an FMU to be used in an algorithm to estimate the state in the prediction step, since it can be initialized using the corrected state estimate from the previous correction step.

An *observer* is an alternative technology for estimation of the plant states and parameters. An observer is a deterministic, continuous-time dynamical system that takes as input the measured input and measured output of the plant, and produces as its output an estimate of the state of the plant. It is similar to the Kalman filter, but based on deterministic assumptions and mathematics. Fundamentally, the concept of *output injection* is used to stabilize the observer error dynamics, which govern the difference between the estimated state and the plant state. Output Injection means that a signal is injected (added) to the derivative of the observer state vector as stabilizing feedback. Because of this, it is the continuous-time dynamics of the plant *with* output injection that needs to be simulated. There are not separate prediction and correction steps.

In this paper we show how an instantiation of a model-exchange type of FMU can be used with the Dymola tool to realize output injection, enabling design and implementation of linear and nonlinear state observers and specifically the Extended Luenberger Observer (ELO). Our specific interest is to estimate unmeasured performance variables of a building and HVAC system as a part of a building “digital twin.” Toward this end we have considered several alternative methods to estimate the performance variables, including various flavors of the EKF. However, these may prove too computationally burdensome for our application because the number of states can be large (hundreds), the number of measurements can be large (tens to hundreds), and the EKF can be computationally challenging because of the covariance update, although there are many techniques such as model reduction and square root filtering that are available to improve its computational efficiency. More importantly, an EKF can fail to converge, or in some cases, cause the model to fail at run time, at least for our building HVAC applications. Convergence failures are caused by some of the characteristics of the model that we consider in this paper, which are not unusual for this field of application. The model is stiff (with time constants ranging from milliseconds to several weeks — eight orders of magnitude), and is numerically ill-conditioned (with states varying 8-9 orders of magnitude because of the choice of units). Thus the Jacobian may not accurately predict the state over the fixed and usually large EKF sample time, causing it to diverge. Moreover, the model itself contains state constraints, such as a non-negative limit on mass concentrations, which can be violated at run time because of the EKF correction step, causing a run-time error.

On the other hand, the ELO is relatively simple and light-weight computationally. In its simplest form, it uses a constant feedback gain matrix that is computed at design time from the steady-state solution of a Ricatti equation, and therefore avoids the real-time covariance update and computation of the system Jacobian that is necessary for the EKF. Further, it may offer improved stability and performance advantages over the EKF (and similar filters) for certain applications because it makes use of implicit variable-step solvers for the continuous-time model.

This paper is organized as follows. In Section 2, we review the basics of the Extended Luenberger Observer. In Section 3, we construct an ELO for a case-study building and HVAC system and show some simulation results. We show how the FMU is used to allow for the output injection. Finally in Section 4 we conclude by making some observations on potential improvements of FMI to better enable realization of estimators of different types.

2 Background

Following (Zeitz, 1987), consider the nonlinear system

$$\dot{x} = f(x,u,d) \tag{1a}$$
$$y = h(x) \tag{1b}$$
$$z = g(x) \tag{1c}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the control input, assumed measured, $d \in \mathbb{R}^q$ is a disturbance measurement, assumed measured, $y \in \mathbb{R}^r$ is the measured output, and $z \in \mathbb{R}^p$ is the performance output, assumed unmeasured. Our objective is to estimate the performance output z. The Extended Luenberger Observer is the system

$$\dot{\hat{x}} = f(\hat{x},u,d) + K(y-\hat{y}) \tag{2a}$$
$$\hat{y} = h(\hat{x}) \tag{2b}$$
$$\hat{z} = g(\hat{x}) \tag{2c}$$

where $\hat{x} \in \mathbb{R}^n$ is the state estimate, $\hat{z} \in \mathbb{R}^q$ is the performance output estimate, and K is the observer gain. System (2) is a copy of the original system, with the vector $K(y-\hat{y})$, which is called *output injection*, added to the state equations.

The state estimate error $\tilde{x} = x - \hat{x}$ is then governed by the system

$$\dot{\tilde{x}} = f(x,u,d) - f(\hat{x},u,d) - K(y-\hat{y}) \tag{3a}$$
$$\tilde{y} = h(x) - h(\hat{x}) \tag{3b}$$
$$\tilde{z} = g(x) - g(\hat{x}). \tag{3c}$$

We linearize (3) about an equilibrium $\bar{x}$ in a neighborhood of x, defining

$$F = \frac{\partial f}{\partial x}|_{x=\bar{x}}\ , \quad H = \frac{\partial h}{\partial x}|_{x=\bar{x}}\ , \quad \text{and} \quad G = \frac{\partial g}{\partial x}|_{x=\bar{x}}\ , \tag{4}$$

so that the linearized error dynamics, neglecting higher-order terms, are

$$\dot{\tilde{x}} = (F-KH)\tilde{x} \tag{5a}$$
$$\tilde{y} = H\tilde{x} \tag{5b}$$
$$\tilde{z} = G\tilde{x} \tag{5c}$$

There exists an observer gain K to make the origin of (5a) locally exponentially stable if the pair (F,H) is detectable.

There are many methods for the design of the observer gain K e.g. (Luenberger, 1971; Chen, 1984; Friedland, 1986). In fact, more generally we can consider nonlinear changes of state coordinates $z = \Phi(x,u,d)$, nonlinear changes of the output coordinates $\xi = \Gamma(y)$, and nonlinear output injection $K(y)$ as in (Krener and Isidori, 1983; Krenner and Respondek, 1985; Hou and Pugh, 1999). Research on methods for computing these remains an active area of research e,g, (Boutat et al., 2009; Tami et al., 2013). Here we will simply linearize the system (1) about an equilibrium and compute the gain K that minimizes the quadratic cost

$$J = \min \int_0^\infty \tilde{z}^T Q \tilde{z} + \tilde{y}^T R \tilde{y}\, d\tau \tag{6}$$

by solving the steady-state Algebraic Riccati Equation

$$0 = AP + PA^T - PH^T R^{-1} H^T P + \Phi^T Q \Phi, \tag{7}$$

from which the observer gain is $K = (R^{-1}HP)^T$.

3 Building "Digital Twin" Case Study

In this section we design an ELO to estimate unmeasured performance outputs in a commercial building HVAC system. The primary purpose of the observer is to estimate heat flows through the walls, ceiling and floor, and also to estimate the unmeasured heat loads, denoted q, in the occupied space. These estimates can be used to better understand building performance and improve human comfort and energy efficiency.

The building, diagrammed in Figure 1, is the top floor of a medium-sized commercial office building, with open floor plan for office work. We model the floor as a single room with four outside walls, a floor and a ceiling. Above the ceiling is a small plenum space that separates the ceiling from the roof. The walls are made up of between one and four layers of building materials. Windows are on the South and West facing facades. The air conditioning system is a chilled water plant, with fan coils for cooling. Outside air ventilation is provided by a constant speed ventilation fan, and the outside air passes through an Energy Recovery Ventilation Unit (ERV) for pre-cooling in

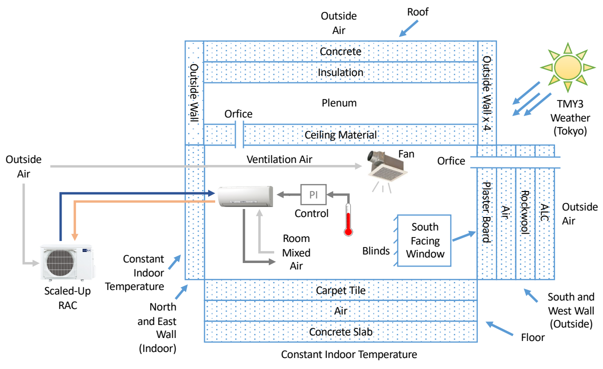


Figure 1. Building with plenum.

the summer season, but is otherwise not treated. For purposes of design, we assume there are three measurements available on a one minute sampling interval: The room temperature T_r, the plenum temperature T_p, and the return water temperature T_w. We also assume that the weather variables are measured hourly. These include the outside air temperature, humidity, wind speed, direct and indirect solar radiation in visible and infra red radiation, cloud conditions, and the atmospheric pressure. The room temperature T_r is compared to a reference set-point, and the error is fed back through Proportional-Integral (PI) feedback to actuate the valve in the fan coil.

The system is modeled using the Modelica buildings library (Wetter et al., 2014) as two rooms: one representing the working space, and the second representing the plenum, as shown in Figure 2. The outside walls have four layers, and the windows are double-paned glass. Orfices are put between the plenum and room to represent airflow between them, although its velocity is very close to zero nominally. A cooling coil is connected to a variable speed chilled water pump to provide variable capacity cooling. An Energy Recovery Ventilator (ERV) is included to precool the outside ventilation air, which is provided at a fixed rate. All of the model components are taken from the Modelica buildings library. Typical Meteorological Year (TMY) weather for Tokyo is used in all simulations. The complete model has 85 states, three measured outputs, one input (the water pump speed), and eleven disturbance inputs corresponding to the eleven weather variables used in the building library. A PID controller from the Modelica Standard Library is added to the model later for feedback to regulate the room temperature to a desired set-point.

We now step through the design and implementation steps, beginning with model augmentation, which is done in order to estimate unmeasured model inputs, then model linearization, order reduction, feedback gain design, and FMU realization.

3.1 Model Augmentation

After constructing the nominal model, it must be modified for use as an estimator. Normally the heat load q is considered an *input* to the model. (Actually, there are three different types of heat load: Radiative, Sensible and Latent. Here we assume all of the heat load is sensible.) However, in order to *estimate* q from the available measured outputs, we augment the model to include q as a state. We assume that the heat load is constant, and then add the equation

$$\dot{q} = 0 \tag{8}$$

to the Modelica model. This is done by adding an integrator to the model as the heat load, with its input set to zero. This will allow us to estimate the heat load with zero steady-state error if it is constant, and a small tracking error if it is time-varying.

Mathematically, the building and HVAC model is

$$\dot{x} = f(x,u,d,q) \tag{9a}$$
$$\dot{q} = 0 \tag{9b}$$
$$y = h(x) \tag{9c}$$
$$z = g(x) \tag{9d}$$

where z is the heat flow through the surfaces of interest (floor, walls, ceiling, and window), y is the three measurements, x is the 85-dimensional state vector, d represents the measured weather inputs into the model, and u is the water valve control input. The model used for estimator design does not include the PI feedback controller, which is added later for simulations.

3.2 Linearization

We then simulate model for approximately one million seconds (about 1 week). This is necessary because the slowest observable mode in the model has a time constant of approximately eight hours, which comes from the concrete building materials in the walls. For the linearization, we zero the radiative effects of the weather, and assume the outdoor temperature and humidity are constants representing typical weather in the summer. This is not ideal, since the radiative effects are dominant. However, it is effective for this particular application. The linearization is

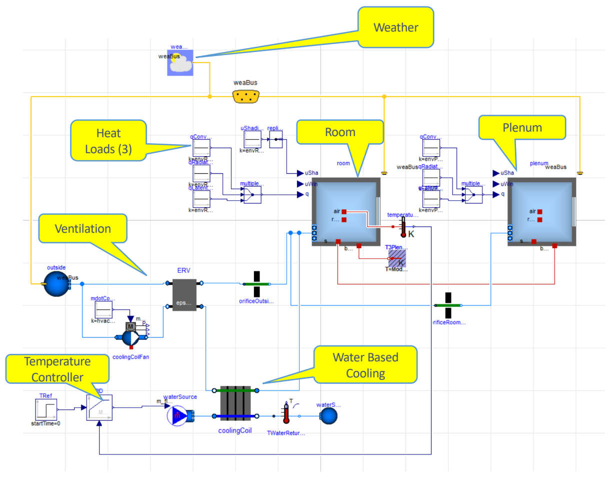


Figure 2. Modelica model.

represented as

$$\dot{x} = Ax + Bu \tag{10a}$$
$$y = Cx \tag{10b}$$

3.3 Observer Gain Design

We design the observer gain $K \in \mathbb{R}^{86\times 3}$ as outlined in the previous section, with a penalty $Q \in \mathbb{R}^{86\times 68}$ on the estimated states, and $R \in \mathbb{R}^{3\times 3}$ penalizing the measurements. For simplicity, these are set to be diagonal matrices. However, we find that a solution to the Riccati equation (7) for the linearized model and any such values of Q and R does not exist! We must analyze the linearized model (10), and then modify and reduce it in order to properly design the feedback gain K.

Computing the spectrum of A, we find a total of three states have eigenvalues at exactly zero, one state has an eigenvalue at almost zero, but corresponding to a time constant of several *months*, and the remainder have real negative parts with time constants ranging from 12 ms to 7 hours, as expected. (It may surprise the reader to see such fast modes in a model of an HVAC system. These are due to heat flow in the metal heat exchanger.) One of the three zero eigenvalues corresponds to the integrator, which can be verified by computing the left eigenvalues of A and showing that the integrator state corresponds exactly with the corresponding left eigenvector. (This means that the integrator state is affected by none of the other states, but it does affect other states, and is, in fact, observable.) The other two states with exactly zero eigenvalue correspond to "physical" states that are introduced into the orfice equations in the model, which can be seen by inspecting the following code taken from the Modelica buildings library.

```
Real mExc(quantity="Mass", final unit="kg")
    "Air mass exchanged (for purpose of
       error control only)";
initial equation
  mExc=0;
equation
  if forceErrorControlOnFlow then
    der(mExc) = port_a.m_flow;
  else
    der(mExc) = 0;
  end if;
```

We see that the state `mExc` is introduced for error control, and has its derivative set to zero if `forceErrorControlOnFlow=false`. This state has no effect on a simulation, but it is included in the linearization. Inspection of the corresponding rows of B and C verify that this state is neither controllable nor observable, and is obviously not stable. Its presence in the model therefore causes the Riccati equation solver to fail. We therefore symbolically remove the two states `mExc`, corresponding to the two offices in our model, from the linearization by removing the corresponding rows and columns. Note that this is not a numerical calculation.

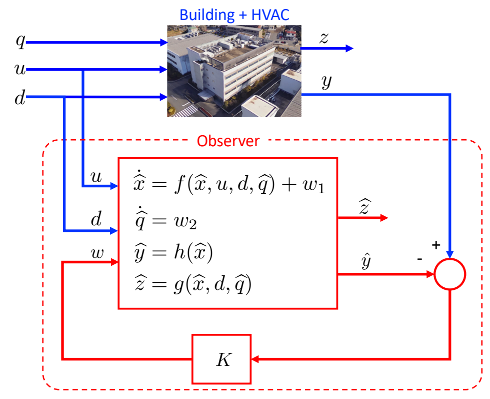


Figure 3. Observer block diagram.

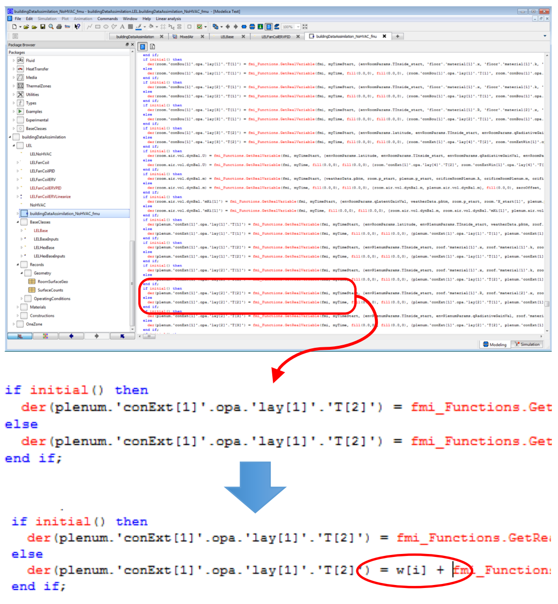
Then in the estimator, we simply initialize these states at zero and they are effectively ignored.

The other eigenvalue near zero has an eigenvector that is nearly aligned with the potential energy state of the plenum air. However it is not an exact alignment, so we cannot say that the physical state is exactly this slow state. Its presence in the model causes the Riccati solver to fail for some values of Q and R. We therefore remove it from the linear model by modal decomposition, resulting in an 83-dimensional reduced model, which is detectable from our three measurements (because it is exponentially stable). This reduced model is used to design a reduced-order feedback gain K_r, and the full order gain is computed by using a value of zero for the three states that were removed and expanding back to the original 86-dimensional system.

3.4 FMU Realization

A block diagram of the observer is shown in Figure 3. This shows the structure of the inputs and outputs to the observer. It takes as input the control input u, the measured disturbances d, and the output injection vector w, which is the feedback signal $K(y-\hat{y})$. The output injection vector w is added to the dynamic equations. This diagram shows the augmented state to include the unmeasured heat loads q.

An FMU makes realization of the observer possible, because it is essentially a DLL for the right-hand side of the ordinary differential equation, and once loaded into a tool like Dymola, can be manipulated to allow for the output injection. Figure 4 shows the Modelica model that adds the output injection vector w to the right-hand side of the differential equation that is defined by the FMU. Essentially we declare the real input vector w and add each component to the lines that define the `der( · )`. We have created Python scripts to automate the process of editing the Modelica file. We then instantiate the modified FMU, wrap the feedback gain around it, and declare inputs and outputs to drive the new model with data. Note that the order of the states in the linearization is often different than the order of states in the FMU. So as a practical matter,



```
if initial() then
  der(plenum.'conExt[1]'.opa.'lay[1]'.'T[2]') = fmi_Functions.Get
else
  der(plenum.'conExt[1]'.opa.'lay[1]'.'T[2]') = fmi_Functions.Get
end if;
```

```
if initial() then
  der(plenum.'conExt[1]'.opa.'lay[1]'.'T[2]') = fmi_Functions.GetRe
else
  der(plenum.'conExt[1]'.opa.'lay[1]'.'T[2]') = w[i] + fmi_Function
end if;
```

Figure 4. Modification of FMI in Dymola.

we typically re-order the states of the linearization so that it corresponds to that in the FMU.

3.5 Simulation Results

To test the observer, we first simulate it using data generated from the original model. For both systems, we design a PI feedback controller to regulate the room temperature. We then simulate the data-generating model for Tokyo weather during the last week of June. We drive this model with an “actual" heat load as an input, assumed to be zero until 8:00am when the workday starts and it ramps up continuously to 4kW over one hour. (Of course, the observer *estimates* this value.). We sample the weather hourly, and the three temperature measurements on a one minute clock, which is the typical sampling rate for these applications. We then apply this data to the modified FMU, which also includes the same feedback controller.

Some of the results are shown in Figure 5 and 6. In Figure 5 we see that the ambient, plenum and water return temperatures have good information content, while the regulated room temperature remains relatively constant and therefore provides little information to the observer. The plot also shows the estimated heat flows. The flow through the ceiling is dominant, while that through the south and west walls is relatively small. Heat flow through the west wall is larger in the early evening, due to solar radiation. The heat flow through the ceiling peaks about six hours after the solar radiation peak, because of the large amount of heat storage in the concrete above the plenum. The plot at bottom shows the estimated and “actual” heat load. The observer is able to estimate the heat load with little lag, and with zero steady-state error as expected. Figure 6 shows a close-up of the estimated and actual heat loads. The observer is able to estimate the heat load with some small lag and zero steady-state accuracy when the actual load achieves its constant value at 9:00am.

4 Conclusions

In this work we have used FMU to realize an Extended Luenberger Observer for a building HVAC application. The approach is an alternative to an Extended Kalman Filter, and may offer some advantage in some applications, such as improved convergence and reduced computational complexity. The observer is constructed by augmenting the model dynamics to allow for estimation of boundary conditions, which is the heat load input to the model, linearizing, reducing and designing a feedback gain to stabilize the observer error dynamics, and then realizing the feedback using output injection by modifying the FMU. Some initial simulation results are provided as a simple proof of concept.

There are several extensions to this work and we expect to publish alternative formulations and experimental validation in the future. The most obvious is to compare the performance to an Extended Kalman Filter and its variants. The design of the EKF is made possible by features of FMI that allow for computation of the system Jacobian, starting and time stepping of the model, and setting of the model initial conditions which is done in the correction step.

To date we have experienced quite a few challenges with the EKF for this application. First, we find that the correction step, which modifies the state, can push the model outside its domain of validity. Often the states are corrected in a manner that causes a state to violate one of its limits. Mass fractions of water are particularly troublesome. Although we might consider using dry air models, the performance of the HVAC system is strongly affected by humidity, and neglecting this physics is not desirable. Is it possible to derive Modelica models that extend regions of validity, into perhaps non-physical domains? Modelers should think about this possibility, since the models themselves are useful for things beyond forward time-domain simulations. Of course, it may be possible to modify the EKF itself, preventing the correction step from violating constraints. Indeed, a key reason to consider Moving Horizon Estimators is that the constraints in the model may be enforced.

A second difficulty we have experienced with the EKF is divergence, which may be caused by the stiffness and poor conditioning of the model itself. We find that often the very slow states can be perturbed in the correction step, causing very slow convergence or simply poor performance. It may be possible to avoid some of this by projection or resetting some of the states, although some of the states of interest, e.g. some heat flows, depend on the slow dynamics in the model. On the other hand, the ELO seems more robust. This may be because it is using the implicit variable-step DASSL solver.

We remark that a more thorough analysis of the slow modes in these models is necessary. Often their presence in a linearized model can cause conventional Hankel-norm model truncation to fail. This is because these modes are very slow, with eigenvalues very close to zero. The Hankel-norm truncation begins by computing a spectral decomposition, and only removes those modes with sufficiently small Hankel singular value, and that are sufficiently stable i.e., have a sufficiently negative eigenvalue. Such a truncation will keep these slow modes in the model, even if they are very weakly controllable and observable. Therefore, they must be removed from the linearization before the Hankel-norm truncation is done. Although these modes can apparently be removed in a spectral decomposition of the linearization at design time, there is no guarantee that the resulting reduced order model will result in a correct estimator or controller design, and the modes are still present in the simulation model. There are open questions such as how these should be initialized in an estimator. The precise cause of these slow modes needs further investigation.

References

D. Boutat, A. Benali, H. Hammouri, and K. Busawon. New algorithm for observer error linearization with a diffeomorphism on the outputs. *Automatica*, 45(10):2187–2193, 2009.

Jonathan Brembeck, Martin Otter, and Dirk Zimmer. Nonlinear observers based on the functional mockup interface with applications to electric vehicles. In *Proceedings of the 8th Modelica Conference*, pages 474–483, 2011.

Jonathan Brembeck, Andreas Pfeiffer, Michael Fleps-Dezasse, Martin Otter, Karl Wernersson, and Hilding Elmqvist. Nonlinear state estimation with an extended FMI 2.0 co-simulation interface. In *Proceedings of the 10th International Modelica Conference*, pages 53–62, 2014.

Chi-Tsong Chen. *Linear System Theory and Design*. Holt, Rinehart and Winston, 1984.

Bernard Friedland. *Control System Design: An Introduction to State-Space Methods*. McGraw-Hill, 1986.

M. Hou and A. Pugh. Observer with linear error dynamics for nonlinear and multi-output systems. *Systems & Control Letters*, 37(1):1–9, 1999.

A. Krener and A. Isidori. Linearization by output injection and nonlinear observers. *Systems & Control Letters*, 3(1):47–52, 1983.

A. Krenner and W. Respondek. Nonlinear observers with linearizable error dynamics. *SIAM Journal on Control and Optimization*, 23(2):197–216, 1985.

D. Luenberger. An introduction to observers. *IEEE Transactions of Automatic Control*, 16(6):596–602, 1971.

Sigurd Skogestad and Ian Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. Wiley, 2005.

R. Tami, D. Boutat, and G. Zheng. Extended output depending normal form. *Automatica*, 49(7):2192–2198, 2013.

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

M. Zeitz. The extended luenberger observer for nonlinear systems. *Systems & Control Letters*, 9(2), 1987.

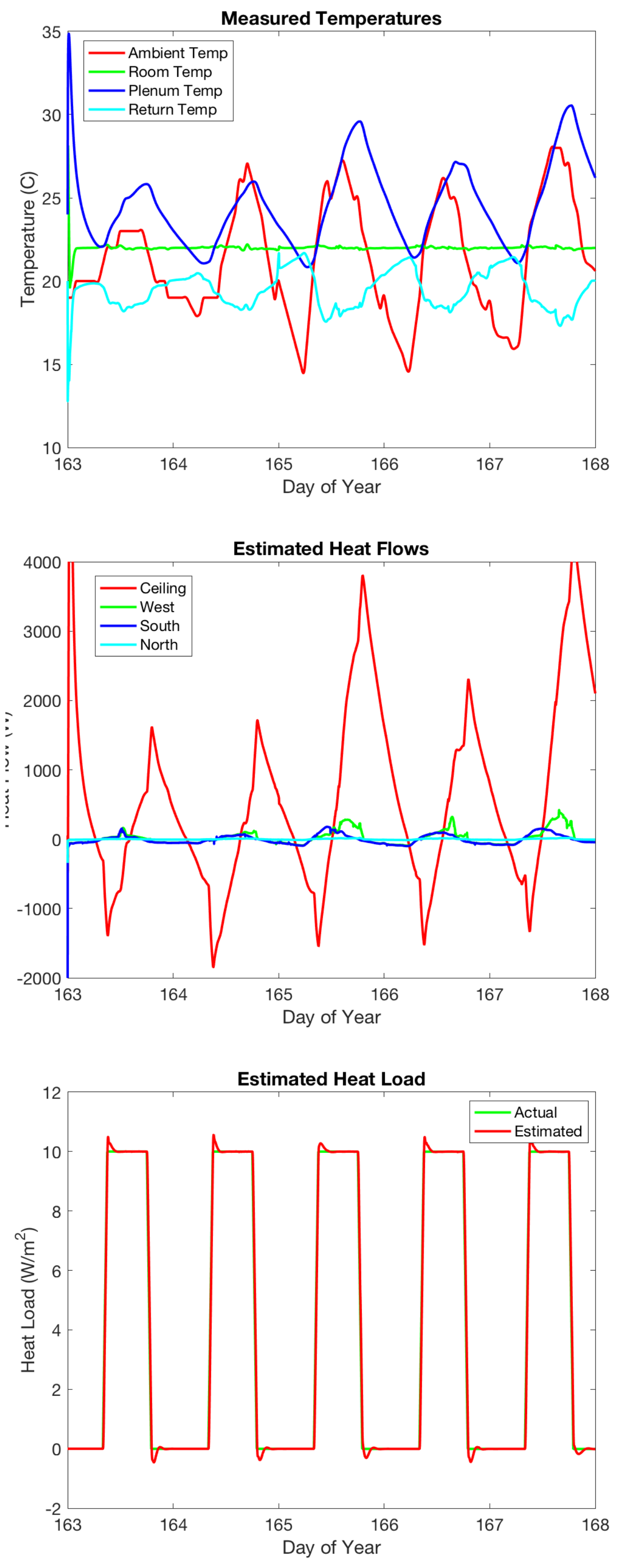


Figure 5. Simulation Results.

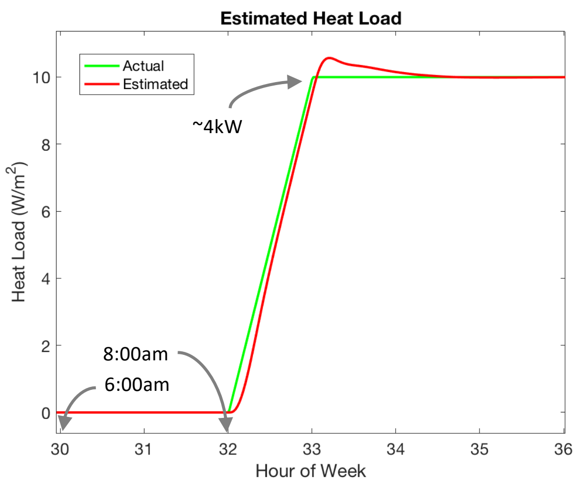


Figure 6. Close-up of the heat load estimation.

Session 2B: Power & Energy 2

A Modelica-Based Framework for District Heating Grid Simulation

Dipl.-Ing. Torsten Schwan[1] Dipl.-Ing.-cand. Ole Ziessler[1] Dipl.-Ing. Tom Eckhardt[1]
Dipl.-Ing. René Unger[1]

[1]EA Systems Dresden GmbH, Germany, {torsten.schwan, ole.ziessler, tom.eckhardt, rene.unger}@ea-energie.de

Abstract

The interdisciplinary modelling language Modelica is increasingly used in the design and evaluation of energy systems. Heat supply represents a considerable share of the global energy supply. Especially in European cities, district heating grids are often used and implemented for heat coverage. The increasing integration of renewable energies and the extension of existing grids require engineers to be able to analyze and evaluate the behavior of such grids, not only statically in certain operating conditions, but also dynamically to enable the representation of complex system interaction.

This paper shows and describes a new approach as to how Modelica models can be used to evaluate the dynamic behavior of district heating grids. It furthermore introduces a consistent framework to parameterize these models with GIS-data via the COM interface. The advantages of the shown approach compared to previously used static methods are shown with specific case studies.

Keywords: district heating grid, renewable energies, heat supply, GIS-data integration

1 Introduction

The development and planning of energy systems represents an increasing challenge for engineers. On the one hand, the goal of decarbonization in energy supply requires an increasing integration of volatile renewable energies. This volatility requires the integration of additional storage systems, ultimately resulting in the introduction of additional degrees of freedom and thus complexity, in power plant control.

On the other hand, the distribution of energy between production and consumers must be adapted to increasing decentralization and partial changes of exergy levels. This applies distinctly to district heating grids, which are particularly complex and widespread. The lowering of temperature levels within these grids leads to significantly reduced distribution losses. This temperature reduction also enables the integration of alternative heat sources such as solar thermal, which are not dependent on conventional combustion-based heat production.

For engineers, the challenge is both the design and evaluation of the central heating plants and the grid itself. The methods used up to present for this purpose have primarily included static calculations, of the system behavior for specific operating points. With this method the maximum load in winter is given priority and partial load cases are only considered subordinately. However, in order to be able to compare the system behavior with regard to energy efficiency, operating costs and ecological footprint, it is particularly important to consider these partial load cases. In addition, the integration of condition-based systems, such as storage, require a dynamic analysis approach rather than a static operating point analysis, based on system equilibrium.

Static grid calculation tools, such as STANET or BENTLEY, combine a clear, GIS-based grid representation with a calculation of the behavior of individual grid components, such as pipes, branches and house connections, based on extensive databases. These tools enable the calculation of temperature and pressure behavior in different grid areas for specific operating points on the basis of an iterative calculation approach. They also enable a clear grid and result representation on the basis of map data which facilitate an easy-to-understand result evaluation and interpretation. However, these tools do not have dynamic modeling capabilities.

Pressure and temperature are scalar physical states. This makes the use of the versatile modelling language Modelica ideal for adding dynamic considerations to existing calculation approaches. The approach of modelling district heating networks on the basis of Modelica has already been discussed in several research studies.

Soons et.al. 2014 implemented a complex district heating grid model including heat production, distribution and thermal building models with reduced complexity, based on Modelica, of a renewable energy building campus. The study considered temperature losses within the grid as well as resultant pressure drops depending on pipe friction. Schwan, et.al. 2014 implemented a Modelica model of a small rural town center with complex building models, district heating grid piping and renewable heat production based on the

Green Building library models. Hägg 2016 adapted available pipe models of the Modelica Standard Library (MSL) by replacing the finite volume method with a spatial distribution operator approach. Schweiger, et.al. 2017 implemented a Python-based framework to automatically generate a district heating grid model based on Modelon's Thermal Power Library. The master thesis of Hermansson et.al. 2017 describes a framework using Matlab to automate the data processing, modelling and simulation of Modelica district heating models.

All these approaches and studies already address a wide range of required toolsets and simulation models which engineers require to analyze and evaluate district heating grids in a dynamic way. The models and methods are mainly based on research work at universities and associated companies. However, planners and engineers involved in the practical implementation of such district heating networks require a uniform toolset based on standard planning tools and databases as well as a uniform presentation of the results of planning-specific parameters. The approach presented here enables the automated transfer of data from standardized district heating network simulation tools, such as STANET and BENTLEY, using standard MS Office products and the COM interface. In this way the often 5,000+ parameters for the modelling of a grid can easily be gathered, transferred and written into the model. In addition, the results obtained from the model can be fed back into the existing evaluation procedures.

2 Modelling Approach

Simulation models of hydraulic grids are comparatively easy to implement compared to complex power plant systems. The number of necessary model components is manageable. However, the size and complexity of a heating grid requires an overall model with a multitude of equal model components which are each defined by a multitude of parameters. This aspect characterizes the actual challenge in dynamic modelling.

Within this work a Modelica-based library of grid components has been implemented with the following six components:

- Pipeline
- Heating plant
- Grid node
- House connection station
- Pipe branch
- Pipe junction

The pipeline is the most important element of such a library. This model component describes the heat loss over the insulation ($\dot{Q}_{loss}$) in the flow and return pipes of individual grid sections as described in equation 1:

$$\dot{Q}_{loss} = \Delta Q_{insulation} \cdot l_{pipe} \cdot (T_{med} - T_{ground}) \quad (1)$$

The model also calculates the individual pressure drop (Δp) of a pipe depending on pipe roughness (2), pipe bends (3) as well as other fittings (4). Furthermore, it identifies pressure losses due to geodetic elevation differences (5).

$$\Delta p_{rough} = \lambda_{pipe} \cdot l_{pipe} \cdot \rho_{med} \cdot \frac{v_{pipe}^2}{2 \cdot d_{i,pipe}} \quad (2)$$

$$\Delta p_{bends} = \zeta_{pipe} \cdot l_{pipe} \cdot \rho_{med} \cdot \frac{v_{pipe}^2}{2} \quad (3)$$

$$\Delta p_{fittings} = \sum \Delta p_{element} \quad (4)$$

$$\Delta p_{geo} = \rho_{med} \cdot g \cdot \Delta z \quad (5)$$

The sum of these elements characterizes the total pressure drop within a pipeline. The gradient dependent, absolute pressure losses of a pipe are almost compensated between flow and return (i.e. only the difference between temperature-specific densities of the fluidic medium (ρ_{med}), cause a pressure drop).

The main pressure loss in a pipeline is dependent of the pipe friction (i.e. 90% plus in horizontal pipes). This pipe friction is highly reliant on the type of stream, i.e. laminar or turbulent in a smooth or rough pipe. To identify the stream type, the pipe friction coefficient (λ_{pipe}) is calculated based on Reynolds number (Re), includes the dynamic viscosity coefficient (η_{med}) as well as the roughness coefficient (k_{pipe}) and the pipe diameter ($d_{i,pipe}$).

$$Re = \frac{v_{pipe} \cdot d_{i,pipe} \cdot \rho_{med}}{\eta_{med}} \quad (6)$$

The Reynolds number is thus used as an indicator for the stream type. If the Reynolds number is smaller than 2,300, the stream type is defined as laminar.

$$\lambda_{pipe,lam} = \frac{64}{Re} \quad (7)$$

A Reynolds number between 2,300 and 100,000 indicates a turbulent stream type for a smooth pipe.

$$\lambda_{pipe,turb,smooth} = \frac{0.3164}{\sqrt[4]{Re}} \quad (8)$$

Any higher Reynolds number than 100,000 represents turbulent stream for a rough pipe.

$$\lambda_{pipe,turb,rough} = \frac{1}{(2 \cdot log_{10} 3.71 \cdot \frac{d_{i,pipe}}{k_{pipe}})^2} \quad (9)$$

All other model components of the library are less complex. They are mainly implemented with reduced complexity, to contribute to a suitable simulation performance such as in the case of large grids with multiple house connection stations and complex pipelines.

The house connection station model is defined through an inverse model which calculates required volume flow dependent on simulated flow temperature and the associated return temperature (T_{Return}).

$$T_{Return} = \min(T_{Return,max}, T_{Flow} - \frac{Q_{Heat\ Consumption}}{c_{p,med} \cdot \rho_{med} \cdot q_{v,max}}) \quad (10)$$

The maximum return temperature ($T_{Return,max}$) is a system specific parameter, which depends on heating surface configurations and even more importantly, on hot water supply system type. Additionally, the return temperature is determined by the maximum volume flow of the considered house connection station. If the total heat consumption exceeds the defined maximum level, the maximum volume limit further decreases resultant return temperature.

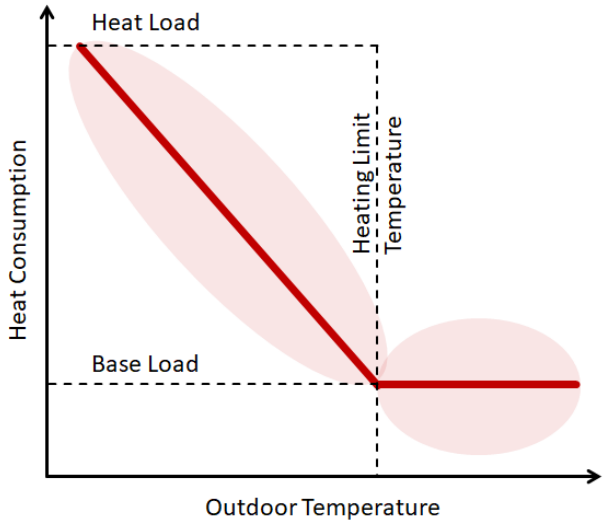


Figure 1: Concept of the heat consumption calculation in the House Connection Station model

The dynamic volume flow is only influenced by the temperature difference between flow and resultant return temperature as well as simulated heat consumption. To provide suitable simulation performance, the heat consumption is not calculated with a complex multi-zone building model but by a look-up representation of overall heat consumption of a building, dependent on outdoor temperatures (c.f. Figure 1).

This approach is especially feasible for residential or office buildings and building complexes as well as similar occupancy types. These building types do not include high internal heat loads nor major solar heat gains (if the window share is not higher than roughly 25%). Below the heating limit, the heat consumption is thus, mainly linearly dependent on the outdoor temperature. Above this limit, heat consumption is comparatively constant (i.e. base load) and is mainly influenced by occupancy specific heat consumption (i.e. hot water consumption).

The developed modelling approach describes the grid behavior in an inverse direction. It highly simplifies the simulation of individual buildings' thermal behavior by only using 10 parameters. The main results of a grid simulation include pressure and temperature behavior in different grid parts as well as the total required heat supply and pressure drop in the considered heating plant. This heating plant model provides an outdoor temperature dependent flow temperature (i.e. heating curve) as well as the maintenance return pressure. Further possible calculations include the total heat supply dependent on temperature difference, volume flow and total pressure drop dependent on resultant flow pressure. These values represent the most important dimensioning variables of a heating plant.

A district heating network can be constructed as a radial, ring or mesh system. Radial networks represent the simplest form of a network in which a large main pipeline feeds several distribution pipelines, forming individual branches. In these branches, the pressure of the main pipe is distributed homogeneously over all distribution pipes. The resultant flow pressure is again inversely calculated by the maximum pressure drop of all distribution pipes. The flow temperature for all branches corresponds to the main pipe and the return temperature is calculated using the mixing ratio of the distribution pipes.

$$p_{return,main} = p_{return,branch1} = p_{return,branch2} \quad (11)$$

$$p_{flow,main} = \max(p_{flow,branch1}, p_{flow,branch2}) \quad (12)$$

$$T_{flow,main} = T_{flow,branch1} = T_{flow,branch2} \quad (13)$$

$$T_{return,main} = \frac{q_{v,branch1} \cdot T_{return,branch1} + q_{v,branch2} \cdot T_{return,branch2}}{q_{v,branch1} + q_{v,branch2}} \quad (14)$$

Modelling ring or mesh systems requires additional components (i.e. pipe junction) which calculate the volume flow distribution between two pipes of a junction, depending on the pressure drop (c.f. equation 15).

$$q_{v,junction1} = q_{v,main} \cdot \frac{\Delta p_{junction2}}{(\Delta p_{junction1} + \Delta p_{junction2})^2} \quad (15)$$

A grid node is also added which is used to identify a specific point of the grid between two parts of one pipe (e.g. in case of a diameter reduction).

Furthermore, district heating grids can include two or more heating plants at different grid positions. This case highly increases the model complexity as it is not possible to implement such a structure with a complete inverse modelling approach.

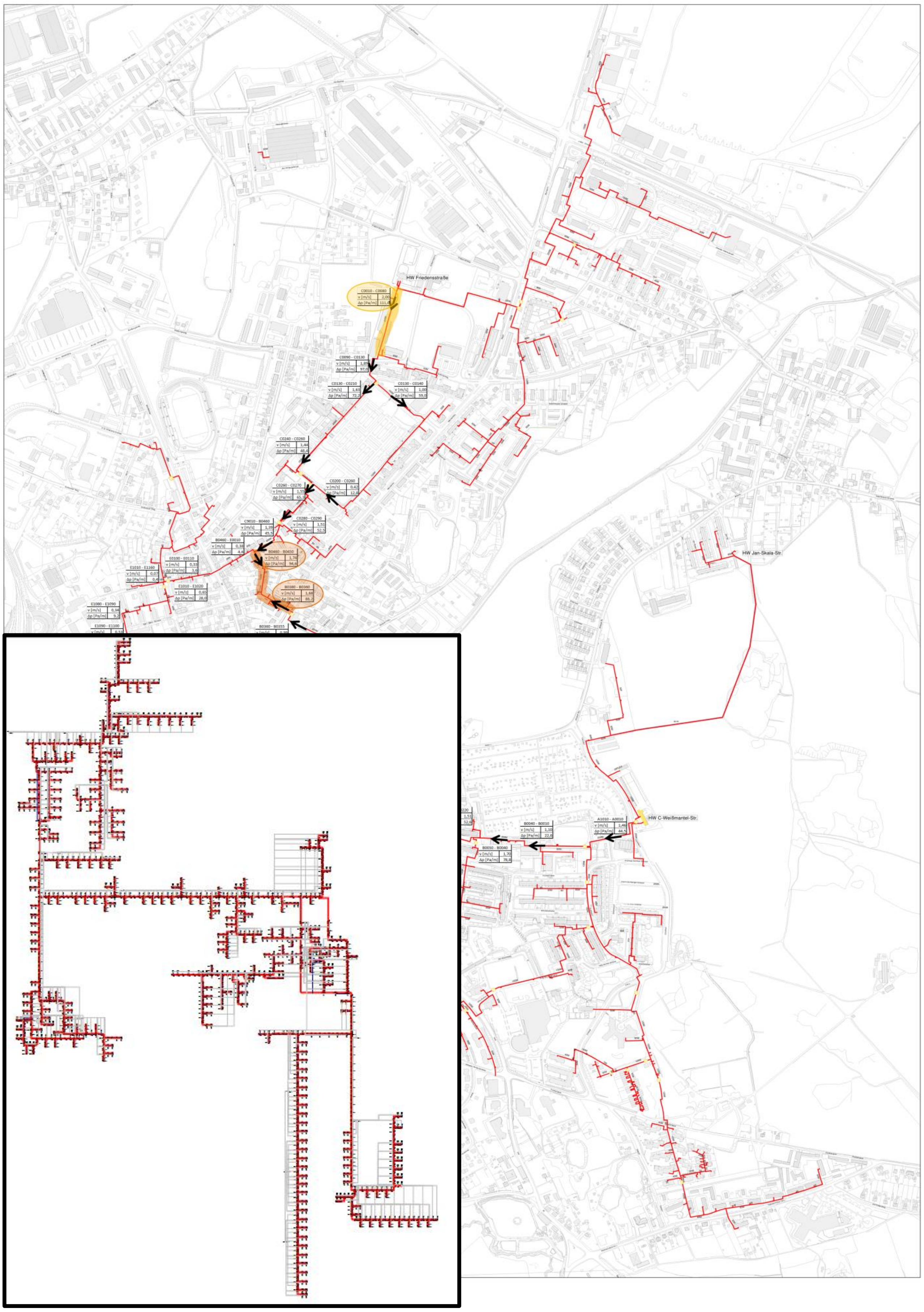


Figure 2: Example grid structure and corresponding Modelica simulaton model as well as sample result representation

The main challenge in these models is to identify the dynamic movement of the grid point at which volume flow reversal takes place, especially if both heating plants work with different operation strategies (e.g. basic heat supply, residual heat supply).

The first approach to solving this modelling problem considered the application of the Navier-Stokes equation for the development of a non-inverse pipe model.

However, first implementations with this methodology showed significant disadvantages regarding simulation performance. The alternative approach implements pipe components (i.e. flow and return) with two inverse directions. The model is able to dynamically detect reverse flow and switch grid calculation between both pipe elements. This shows that to achieve maximum simulation performance more complex modelling techniques and methods are necessary. However, this disadvantage can be compensated using script-based partly automated modelling frameworks.

3 Modelling Framework

The library components described above show the implemented approach for modelling the behavior of a district heating grid in the Modelica modelling language. However, in this area of application, the main challenges faced are the parameterization of the model and the evaluation and presentation of the results. A typical district heating grid with a total length of approx. 50 km and 250 house connection stations requires the processing of 12,000 plus parameters as well as the evaluation of more than 1,000 grid components.

Most parameters of district heating grids today are already available in electronic form such as in data bases of GIS-based but static district heating grid simulation models (e.g. BENTLEY, STANET, etc.). These data bases can most often be easily exported to common data formats like *.csv or *.txt and therefore be imported in MS Excel.

Furthermore, SimulationX, the Modelica simulation environment used to implement the above described library components, provides a script-based (via Python or VBA) access to the simulation model via the MS COM interface. Imported grid parameters can thus be used to automatically parameterize the implemented district heating-grids, simulation model. Even the model structure itself (i.e. components' position, orientation and connections) can be implemented via script using model internal annotations. Therefore, the model structure represents real-world grid layout and available GIS data (i.e. x and y coordinates) provide sufficient information for automatic modelling.

Besides automated variant analyzes, SimulationX furthermore enables an automatic export of simulation results via the same COM interface (Neidhold et.al. 2018). Therefore, a set of suitable MS Excel templates and evaluation scripts provide an easy to use framework to integrate Modelica simulation models as well as common MS Office tools in a consistent workflow for district heating grid analyzes.

4 Simulation Examples

One complex simulation example is a medium-size district heating grid in a town in eastern Germany. It has a total pipeline length of approximately 50 km. The total installed heating power output is 20 MW which is divided by about 40% to 60% between a base load and residual load heating plant (c.f. Figure 2).

The grid only has to overcome slight geodetic differences in height of about 40 m. A maintenance pressure of an estimated 5 bar is thus provided by the residual heating plant.

The grid is currently under reconstruction. It previously consisted of two main individual grids, each with their own heating plant, supplying heat to the north western and south eastern part of the city. Both former grids will now be connected to one complex district heating grid with a base and a residual load heating plant. Furthermore, a small local heating grid in a peripheral residential area will be connected to benefit from the increased heat supply efficiency of the new grid with modern cogeneration power plants.

Finally, new customers of the district heating grid shall be acquired. Therefore, additional pipelines to peripheral buildings (e.g. a large school complex) will be built in the south west and north east area.

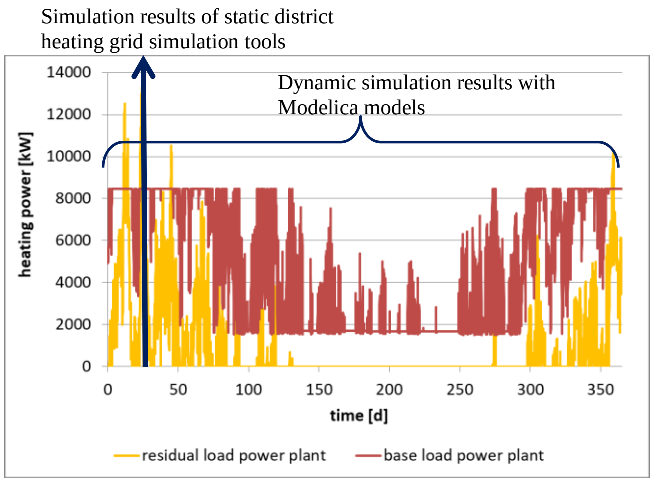


Figure 3: Simulated heat power output in both heating power plants in a reference year – Differences between static and dynamic grid simulation

To effectively plan the reconstruction it is necessary to analyze the resultant requirements on total heat supply as well as pressure drops in both heating plants with regards to the developed operation strategy. Additionally, an evaluation of all relevant grid areas in regards to maximum flow pressure as well as heat and

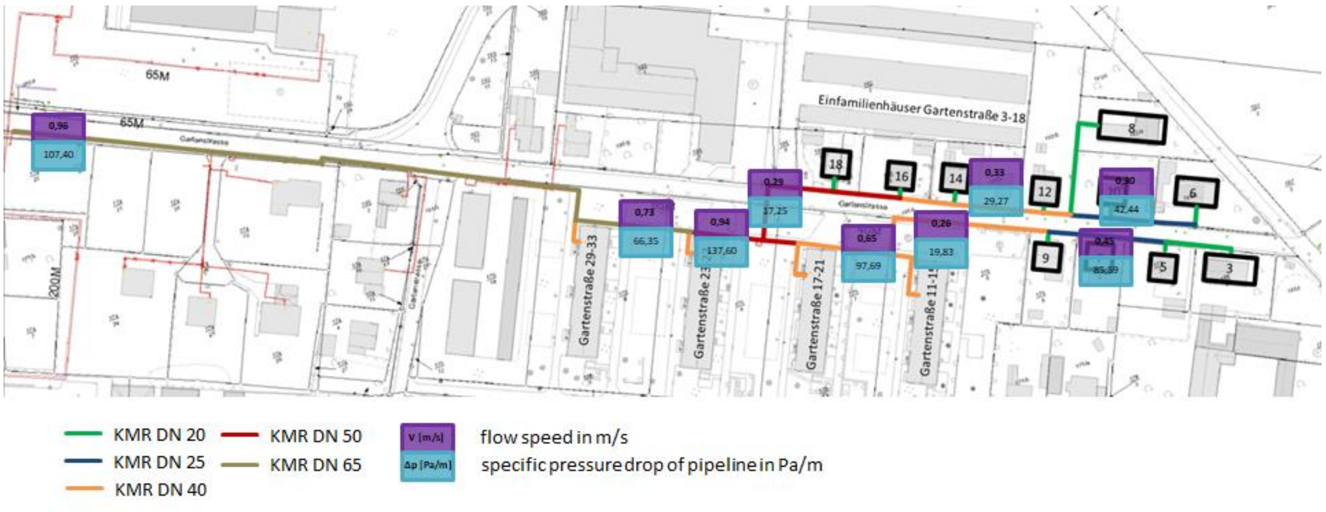


Figure 4: Example simulation results – maximum flow speed and specific pressure drop in one of the grid parts

pressure losses in varying pipelines, is necessary. This is ultimately required to confirm that the planned grid operation with the two power plants fits the requirements of all customers and all weather conditions. Furthermore, the simulation results are used in the sizing process to determine the right dimension of circulation pumps, cogeneration units as well as peak-power boilers.

Additionally, the simulation results characterize the remaining grid capacity of all grid parts which can be utilized (i.e. provide access to additional customers). They also indicate pros and cons of different piping solutions for additional grid parts.

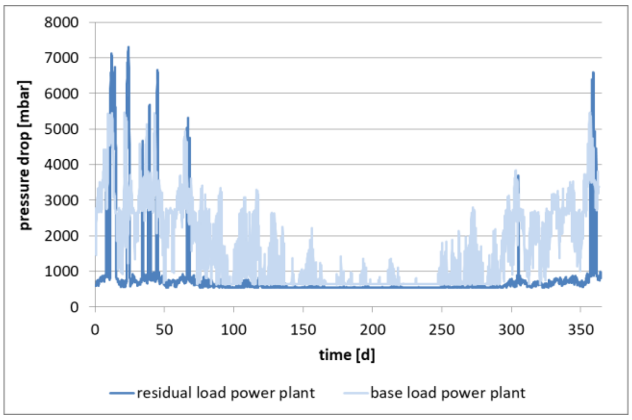


Figure 5: Simulated pressure drop in both heating power plants in a reference year

Figure 3 depicts one of the previously described result sets from the district heating grid simulation model, showing the dynamic heat power output of both the base load and the residual load power plant. It illustrates that most heat over the year is provided by the base load power plant (heat power output with a maximum of 9MW can be supplied for almost half of the year). The peak-power output of the residual heat power plant however, exceeds the maximum of the base load with about 13 MW at the end of January. This significant difference between peak load and base load (about 2 MW) results mainly from the connected residential buildings which only consume little amounts of heat to produce domestic hot water, during the summer time (base-load periods).

Figure 5 shows the corresponding dynamic pressure drop behavior in both heating power plants mainly dependent of outdoor-temperature specific, heat-power output. In times of shutdown the pressure drop at the residual power plant remains at the minimum level which is provided from the base load power plant at its grid position. During these times, the complete flow is provided by the circulation pumps of the base load power plant. In the case that peak-heat power is required, the pressure drop in the residual load power plant significantly increases (up to 7 bar), often even above the base load heat power plant pressure parameters (max. about 5.5 bar). In this case, most of the grid's customers are supplied by the residual load power plant.

Figure 4 furthermore shows a section of the grid in the north eastern area which shall be extended with additional pipes to supply further single-family houses. The main question regarding the maximum grid capacity in this area, that is posed, is if all considered buildings can be additionally connected to the grid without exceeding maximum capacity.

The simulation results showed that the maximum specific pressure drop in the considered pipelines as well as the maximum flow speed does not overrun pipe-specific limits (i.e. 1 m/s flow speed and 150 Pa/m specific pressure drop). Thus the model could confirm that the existing grid capacity is sufficient, to additionally connect all remaining single-family homes in the considered street.

5 Heating Power Plant Models

The above described approach and methods enable the dynamic simulation of district heating grids which allows engineers to better evaluate and plan such grids. This results in more detailed and accurate grid parametrizations and information.

Using this information gain and utilizing the Modelica modelling language further aspects of district heating can be examined and developed. One of these areas is the development of detailed heating power plant models to test and evaluate unit commitment algorithms and methods.

Due to the global goal of decarbonization cogeneration units have been promoted as a middle-term solution to decentralizing energy production. These units combine power and heat production. This improves overall process efficiency and thus such units are common in district heating grids. The produced electricity is directly marketed on the stock exchange, meaning that prices vary on a quarterly hour bases (prices are released for the next 24 hours). Therefore, through utilizing heat storage capacities, the trade with production flexibility offers new economic incentives for heating power plant operators.

Operators are required to plan their power production for the next 24h (Day-Ahead Planning). To automate and optimize this planning process, unit commitment algorithms have been developed which take into account fluctuating electricity prices and future heat demands. These algorithms can mostly only be tested using simplified static verification methods. This can be problematic as simplifications are often made in the algorithm development process which cannot be tested or evaluated using these static methods. The following chapter will describe an approach that enables a dynamic simulation and evaluation of such algorithms using the Modelica-based library Green City.

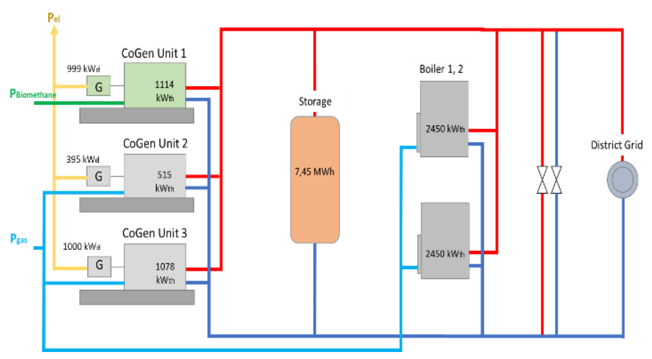


Figure 6: Example Heating Power Plant

Green City is a newly developed simulation library in ESI ITI's SimulationX for holistic modeling of heating, cooling and electric power supply, storage systems and consumption models in buildings and city quarters. It is based on the existing Green Building approach used to simulate sophisticated HVAC systems including renewables, storages, control strategy and eMobility.

To enable the above described dynamic simulation a model-based test platform was developed on the basis of an example heating power plant. The simplified example plant consists of 3 different cogeneration units, one heat storage system, 2 peak-heat boilers and a district grid (summarized as one thermal load).

The challenge of building the above example plant was that not all of the components needed where available in the standard Modelica libraries, like Green City in SimulationX. Since the cogeneration units needed to be controlled through external *.txt files (via a time-dependent reference power output curve) that were written by the unit commitment algorithms, an interface as well as a new unit controller was developed, that enabled the coupling of the planning algorithm and the model. Another Interface was established that enabled the link between the heat prediction algorithms (temperature dependent) and the district heating grid.

The cogeneration units were further developed to be able to adjust running cycles so that minimal unit modulations were upheld. Furthermore, the boilers control technology was adapted so that a peak-heat operating mode was possible. Utilizing these new library components a complete model based platform could be developed.

To examine the above developed model and test its validity different test scenarios were defined and implemented. The first looked at plausibility and the second at sensitivity to prediction errors. These test scenarios were simulated using three daily case examples which are defined through different heating periods (i.e. summer, winter, transition period). Furthermore 2 algorithm types were used to create different unit commitment plans for the above described example days. One algorithm applied only a heat-controlled operation (baseline algorithm) and the other implementing an electricity revenue optimization. This allowed for an overall validity evaluation of the described model and the investigation of the possibility of algorithm assessment.

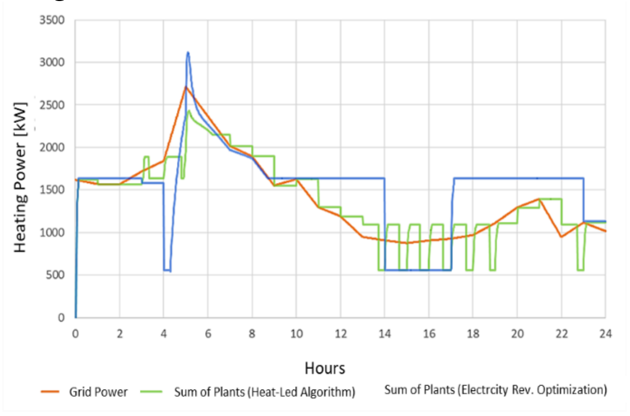


Figure 7: Example of Simulation Assessment

The following figure shows an example of such a plausibility investigation. It is visible that technological processes such as running cycle adjustment and peak power compensation through the boiler are correctly implemented within the model. Furthermore, the incorporation of the heating storage is visible in the flexible relocation of heat production.

The described test scenario evaluations showed that the Modelica based model was able to plausibly depict the system technology of a heating power plant including cogeneration units, taking into account physical and technological constraints. The developed simulation model thus, enables a dynamical and transient validation of unit commitment planning algorithms. The study shows that both the plausibility and sensitivity of such algorithms can be investigated using the developed model based test platform. This ultimately enables the optimization and evaluation of such algorithms before real world implementation. It also showed the advantages of dynamic investigations, using the Modelica modelling language, opposed to mere static approaches.

6 Conclusion

The presented simulation approach enables engineers and scientists to simulate thermal and hydraulic behavior of a district heating grid. Apposed to conventional GIS-based grid simulation tools, the developed approach using the versatile modelling language Modelica enables the consideration of weather dependent dynamic effects as well as storage capacity influences, over a year, within one model. This inevitably enables engineers to better evaluate part load conditions.

GIS-based grid simulation tools however, provide copious data bases of required system elements (e.g. pipes of different sizes and manufacturers) as well as an easy-to-understand frontend to illustrate simulation results with graphical references to individual grid parts on the map. To close this gap, the existing COM-Interface between the Modelica simulation environment SimulationX and MS Excel was extended to automatically build and parameterize gird models utilizing the imported grid data base. Furthermore, the interface was also used to implement post processing routines for result evaluation and graphic presentation.

The approach has been tested with sufficient measurement data of several example grids. The results are valid for district heating grids of small to medium size.

As an example, measurement data of another 20 km district heating grid in a small city in eastern Germany was used which allowed for a comparison of both thermal as well as hydraulic behavior of the implemented models. Figure 8 shows a brief comparison of the measured vs. the simulated pressure drop of this analyzed grid depending on the outdoor temperature. Due to the highly simplified approach of building modelling (c.f. Figure 1), the grid model cannot fully represent building storage capacities. This thus results in fluctuating measurement values regarding specific outdoor temperature. It however, sufficiently reproduces the hydraulic behavior in all grid parts (peak loads, basic loads) which are necessary for grid analyzes and system design.

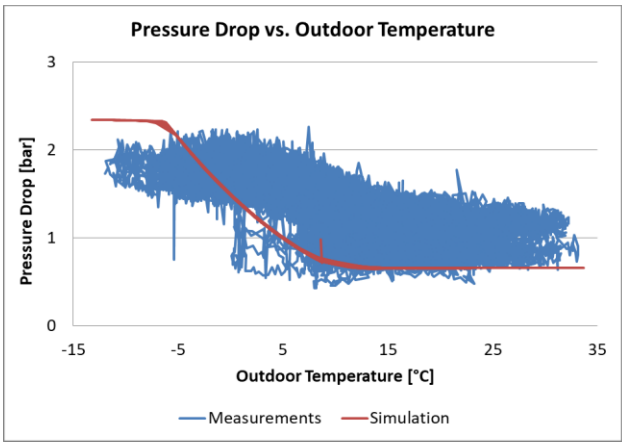


Figure 8: Comparison of measured vs. simulated pressure drop depending on outdoor temperature

The simplified modeling approach enables high performant models with sufficient simulation speed. A 50 km plus grid with about 2,000 grid elements and about 1,400 model states needs an estimated time of one to two hours for a yearly simulation. Furthermore, the dynamic modelling approach of Modelica enables the evaluation of 100+ grid operating points within a single simulation run. Existing static district heating grid simulation models only allow for the evaluation of one operating point with each simulation run (c.f. Figure 3).

Future development will include extended process automation to further expand the approach, enabling the simulation, evaluation and presentation of large-scale district heating and ultimately cooling grids.

Nomenclature

Following definitions and symbols are used within this paper to describe the model functionality in equations 1 to 15.

$\dot{Q}_{loss}$	- Heat loss over the insulation
$\Delta Q_{insulation}$	- Heat transmission coefficient of pipe
l_{pipe}	- Length of pipe
T_{med}	- Medium temperature
T_{ground}	- Ground temperature
Δp_{rough}	- Pressure drop of pipe dependent on pipe roughness
λ_{pipe}	- Pipe friction coefficient
ρ_{med}	- Medium density

v_{pipe}	- Pipe flow speed
$d_{i,pipe}$	- Inner pipe diameter
ζ_{pipe}	- Pressure loss coefficient of pipe
Δp_{bends}	- Pressure drop of pipe bends
$\Delta p_{fittings}$	- Pressure drop of pipe fittings
$\Delta p_{element}$	- Constant pressure drop of individual pipe fittings
Δp_{geo}	- Pressure drop of pipe dependent on geodetic elevation differences
g	- Gravity constant
Δz	- Elevation difference
η_{med}	- Dynamic viscosity coefficient
k_{pipe}	- Roughness coefficient
Re	- Reynolds number
$\lambda_{pipe,lam}$	- Pipe friction coefficient of laminar stream
$\lambda_{pipe,turb,smooth}$	- Pipe friction coefficient of turbulent stream for a smooth pipe
$\lambda_{pipe,turb,rough}$	- Pipe friction coefficient of turbulent stream for a rough pipe
T_{Return}	- Return temperature
$T_{Return,max}$	- Maximum return temperature
T_{Flow}	- Flow temperature
$Q_{Heat\ Consumption}$	- Building heat consumption
$q_{v,max}$	- Maximum volume flow of building's grid connection
$c_{p,med}$	- Specific heat capacity of medium
$p_{return,main}$	- Absolute pressure of main return pipe
$p_{return,branch1}$	- Absolute pressure of return branch1
$p_{return,branch2}$	- Absolute pressure of return branch2
$T_{flow,main}$	- Flow temperature in main pipe
$T_{flow,branch1}$	- Flow temperature in branch1
$T_{flow,branch2}$	- Flow temperature in branch2
$T_{return,main}$	- Return temperature in main pipe
$T_{return,branch1}$	- Return temperature in branch1
$T_{return,branch2}$	- Return temperature in branch2
$q_{v,branch1}$	- Volume flow in branch1
$q_{v,branch2}$	- Volume flow in branch2
$q_{v,junction1}$	- Volume flow in junction1
$q_{v,main}$	- Volume flow in main pipe
$\Delta p_{junction1}$	- Total pressure drop in junction1
$\Delta p_{junction2}$	- Total pressure drop in junction2

References

Soons, F.F.M.; Torrens Galdiz, J.I.; Hensen, J.L.M.; Schrevel, R.A.M. de. A Modelica based computational model for evaluating a renewable district heating system. *9th International Conference on System Simulation in Buildings, December 10-12, 2014, Liege, Belgium*

Hägg, R. Dynamic Simulation of District Heating Networks in Dymola. *Thesis for the degree of Master of Science in Engineering Department of Energy Sciences Faculty of Engineering, Lund University, 2016.*

Schwan, T.; Unger, R.; Lerche, C.; Kehrer, C. Model-Based Design of Integrative Energy Concepts for Building Quarters using Modelica. *10th International Modelica Conference, March 10-12, 2014, Lund, Sweden.*

Schweiger, G.; Runvik, H.; Magnusson, F.; Larsson, P.-O.; Velut, S. Framework for dynamic optimization of district heating systems using Optimica Compiler Toolkit. *12th International Modelica Conference, May 15-17, 2017, Prague, Czech Republic.*

Hermansson, K.; Kos, C. Building and Simulating Dynamic Models of District Heating Networks with Modelica. *Master thesis, School of Business, Society and Engineering Energy Engineering, Mälardalen University, Sweden, 2017.*

Neidhold, T.; Hofmann, T. New in SimulationX 3.9. *ESI Forum 2018, November 8-9, 2018, Weimar, Germany.*

Optimization of District Heating Systems: European Energy Exchange Price-Driven Control Strategy for Optimal Operation of Heating Plants

Abdulrahman Dahash[1] Annette Steingrube[2] Fabian Ochs[1] Mehmet Elci[2]

1. Unit of Energy Efficient Buildings, University of Innsbruck, Technikerstraße 13, 6020 Innsbruck, Austria, Abdulrahman.Dahash@uibk.ac.at
2. Fraunhofer-Institute for Solar Energy Systems, Heidenhofstraße 2, 79110 Freiburg im Breisgau, Germany Annette.Steingrube@ise.fraunhofer.de

Abstract

District heating (DH) systems are often seen as a good practical approach to meet the local heat demand of districts. Yet, under today's regulations to renovate buildings on high efficiency standards, the local heat demand is decreasing. Therefore, the operation of DH systems is also affected by the changing heat demand profile, which might lead to less profit for the operators of DH systems. Thus, the operators strive for an optimal operation at which the heat demand is met and the profits are maximized. In this work, a control strategy for optimal operation of a combined heat and power (CHP) based DH is presented. The proposed control strategy couples the operation of CHPs to the European energy exchange (EEX) price by implementing different operation constraints. This configuration is accompanied with another, which is the installation of additional storage volume. Thereby it is held to provide the optimal operation for the plant technically and economically.

Keywords: Modelica/Dymola, District Heating, Heating Plant, Power-Based Model, Optimal Operation, Control Strategy, Storage.

1 Introduction

District heating (DH) systems represent a key energy solution that have been deployed for years in a growing number of cities worldwide (Werner, 2017) (Rezaie, B. and Rosen, M. A., 2012). Thereby, DH systems are envisioned as an effective approach to provide affordable, local and low-carbon energy to the consumers through diversity of supply, energy balancing and storage (Guelpa et al., 2018). Therefore, DH systems are envisioned as one of the practical approaches for the global transition to sustainable energy utilization in many urban centers (Fiacro Castro Flores, 2018).

Moreover, combined heat and power (CHP) based DH systems are seen as a flexible heat-supply option as it provides heat to meet the local heat demand in urbans and, therefore, these CHPs are frequently heat driven (Elci et al., 2015). CHP produces also electricity as a byproduct and feeds it into the national power grid helping in balance it due to the fluctuating renewables, especially during periods where renewables hardly provide useful energy (Buffat, R. and Raubal, M., 2019). Consequently, CHP's electricity is fed into the grid at a variable or fixed tariff depending on the European energy exchange (EEX) market. Thus, this fed electricity might lead to gain profits out of the electricity produced.

In the coming years, it is believed that buildings' heat demand will gradually decrease due to the refurbishment regulations. The goal in the refurbishment process is to have energy demand as low as possible. This demand profile and the national electricity demand fluctuate seasonally and hourly with asynchronous patterns, thus it is important to guarantee an optimal-operation of the heating plants coupled to the DH networks. Thus, the operators of DH systems strive for an optimal operation at which the heat demand is met and the profits are maximized (Dahash et al., 2017). Consequently, it is of importance to introduce an optimal-operation for the heating plants coupled to DH networks.

Furthermore, this operation strategy should be subjected not only to the buildings' heat demand, but also to electricity selling price in the market, fuel costs and electricity demand in the national power grid. In this context, it is pointed out that such an optimal operation is a future challenge in DH domain due to the complexity, the high number of parameters and its combinatorics and the optimal planning for heat-generation between the different heat sources in the heating plant (Zhou et al., 2014). Following this challenge, it is highly advised to rely on decision support/making tools, which are dependent on model predictive control (MPC) to achieve the optimal operation (Giraud et al., 2017).

In this study, we present a control strategy for optimal operation of CHPs in heating plants coupled to DH networks. In this control strategy, the electricity market price is introduced to take advantage of the periods during which the electricity price is relatively high to maximize revenue. To test this strategy, a validated power-based model of a DH system is used. This model shows the amount of energy flows between the different parts of the DH system (supply side, transmission network and demand side). The reasons behind this modeling approach are the less simulation time and the better insight in the heating plant's operation (Dahash, 2016). For the modeling, Modelica/Dymola was used as a simulation tool since Modelica supports the description of mathematical equations following the physical modeling paradigm.

2 Methodology

2.1 Case Study

As a case study, a district in the city of Freiburg in the south of Germany was used. This district is called Weingarten and it was built in the 1960s. Under current regulations regarding comfort, energy efficiency and modern building technology, the buildings in the western part have to be renovated into more energy-efficient buildings. This refurbishment requires major structural, technical and economic interventions in order to modernize the district's buildings, renew Weingarten's energy supply system and operate it optimally.

The district's heat demand is provided by a central heating plant that supplies heat to two districts (i.e. Rieselfeld, Weingarten) via a DH network as shown in Figure 1.

In the heating plant, 6 gas-fired CHP units, each with 1200 kW_{el} / 1490 kW_{th}, are installed and the operation of them is mainly heat-driven. Consequently, two CHP units are operating almost continuously year-round to meet the base-load. Thus, over 75 % of the annual amount of heat produced comes from CHP units, while the remainder is generated by peak boilers (3*9300 kW). Also, in order to achieve smooth operation of the CHP units, there are two thermal energy storage (TES) systems with a total volume of 360 m^3. They help in meeting the demand over short periods.

2.2 DH System Model Description

A thermo-hydraulic model of a DH system with fully described details requires high computational efforts (simulation time), particularly when modelling large DH systems with many heat supply technologies, consumers, long-distance networks and complex operation scheme. Accordingly, it is important to reduce the complexity of the models to a degree in which all physical properties remain accurate. Thus, it is vital to underline the crucial goal of modelling process in order to reduce the computational efforts and time (Dahash et al., 2019).

In this study, the main goal is to simulate correctly the operation of the central heating plant and to test the proposed control strategy and its applicability with another optimization configuration (installing additional TES volume). As a result, many general assumptions are set in the system layout in order to simplify the model. These assumptions are as follows:

1. The demand side (consumers) is modelled as a single heat sink. This sink is directly connected to the network. This means there is only one single loop in which the heat sink is directly coupled to the network without the need of heating substations.
2. The three peak boilers in the heating plant are represented in a single equivalent boiler with a total thermal power of 27,900 kW.
3. Since the two TES units are connected in series in the heating plant, it is possible to represent them as one single unit. Therefore, a single TES unit with a total volume of 360 m^3 is implemented in the model.

The heating plant model used in this study is extensively described in an earlier study (Dahash et al., 2017). Also, the validation of this model was carried out in the aforementioned literature. Therefore, only the control strategies are comprehensively discussed in the following sections in order to comprehend the changes in the control strategy and to compare the results. Moreover, the reference and the proposed control schemes for the CHP are presented, whereas no changes are seen necessary for the boiler and storage controllers.

2.3 Reference Control Strategy

2.3.1 CHP Controller

Herein, the bottom segment temperature for storage is set to 70°C and the upper one is set to 100°C. For each CHP unit, an individual CHP controller is installed in the model. In this controller, the heat demand and the storage temperatures (upper and bottom) are simultaneously checked. From Figure 2, 3 cases can be determined to run the CHP unit, which are:

1. **Power case (a):** if the heat demand is higher than the nominal CHP's heat output and the temperature of the bottom segment is higher than 70°C, then the CHP unit runs.
2. **Power case (b):** if the heat demand is higher than nominal CHP's heat output and the temperature of the bottom segment is lower than 70°C, then the CHP unit runs.
3. **Power case (c):** the CHP unit runs, when the following conditions are all true:
   i. The heat demand is lower than nominal CHP's heat output, and
   ii. The heat demand is higher than 95 % of the CHP's heat output (equals 1.425 MW), and

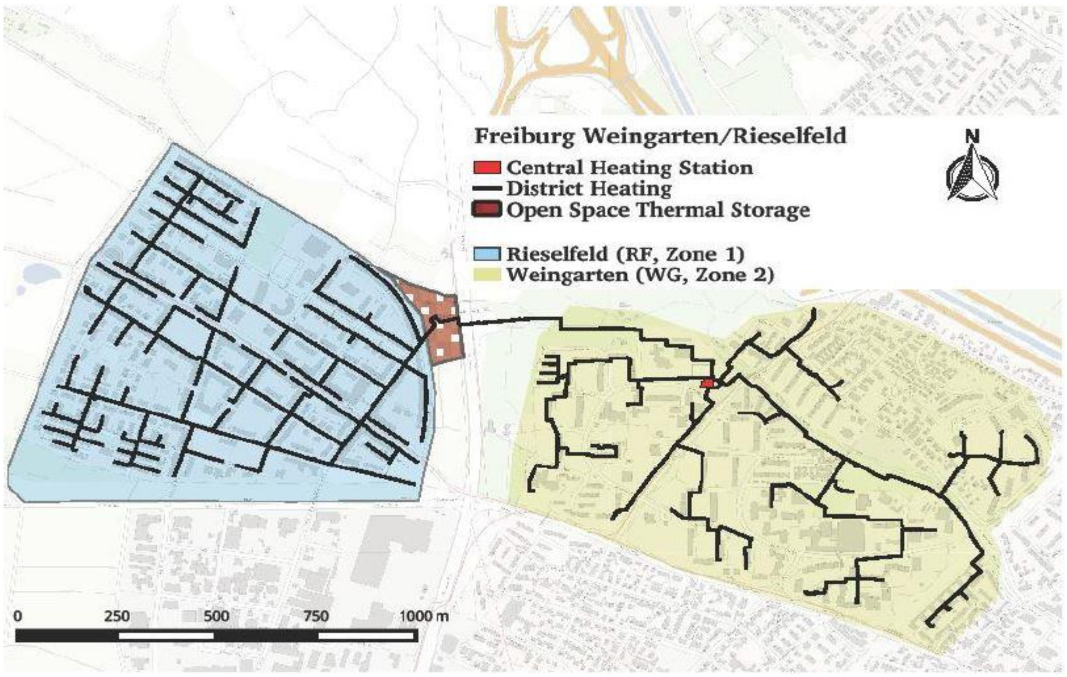


Figure 1: Geographical top view of the Weingarten and Rieselfeld district with the central heating station and DH network (Bachmaier et al., 2015)

iii. The upper storage temperature is lower than 95°C.

Regarding power case (a), as the storage temperature is equal to or higher than 70°C, this means the storage can be discharged. On the contrary, if the storage temperature is less than the set bottom temperature (70°C), this means the energy stored in the storage system cannot be used and, therefore, power case (b) is activated to supply the heat directly to the consumers. While power case (c) is activated in order to cover the heat demand that is higher than 1.425 MW and the remaining of the heat output charges the storage.

Moreover, if the heat demand (or the remaining heat demand for CHP 2-6) is less than 1.425 MW or the upper storage temperature is higher than 95°C, then the corresponding CHP unit turns off.

2.3.2 Storage Controller

This controller plays a role in the energy balance of the entire heating plant, since it gives a signal to discharge or charge the storage system. The controller flowchart is shown in Figure 3. Therein, it is illustrated that there are two inputs and a single output. One input is the storage bottom temperature. Based on the temperature, a decision is made as whether the storage system can be discharged.

If the temperature of storage's bottom, however, is higher than 70°C, this sends a true signal to the switch component to discharge the storage system to cover the remaining demand. Otherwise, the output is set to zero when the temperature is less than 70°C. Thus:

$$\dot{Q}_{\text{storage}} = \dot{Q}_{\text{demand}} - \sum_{i=1}^{6} \dot{Q}_{\text{CHP},i} \qquad (1)$$

Occasionally, the storage system cannot be discharged because the last segment temperature is less than that allowed for discharging and, therefore, the remaining heat demand proceeds to the next controller, which is the boiler controller that runs the boiler in order to meet the required amount of heat.

2.3.3 Boiler Controller

The boiler controller is a simple unit, which computes how much heat demand remains after the total output of the CHP units and the discharged capacity of the storage system. Next, it gives an output signal to run the boiler in a partial mode to meet the remaining heat demand, thus:

$$0 \leq \dot{Q}_{\text{boiler}} \leq 27.9\ \text{MW} \qquad (2)$$

Here, the remaining demand is computed as below:

$$\dot{Q}_{\text{boiler}} = \dot{Q}_{\text{demand}} - \sum_{i=1}^{6} \dot{Q}_{\text{CHP},i} - \dot{Q}_{\text{storage}} \qquad (3)$$

The term $\dot{Q}_{\text{storage}}$ refers to the usable heat in the storage system. Therefore, the usable temperature lies between 70°C and 100°C.

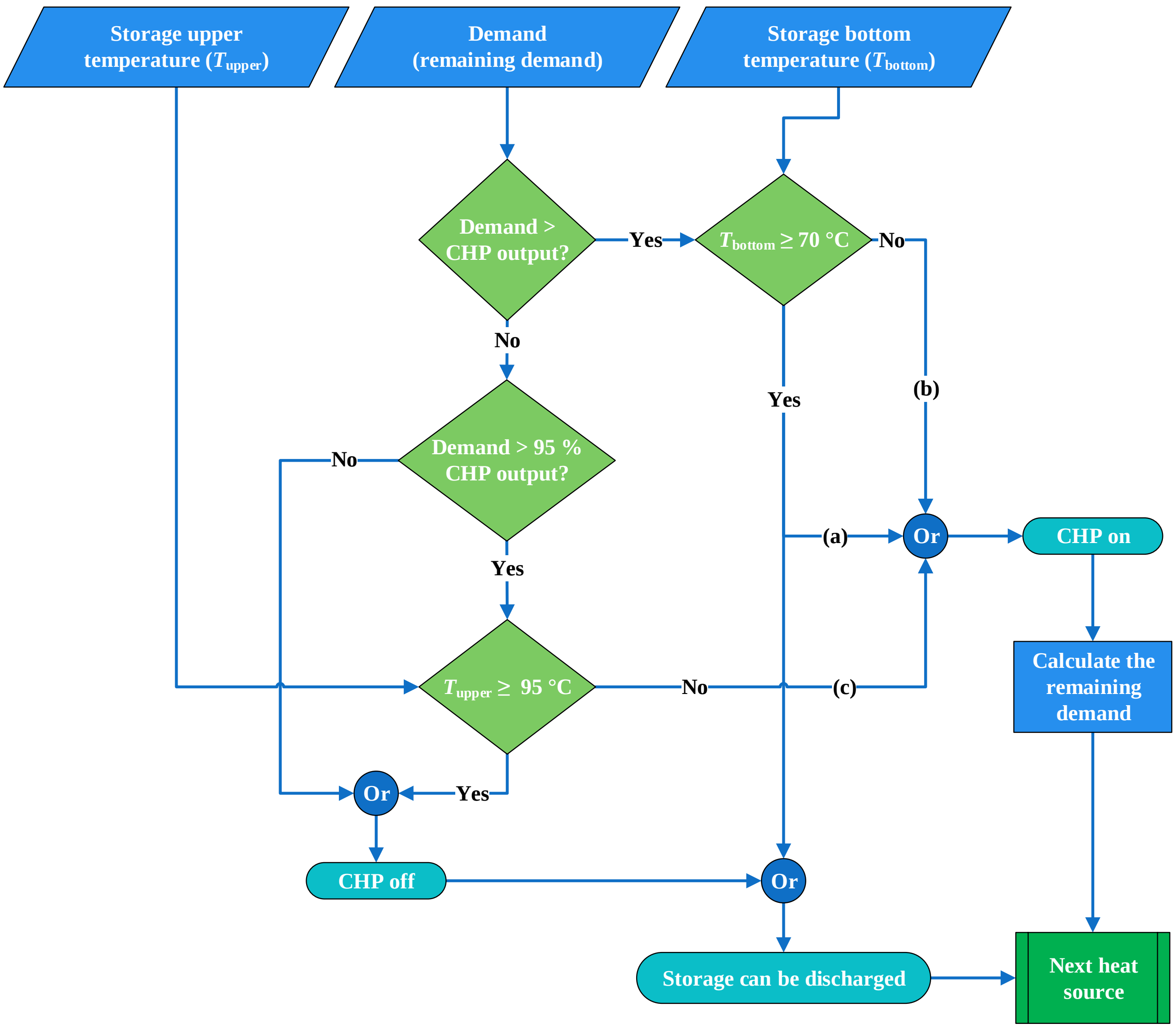

Storage upper temperature (T_{upper})
Demand (remaining demand)
Storage bottom temperature (T_{bottom})
Demand > CHP output?
Yes
$T_{bottom} \geq 70$ °C
No
No
Yes
(b)
Demand > 95 % CHP output?
No
Yes
(a)
Or
CHP on
Calculate the remaining demand
$T_{upper} \geq 95$ °C
No
(c)
Yes
Or
CHP off
Or
Storage can be discharged
Next heat source

Figure 2: CHP controller flowchart (reference case).

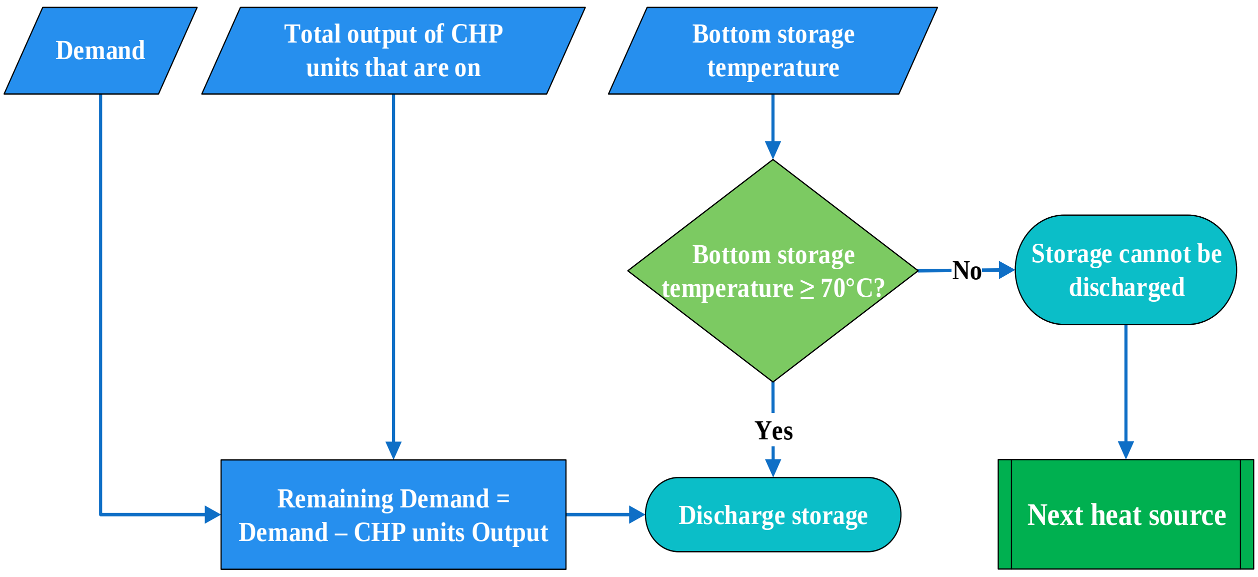

Demand
Total output of CHP units that are on
Bottom storage temperature
Bottom storage temperature ≥ 70°C?
No
Storage cannot be discharged
Yes
Remaining Demand = Demand – CHP units Output
Discharge storage
Next heat source

Figure 3: Storage controller flowchart.

2.4 Proposed Control Strategy

In this control strategy, the major changes take place in the CHP controller, as it is the primary regulator of the heating plant. Firstly, a price threshold is determined and implemented in the CHP controller. Then a constraint is set in the controller, if EEX price is higher than this threshold, then it is worthwhile to run the CHP units to maximize the revenue regardless of the heat demand. Nevertheless, the following question arises: **What if a situation occurs where the storage is fully charged, the EEX price is high enough to run the CHP units and the heat demand is relatively low?**

Having considered this scenario, the control strategy is constructed such that if the heat demand is high and EEX price is lower than the threshold, the CHP units run to meet the demand and the rest is stored. Accordingly, the storage is set to a predetermined temperature. In addition, when EEX price is more than the threshold, the CHP units run to feed the electricity produced into the grid and the storage is charged until 100°C (fully charged) regardless of the heat demand. Furthermore, to determine the optimum storage temperature to fulfill the energy balance constraint, an iterative algorithm is built as shown in the Figure 4.

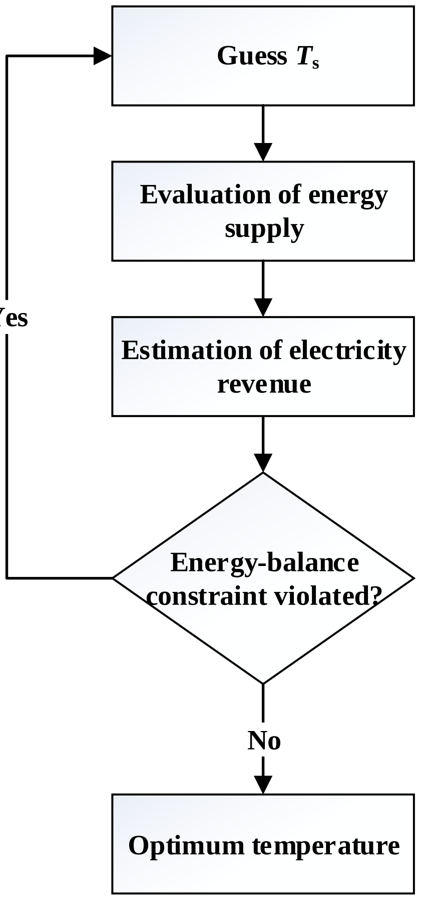

Guess T_s
Evaluation of energy supply
Estimation of electricity revenue
Energy-balance constraint violated?
Yes
No
Optimum temperature

Figure 4: An iterative approach to determine the optimum storage temperature.

Figure 4 illustrates that firstly, a guess for the storage temperature is made, and it is assumed 80°C, and then the simulation runs until the energy supplied by the heating plant is seen. Next, the energy-balance constraint is inspected, if it is true and there is a balance, then optimum storage temperature is found. If not, then another guess is made and the simulation runs again until the energy balance constraint is fulfilled. Following this iterative approach, the optimum storage temperature is determined to be 86°C and the price threshold is set to a specific value (e.g. 36 €/MWh), at which the energy-balance constraint is fulfilled. Thus, the storage capacity is used up to 86°C when there is a heat demand regardless of the EEX price. Whereas the rest of storage capacity (86°C up to 100°C) is kept for periods at which EEX, price higher than the threshold.

Essentially, the CHP units are forced to run when the electricity price is high even though there is no heat demand. Therefore, the heat can be stored in the TES and the electricity is fed into the grid and sold for high prices. Figure 5 illustrates the flowchart for this proposed controller, and it is seen that there are 4 power cases to run the CHP unit.

The power cases are:

1. **<u>Power case (a):</u>** if the heat demand is higher than the nominal CHP’s heat output and the EEX price is higher than 36 €/MWh, then the CHP unit runs.
2. **<u>Power case (b):</u>** if the heat demand is higher than nominal CHP’s heat output and the temperature of the bottom segment is lower than 70°C, then the CHP unit runs.
3. **<u>Power case (c):</u>** if the heat demand is higher than the nominal CHP’s heat output and the temperature of the bottom segment is higher than 70°C, then the CHP unit runs.
4. **<u>Power case (d):</u>** if the heat demand is lower than the nominal CHP’s heat output and the EEX price is higher than 36 €/MWh, then the CHP unit runs.

It is worth to mention that if the power case (d) is activated, this means the heat demand is lower than CHP’s heat output and, therefore, the demand is met firstly and the rest goes to the storage to be stored. The storage here is allowed to store heat up to 100°C, this matches (f) in Figure 5. While if the EEX price is lower than 36 €/MWh, then the storage is allowed to store the heat at a maximum temperature of 86°C regardless the heat demand as seen in the flowchart above, this matches line (e) in the flowchart. Also, in case the heat demand is higher than CHP’s heat output, then the CHP unit runs regardless the EEX price and this constraint is covered by power cases (b) and (c). Moreover, in the below flowchart, the cases to discharge the storage are not shown.

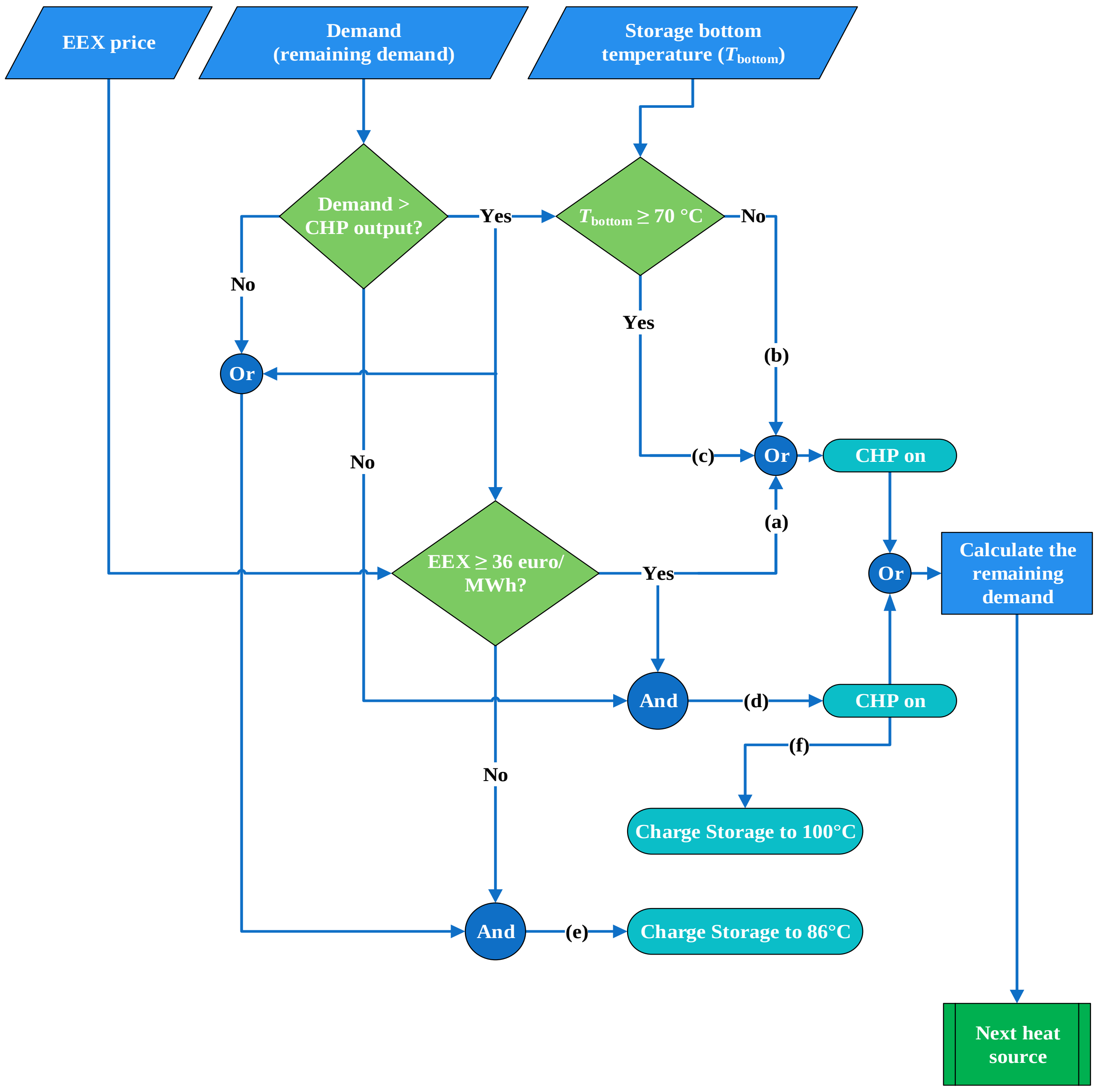


Figure 5: CHP controller flowchart (proposed case).

2.5 Techno-Economic Governing Equations

The energy consumed (gas) to cover the production of heat and electricity from a CHP is computed as:

$$Q_{\text{fuel}} = \frac{Q_{\text{th}} + E_{\text{el}}}{\eta_{\text{tot}}} = Q_{\text{gas}} \tag{4}$$

Same equation (4) is also applied to calculate the thermal energy produced by and energy supplied to the boilers, as they only produce heat. Additionally, the total cost of fuel energy supplied (i.e. gas) to the CHP or the boiler is given by the following equation:

$$C_{\text{gas}} = c_{\text{gas}} \cdot Q_{\text{gas}} \tag{5}$$

c_{gas} is the specific cost of the gas as stated in the contract between the gas supplier and the heating plant operator. It is taken as 30 €/MWh, whereas C_{gas} is the cost of gas in euros.

As the heating plant operator, Badenova, sells the heat produced with a mean price of 5.5 cent/kWh calculated out of Badenova's pricing sheet, the revenue gained from the heat produced is also computed as below:

$$R_{\text{heat}} = c_{\text{heat}} \cdot Q_{\text{demand}} \tag{6}$$

On the other hand, Badenova does not sell the electricity generated at a fixed price due to a varying EEX price in electricity market. Thus, it is possible to implement the variable electricity price in the future and, therefore, the electricity revenue is calculated as below:

$$R_{\text{electricity}} = \int_0^t P_{\text{el}} \cdot c_{\text{el}}(t) \cdot \mathrm{d}t \tag{7}$$

c_{el} is the current electricity price in EEX in [€/MWh].

All these equations are implemented in Dymola as computational units for both operations (reference and proposed) to compute the different parameters (thermal energy production, electricity generation, gas consumption, revenues and costs). After calculating these values, the annual net profit of each operation is calculated as below:

$$NP = R_{\text{heat}} + R_{\text{electricity}} - C_{\text{total}} \quad (8)$$

C_{total} is the total cost of gas supplied to the heating plant for both of CHP units and boilers.

3 Simulation Results and Discussion

The determination of the storage temperature is the most critical aspect in order to avoid violating the energy-balance constraint. Therefore, many simulations were carried out following the iterative procedure that is described in Figure 4. Nevertheless, only the results of successful simulations are considered here and compared with the reference case of the model (the normal CHP controller).

The simulation results reveal a slight increase in the operation of CHPs during the years 2014, 2015, 2016 and 2020 compared to the reference operation. In these scenarios, year 2014 represents the buildings' heating demand before the refurbishment, whereas the years 2015 and 2016 stand for the status during the refurbishment. Year 2020 represents the forecasted heating demand when the buildings are completely renovated into more energy-efficient ones.

Furthermore, the increase in CHPs operation is accompanied by a reduction in the produced heat from boilers as seen in Figure 6.

Yet, Figure 6 depicts that the CHPs' heat output decreases as the price threshold increases from 30 €/MWh up to 45 €/MWh. This is because the period in which EEX price around 30 €/MWh is more frequent compared to both other periods. Therefore, more heat is produced from the boilers during periods with high price threshold. If the price threshold is set to 45 €/MWh and EEX price is around 30 €/MWh, then the CHPs operation will not be feasible since they run in full-load and, thus, they demand high gas consumption. So, they generate electricity, which will be sold at low EEX price and, then, low profits. Whereas it is more feasible to run the boilers since the can run in partial load and cover the low heat demand fully.

As a result, the total gas consumption also varies, which also has in return an influence on the economic feasibility of the new control strategy. However, since the operator of the heating plant buys gas with a fixed price, then the changes in gas consumption have minor influence on the economic feasibility compared to the influence of EEX price.

To increase the feasibility of this control strategy, additional TES units are installed. Each unit with a volume of 210 m^3. The additional volume will allow the CHPs to run smoother, longer and more flexible during periods at which EEX price is high enough. During such periods, the heat demand is met, whilst the remainder of produced heat is stored in TES. When heat is needed later and EEX price is low, then there is no need to run the CHPs or the boilers. It could be enough to discharge TES and if more heat is needed, then the boilers run.

Out of the results, it is decided to select years 2016 and 2020 and show their results in context of CHPs and boilers heat output to demonstrate that increasing the TES volume with such a controller might lead to less operation of boilers and CHPS can run longer and, hence more profits.

Figure 7 and Figure 8 illustrate the amount of heat produced by CHPs and boilers with an increasing TES volume for the selected years (2016 and 2020).

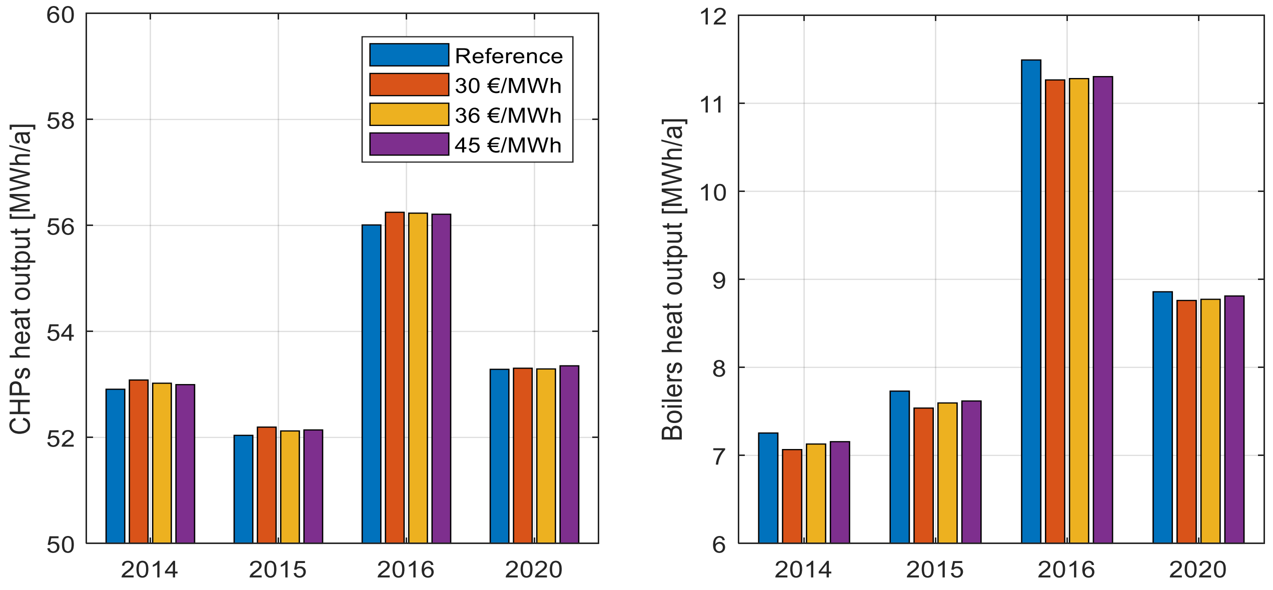


Figure 6: Comparison of CHPs' and Boilers' heat output for three price thresholds compared to the reference case (blue bar) during the years 2014, 2015, 2016 and 2020 (All cases are subjected to V_{TES} = 360 m^3).

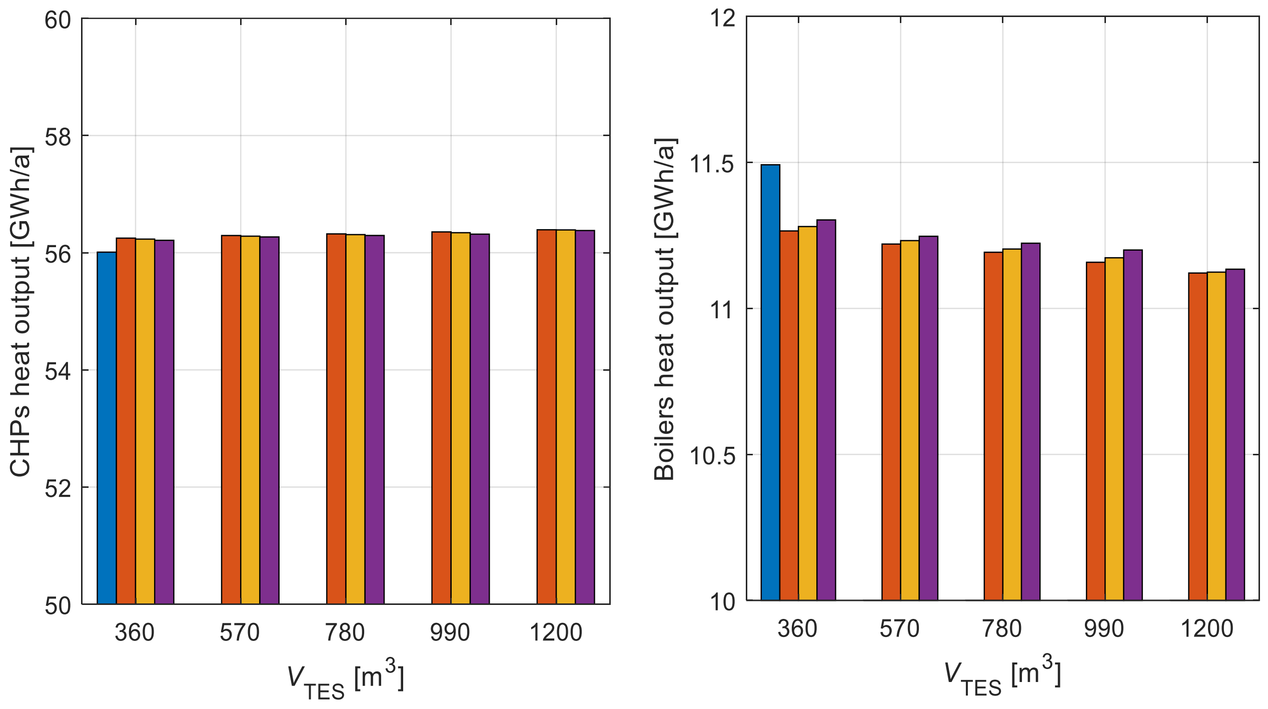


Figure 7: Comparison of CHPs' and Boilers' heat output for three price thresholds compared to the reference case (blue bar) during the year 2016.

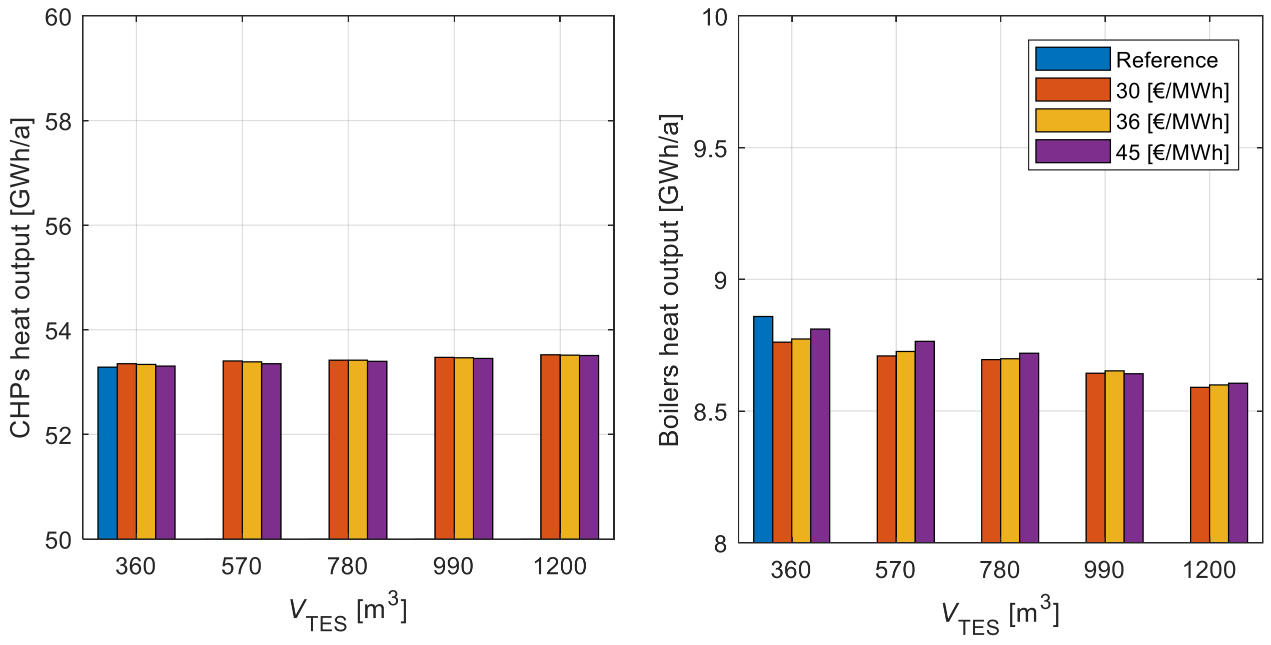


Figure 8: Comparison of CHPs' and Boilers' heat output for three price thresholds compared to the reference case (blue bar) during the year 2020.

The annual net profit is calculated for each scenario following equation (8) and then compared to the reference scenario to display the annual net profit percentage (ΔNP (%)) in Figure 9. The net profit percentage is computed from the following equation:

$$\Delta NP\ (\%) = \frac{(NP)_{\text{pro}} - (NP)_{\text{ref}}}{(NP)_{\text{ref}}} \cdot 100 \qquad (9)$$

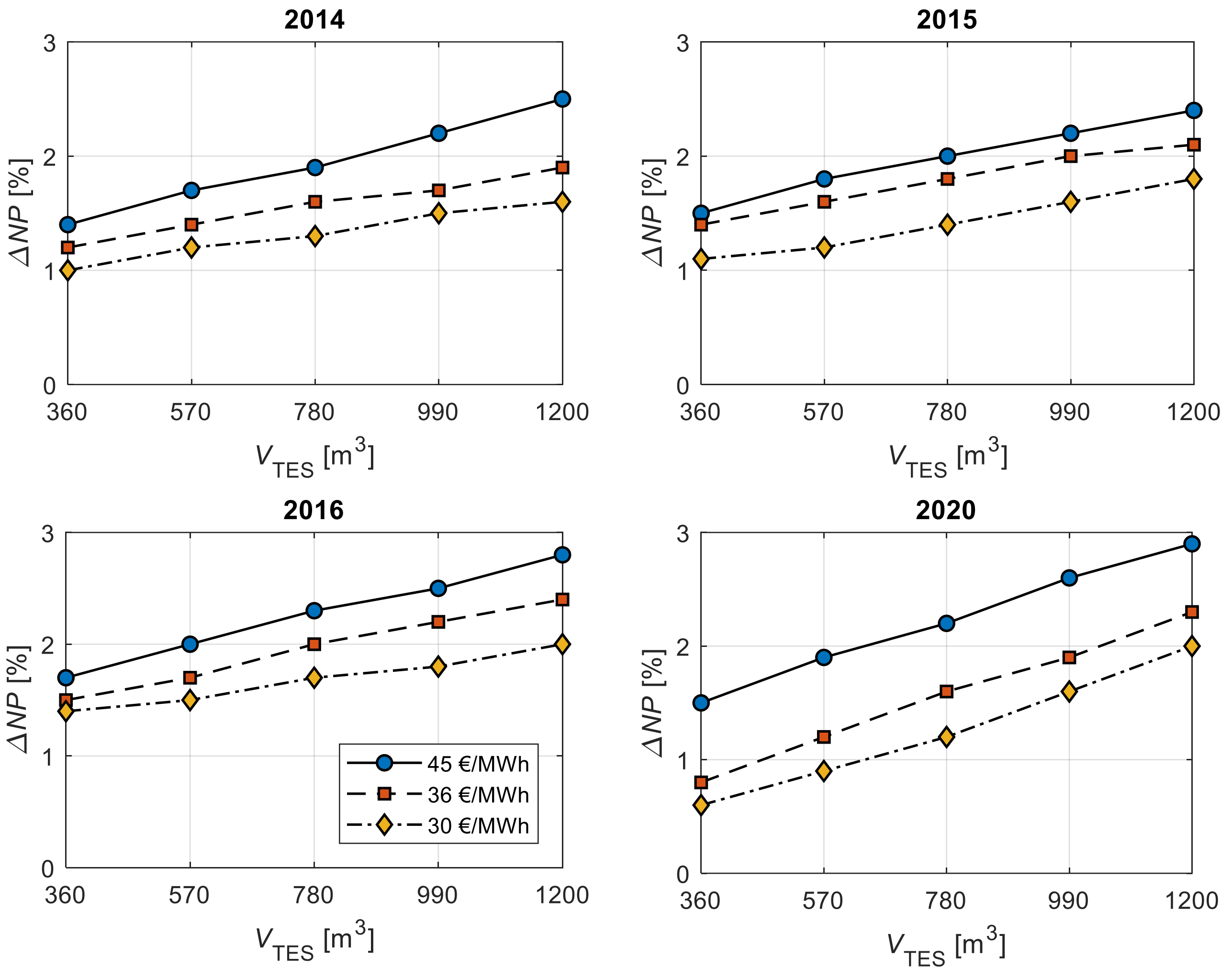


Figure 9: Net profit percentage for the proposed control strategy at different storage volumes compared to the reference case (reference control strategy and V_{TES} = 360 m^3).

It is worthwhile to mention that the control strategy is seen not highly beneficial in year 2020 for the TES volume of 360 m^3. This is attributed to the expected negative EEX prices in the corresponding year, which in return is due to higher electricity share from renewables. Therefore, a drop in EEX prices is expected. Thus, the CHPs run for less time during these periods.

The results pinpoint that this control strategy is held to be feasible, especially in the cases with a TES volume beyond 360 m^3. This volume increase enhances the operation of the heating plant and maximizes the profits as seen in Figure 9.

Obviously, the volume increase triggers the CHPs to run more during periods that do not violate the EEX threshold. Therefore, more thermal energy can be stored in the heat storage even if the heat demand is met. Then, this additional heat is used later when heat is required and EEX prices are lower than the threshold. Thus, TES can be discharged. Additionally, it is significant to underline that this volume increase reduces the boilers' operation to a minimum.

Furthermore, the numerical results indicate that more storage volume yields effective and feasible operation of the heating plant in total as seen Figure 7 and Figure 8, therefore, it is translated into higher profits as shown in Figure 9.

4 Conclusion

This study was carried out to optimize the operation of a CHP-based DH system. A valid power-based model for DH systems was used. Two promising optimization configurations are selected and tested accordingly.

Firstly, an EEX-price driven control strategy is examined to quantify the economic benefits. The outcomes indicate a slight increase in the profits (see Figure 6). Therefore, a second configuration was also tested in parallel with the aforementioned configuration (EEX-driven control strategy), which is installing additional TES volume. This configuration seems to deliver better results than standing the proposed control strategy alone (see Figure 9).

Moreover, an increase in the profit of 0.55 % (price threshold of 30 €/MWh) in year 2020 is not a strong motivation for the operator of the heating plant to implement the new control strategy. Therefore, it is worthwhile to add some other constraints to the control strategy in order to achieve the economic feasibility. Thus, it is recommended to further constraint the CHPs operation such that if the EEX price is negative, then the boilers run instead.

Future work will focus on inspecting the influence of the proposed control strategy on the DH supply and return temperatures. Therefore, thermo-hydraulic models are needed for this task in order to develop the control strategy further and to determine the optimal storage volume.

5 Acknowledgements

This work is part of the project *Weingarten 2020 Monitoring* funded by the BMWi (Federal Ministry for Economic Affairs and Energy, Project No.: O3ET2364A). The presentation of this work is financed by the Austrian “Klima- und Energiefonds” and performed in the frame of the program “Energieforschung”. It is part of the Austrian flagship research project “Giga-Scale Thermal Energy Storage for Renewable Districts” (giga_TES, Project Nr.: 860949). Therefore, we wish to acknowledge the financial funding from the aforementioned agencies. Also, our thanks go to the operator of Weingarten heating plant, Badenova WärmePlus, for the cooperation in the project.

6 Nomenclature

Symbol	Description	Unit
c	Specific cost	[€/MWh]
C	Cost	[€]
E	Energy (i.e. electrical)	[J]
NP	Annual net profit	[€/a]
P	Power	[W]
Q	Heat	[J]
$\dot{Q}$	Heat flow rate	[W]
R	Revenue	[€]
t	Time	[s]
V	Volume of the storage tank	[m^3]
Δ	Difference	[-]
η	Efficiency	[-]
Abbreviation	**Definition**	
DH	District heating	
TES	Thermal energy storage	
CHP	Combined heat and power	
EEX	European energy exchange	
MPC	Model predictive control	

7 References

Bachmaier et al., 2015. Spatial Distribution of Thermal Energy Storage Systems in Urban Areas Connected to District Heating for Grid Balancing. *Energy Procedia,* Volume 73, pp. 3-11.

Buffat, R. and Raubal, M., 2019. Spatio-temporal potential of a biogenic micro CHP swarm in Switzerland. *Renewable and Sustainable Energy Reviews,* Volume 103, pp. 443-454.

Dahash et al., 2017. *A Power-Based Model of a Heating Station for District Heating (DH) System Applications.* Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, pp. 415-424.

Dahash et al., 2019. A comparative study of two simulation tools for the technical feasibility in terms of modeling district heating systems: An optimization case study. *Simulation Modelling Practice and Theory,* Volume 91, pp. 48-68.

Dahash, A., 2016. *A Comparative Study of Modeling Approaches for District Heating Systems,* Offenburg, Germany: Offenburg-University of Applied Sciences.

Elci et al., 2015. Grid-interactivity of a Solar Combined Heat and Power District Heating System. *Energy Procedia*, Volume 70, pp. 560-567.

Fiacro Castro Flores, J., 2018. *Low-Temperature Based Thermal Micro-Grids: Operation and performance assessments*, Stockholm, Sweden: KTH Royal Institute of Technology.

Giraud et al., 2017. *Optimal Control of District Heating Systems using Dynamic Simulation and Mixed Integer Linear Programming.* Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, pp. 141-150.

Guelpa et al., 2018. Thermal request optimization in district heating networks using a clustering approach. *Applied Energy,* Volume 228, pp. 608-617.

Rezaie, B. and Rosen, M. A., 2012. District heating and cooling: Review of technology and potential enhancements. *Applied Energy,* Volume 93, pp. 2-10.

Werner, S., 2017. International review of district heating and cooling. *Energy,* Volume 137, pp. 617-631.

Zhou et al., 2014. Dynamic modeling of thermal conditions for hot-water district-heating networks. *Journal of Hydrodynamics, Ser. B,* 26(4), pp. 531-537.

Automated model generation and simplification for district heating and cooling networks

Michael Mans[1] Tobias Blacha[1] Peter Remmen[1] Dirk Müller[1]

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

Abstract

The current operation of district heating networks often relies on static analyses and control parameters. In the future, possible integration of renewable energy sources like solar or geothermal energy are getting more and more important. To investigate the impact of these new energy source in combination with new control strategies, dynamic simulation models for district heating and cooling systems are getting more important. However, these models are often large and therefore have large computation times and require manual effort to create and optimize them. Thus, this paper investigates in the simulation of a large and meshed district heating network. We present a workflow for automated generation and model simplification of simulation models based on GIS data. The validity of the model simplification is proven and the usability of the model is demonstrated by a Use Case with two different control strategies.

Keywords: District Heating and Cooling Networks, Model Simplification, Control Strategies

1 Introduction

The current design and operation of district heating networks often relies on static analyses and control parameters. In the future, possible integration of renewable energy sources like solar or geothermal energy as well as distributed heat sources is getting more important. As the integration of renewable and distributed energy sources are influencing the dynamic behavior of thermal networks, understanding and representing these dynamic processes within such systems is of great importance. This applies not only to the design of these networks but in particular the operation and implementation of different control strategies. Dynamic simulation models for thermo-hydraulic systems provide an opportunity to gain insights in the dynamics of district heating and cooling (DHC) networks. Although various demonstration for complex district heating and cooling (DHC) systems, with high share of renewable energy have already been realized (Lund et al., 2014) and dynamic models have been applied to thermal networks (Schweiger et al., 2017), dynamic modeling of DHC networks are still rarely used for design and operational optimization. One of the reasons for this is the complex modeling of these systems.

In recent years, the IEA-EBC Annex 60 (Wetter et al., 2015) and the follow-up project IBPSA Project 1 develop models in the modeling language Modelica for building and urban energy systems. This cooperation has resulted in a thermo-hydraulic pipe model that meets the requirements for the simulation of complex district heating and cooling networks (van der Heijde et al., 2017). In addition, tools were developed for the automated generation of models of complex network structures (Fuchs et al., 2016). These tools make use of object-oriented programming in Modelica and GIS (Geographical Information Systems) data to generate network system models.

In the case of complex networks, which are characterized, for example, by several feed in or a meshed network topology, large and complex models are the consequence. These models have a large number of equations, state events and Jacobian-evaluations. This makes the translation and simulation of these models slow and often leads to instabilities. Optimizing the models with several hundred pipes and substation is very time-consuming and requires manual input as well as very detailed expert knowledge. One option to make the models faster and more stable is topological and parametric model simplification. Model simplification for heat network calculations has been widely used, (Larsen et al., 2004). However, these simplifications often involve a loss of spatial information. For distributed networks with multiple feed in with different temperature levels, the spatial distribution plays a decisive role.

For this reason, we present a methodology for model simplification for complex and meshed thermal networks. In a first step, GIS data is subjected to a topological simplification. Certain pipes are replaced by weighted analogous pipes. This is necessary to create an executable system model, which still reflects the spatial properties of the network. In the second step, particularly short pipe sections are identified and modeled with a static pipe model. This reduces the number of equations, state events and Jacobian-evaluations but still keeps the spatial resolution of the model.

The paper is divided into five chapters. After the introduction, the methodology section describes the models used. In addition, the methodologies for model simplification are presented in more detail here. The presented methodology is tested using the example of two control strategies for the thermal network of a research center in Germany. Before the paper ends with a summary, the limitations and an outlook, a verification of the model simplification as well as results of the use case are shown in the results section.

2 Methodology

In this section we present different models and tools used for the dynamic simulation of district heating and cooling networks. For this purpose, the dynamic simulation models of the individual components of a district heating or cooling system that are necessary to build a system model are presented first. Afterwards it is shown how these component models are used for the automated generation of a district heating system model using the Python tools *uesgraphs* and *uesmodels*, (Fuchs et al., 2016). Starting from a GIS data set, the process of model generation, parameterization and model simplification is described. Finally, the control strategies of the district heating network considered as an example are presented.

Dynamic Simulation Models

For the dynamic simulation and analysis of district heating networks, dynamic, thermo-hydraulic simulation models are used which are developed in the modeling language Modelica. Modelica provides the possibility to text-only description of the models and a object oriented modeling approach. This makes it easy to provide a model generation process with the help of other frameworks, more suitable for large data handling, such as Python. In addition, Modelica is capable of a multi-physics simulation approach which is useful for future investigations regarding the flexibility of thermal networks to the electrical network for example.

The system model of the district heating networks consists of three main components: The models for the hydraulic network (i. e. the pipes and junctions), the substation models that represent the consumers connected to the network and the model of the central heat supply. These models are combined according to the considered network

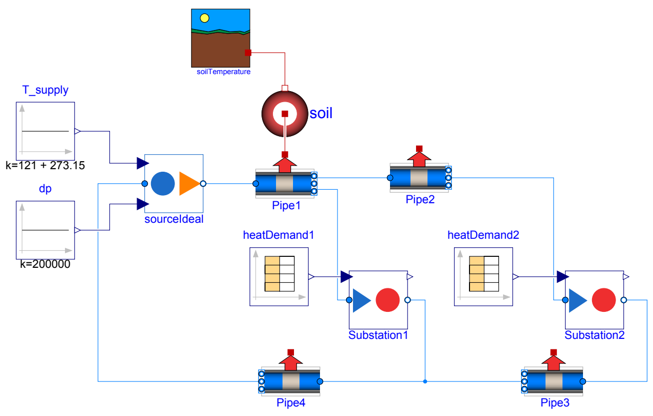


Figure 1. Exemplary system model of a district heating network

topology to build a system model of the respective DHC network. The individual component models are connected to each other by fluid connectors, developed in the Annex 60 (Wetter et al., 2013). These connectors provide information on the state of the fluid, including the mass flow, thermodynamic pressure and the specific thermodynamic enthalpy of the medium. All used models are developed within the Modelica libraries Modelica IBPSA (Wetter et al., 2015) and AixLib (Müller et al., 2016).

The pipe elements of a district heating network are modeled using a dynamic, equation-based thermo-hydraulic pipe model, the so-called plug-flow pipe model developed by (van der Heijde et al., 2017). This section explains the underlying idea of the plug-flow pipe model and describes additional work to use this model within a system model. The plug-flow pipe model is based on a plug-flow approach, which enables the dynamic simulation of long pipes, including the pipe network of district heating and cooling networks. Besides the hydraulic behavior, heat losses, heat storage effects of the medium and the pipe as well as the propagation of temperature waves along the pipe can be simulated. In addition, the pipe model is usable for various network layouts (e.g. branched and meshed networks), thus enabling a wide range of applications in the field of dynamic simulation of DHC networks. For this purpose, the properties of the pipe, such as the length, diameter as well as the thermal and hydraulic properties of the pipes and the pipe insulation can be specified with physical parameters. In order to be able to represent the pressure losses of a pipe segment more accurately, the plug-flow pipe model features the parameter *fac*. This parameter allows to take into account the flow resistances of e.g. bends and thermal expansion joints for each pipe section individually. In order to simulate the heat losses more accurately, we added a model that represents the surrounding soil as a combination of cylindrical thermal resistors (R) and capacitances (C). These RC-combinations model cylindrical heat transfer and thermal storage effects in the pipe and the surrounding soil. The pipes are connected by thermal connectors to the inside of the cylindrical thermal capacities. On the outside, the undisturbed soil temperature is used for the heat loss calculation. In contrast to the static design methods, dynamic simulation models allow to take into account changing soil temperatures during the year. Therefore, the annual profile of the undisturbed soil temperature is calculated according to Florides et al. (Florides and Kalogirou, 2005).

The pipe model is used to model the network topology and thus the system model. In a real network there are pipes and pipe sections of different length. Whereas the dynamic behavior along long pipe sections is of great interest, the effects in shorter pipes play a less important role (e.g. connection to buildings). The *spatialDisribution()* operator is used in the plug-flow pipe model to calculate the advection of fluid through the pipe and thus the temperature propagation along the pipe. For reducing the complexity of the system model and thus the number of equations and the computation time, a pipe model without *spatialDistribution()* operator was developed based on the plug-flow pipe model. This simplified pipe model is used for short pipe length, see section 3.

In the system model, the substation models represent the heat consumption of the connected buildings. For this purpose, the heat demand profiles of the individual buildings serve as model input and can either be determined using dynamic building simulations or defined using measured values. The demand profiles are loaded with *Combi Time Tables* and assigned to the corresponding substations. The model parameters of the substation are the nominal return temperature on the one hand, and the minimum temperature difference between flow and return line on the other. These parameters are used in combination with the heat demand profiles to control the mass flow of the substations. For this purpose, the flow temperature at the inlet of the substation is measured with a temperature sensor. Based on the heat demand profile and the measured flow temperature, the required mass flow is calculated using the defined return temperature. Using this equation-based control approach, the substation ideally covers the heat demand of the building at every time step. By combining several substations in one system model, the total mass flow in the DHC network is determined.
In order to provide the mass flow with the required flow temperature, a heat source is used in the model of the central heat supply of the district heating system. For testing different flow temperature controls, the supply temperature of the district heating network is defined as model input. Constant supply temperatures but also temporal temperature profiles, such as outside temperature-dependent heating curves, are possible. In addition, the return pressure and the pressure difference between flow and return are parameters of the supply model. Using a mass flow sensor and temperature sensors that measure the flow and return temperatures, the heating power of the heat supply is calculated.

Figure 1 shows exemplary the structure of the system model of a simple district heating system with one heat supply, two substations and four pipes. The upper two pipes represent the flow line, the lower two pipes the return line of the district heating network. For one pipe, the connection to the RC-model of the surrounding soil and the connection between the RC-model and the undisturbed soil temperature are also shown. The heat demand profiles of the two substations are integrated with two *Combi Time Tables*. The inputs of the supply model (supply temperature and pressure drop) are defined as constant values in the example.

Workflow and Model Reduction

The used workflow for district heating grid simulations presented in this paper is divided into 4 steps:

- GIS data import
- Network topology optimization
- Model reduction
- Model export

Whereas, the first two steps (GIS data import and network topology optimization) is handled by an Open-Source tool called *uesgraphs* (Fuchs et al., 2016), the last two steps are handled by a tool called *uesmodels*. *Uesgraphs* handles information of district networks (electricity, heating, cooling and gas) as a graph with edges and nodes and is written in Python. In the context of this paper, we use a GIS network topology of a real network in Germany. This data, stored in a *Postgres SQL* database, is imported in *uesgraphs*, which can represent the related buildings and building substation as nodes, as well as pipes as edges. To each node and edge individual information are assigned, for example the address of the building or the diameter and the insulation standard of the pipe.

Figure 2 shows an exemplary cutout of the investigated district heating network, substations in green dots, network edges and nodes in red lines / dots and the central supply as a green dot with a red circle. Using all edges of the network as a representation of single pipes in the simulation model, it is obvious that the resulting simulation model will have a high complexity and a huge number of used components. Therefore we reduce and optimize the shown network topology to reduce the number of nodes and edges. This step, which is carried out on the graph,

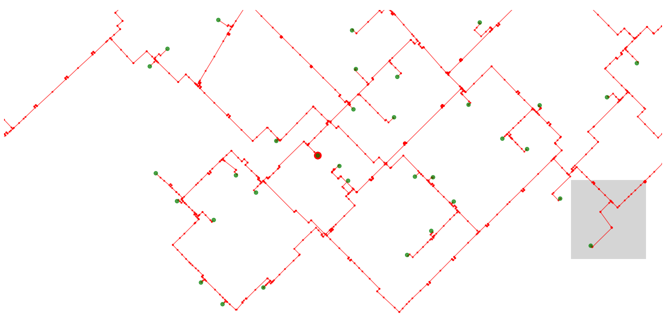


Figure 2. Original model cutout of an exemplary district heating grid

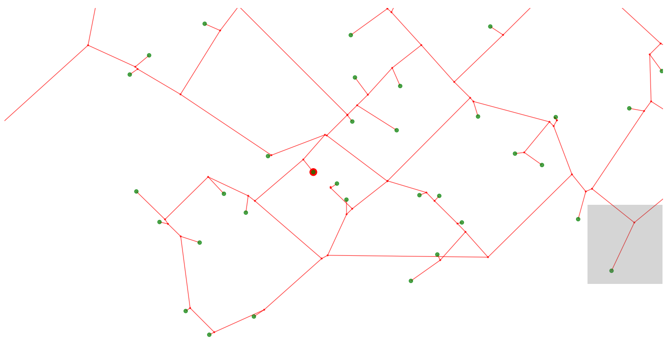


Figure 3. Simplified model cutout of an exemplary district heating grid

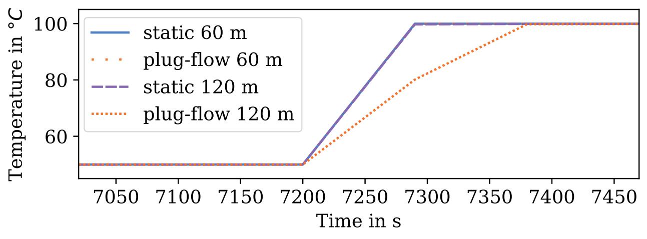


Figure 4. Step response to a temperature step of dynamic and static pipe model with different lengths

refers to the simplification of the network typology (Network topology optimization).

Figure 3 shows the exemplary result of the automated simplified network topology for the same cutout of the district heating grid. The simplified network contains much less pipe network representations by nodes and edges. This is mainly done by a weighted reduction of the existing pipes between nodes which represent junctions, substations or supplies. Therefore, all pipe edges between two network nodes which have more then two connecting edges are summarized and replaced by one pipe edge, representing the weighted diameter and the total length of the pipes reduced. An good example is the substation in the lower right corner, where three edges are simplified to one straight edge, representing the sum of the total length of the original three edges. In addition, the new representing edges summarize the pressure losses of the replaced pipe sections caused by bends and take these into account by increasing the already described parameter *fac* in the plug-flow pipe model.

In the two last steps, the network model and its features are further processed and exported. Therefore, a further model simplification is applied, to reduce simulation time and numerical complexity. In this case, the used Python tool *uesmodels* is able to replace all pipes below a certain length with a static pipe model instead of the IBPSA plug-flow pipe model, which reduces the simulation complexity through the reduction of the system of equations of the system model while causing only minor losses in accuracy. Regarding the impact of this replacement, responses on a temperature step on both models were investigated. Figure 4 shows the comparison of the plug-flow pipe model with the static pipe model, used for the replacement, on a temperature step of 50 K. Two different length with typical pipe diameters and mass flows are compared with a simulation output of 90s. One can see that with longer pipe length, the dynamic behavior of the two models are different, but with shorter pipes, in this case 60 meters of length, there is no difference observed. This leads to the conclusion that shorter pipe length can be replaced with static pipe models with no huge deviation in terms of the dynamic behavior. In addition, its shown that the steady state behavior before and after the temperature step is equal, resulting in a feasible assumption for the replacement.

Afterwards, the network graph and its features gets translated to *Modelica* code with the use of mako templates (Mayer, 2016). The resulting simulation model is ready to run in the simulation environment *Dymola*.

Control Strategies

The control of district heating and cooling networks has a big impact on the operation and efficiency of a district energy system. Especially for the integration of renewable energies into conventional heating and cooling networks as well as into 4th generation heating and cooling networks (Lund et al., 2014), the control of these networks is important because generation and demand are temporally decoupled (Vandermeulen et al., 2018). Since these control strategies and the resulting network operations are becoming increasingly complex, dynamic system models are important for developing and testing these strategies. In this paper, two different simple control strategies of a district heating network are exemplary examined and compared on the basis of dynamic simulations. For this purpose, a constant flow temperature is compared with a variable flow temperature that is dependent on the outdoor air temperature.

3 Use Case

The subject of the presented Use Case is the research center Jülich in Germany. The research campus consists of over 200 buildings with different usage, for example of-

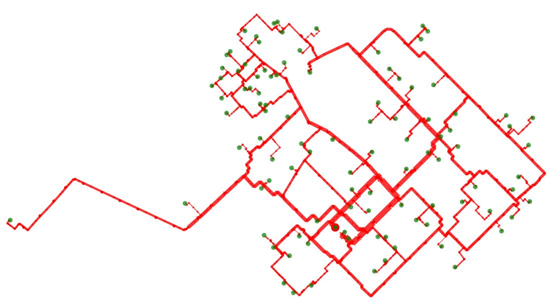


Figure 5. District heating network of the investigated research center in Germany

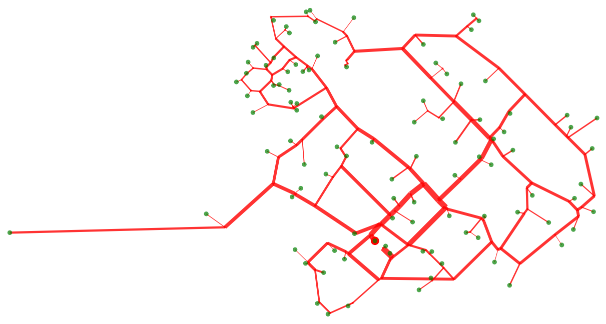


Figure 6. Simplified district heating network of the investigated research center in Germany

Table 1. *Modelica* translation and simulation information of original and reduced model

Model Characteristics	*Original Model*	*Reduced Model*	*Difference in %*
Number of equations	82527	81411	-1.4
Simulation Time in s	116920	64107	- 45.2
Number of Components	15475	15357	- 0.8
Number of state events	13551	10162	- 25.0
Number of Jacobian-evaluations	27855	22681	- 18.6

fices and laboratories. The building stock varies between different construction periods, with a peak between the year 1958 and 1968. Nearly all buildings on the research center are connected to a district heating network, which is supplied by waste heat of a nearby lignite-fired power plant. The annual peak load of around 28 MW are supplied with the help of more then 100 substations to the buildings for space heating, domestic hot water and process heat. Figure 5 shows the district heating network of the described Use Case. The green dots represents the substations for the buildings, the red edges the actual heating network and the green dot with the red circle the central supply unit. The network diameters are qualitatively indicated by the width of the red edges of the graph. The network itself has a total pipe length of 37 km and pipe widths between DN 32 and DN 400. The pipes are all plastic jacket pipes to ensure low heat losses while having high flow temperatures between 90 - 140 °C.

In the context of the presented paper, the Use Case shows the application of a dynamic simulation model with the presented methodology. Leading to a simulation model where the influence of different control strategies can be tested. This will be illustrated in this paper using examples of two different control strategies. In this case, the used geographical network information shown in Figure 5 are reduced with the methodology described in section 2, leading to a simplified, weighted network layout shown in Figure 6. Ones can see that the complexity in terms of the representing network edges is reduced, especially regarding the representation of the thermal expansion joints and exact directions of the pipe edges. The simplified representation is then translated to a dynamic model representation in *Modelica*.

Regarding the application in the context of this paper, different simulation exercises are present in section 4. Starting with the comparison and evaluation of the simulation model reduction in terms of complexity. Basis for this is the graph where the network topology optimization has already been applied. This graph is used to create a simulation model where all pipes are modelled with the plug-flow pipe model. For comparison a simulation model, which is simplified by the step of model reduction described in section 2 is automatically generated. The simplification is verified by a comparison of both simulation results.

The simplified model is used to compare two different control strategies for the heating network. A fixed temperature control strategy and a control strategy with variable flow temperature that is dependent on the outdoor air temperature are applied to present the capability and usability of the simplified dynamic simulation model to answer control strategy related questions of district heating grids.

4 Results

The presented Use Case is used to examine different simulations already described in section 3. Starting with the comparison of a complex model and a simplified model. Evaluating the translation characteristics as well as simulation time and results. Followed by a comparison of two control strategies to exemplary show the capability of the presented methodology and simulation models.

Model Reduction

The presented simulation model reduction takes the network topology optimization, known from Figure 6, as a starting point. Exporting the raw simplified network data to a simulation model, a *Modelica* system model with 100 substations and 258 pipes is created. Translating this model, in the following called *original* model, *Dymola* creates the translation information shown in Table 1. Using a standard workstation with 32 GB of RAM and 12 Cores for the simulation of this model, we are not able to create a stable and reproducible simulation. Replacing 59 pipes of the original 258 pipes with the static pipe model, a simulation model with the characteristics shown in Table 1 was created, following the described methodology. In this simplified case, *uesmodels* replaces all network pipes with a length shorter then 20 m with static pipes, while ensuring that a reduced pipe is not followed by another reduced pipe. Meaning that in the presented case, 64 pipes were shorter than 20 m but only 59 got replaced to ensure that no further error gets introduced by replacing a lot of connected pipes with static pipes.

Comparing the simulation log for this two models, a significant reduction in the number of state events as well as the number of Jacobian-evaluations is achieved. In addition a more stable and reproducible simulation was achieved on the same machine, while reducing the simulation speed by 45.2 %. All simulations were tested on different workstations with comparable hardware specifications, whereas only one of them was able to translate the *original model* without running into a bad allocation er-

Table 2. Results of the control strategy comparison

	Fixed Temperature	*Variable Temperature*
Temperature	120 °C	95 - 120 °C
Provided Energy	83.9 GWh	82.6 GWh
Heat Loss	9.9 GWh	8.6 GWh
Pumping Energy	45.0 MWh	61.5 MWh

rors. After the reduction, all of the testes machines could run the model without errors. Leading to the assumption, that with the current models and level of detail, the investigated network with its meshed structure is on the edge of possible thermal network simulations with the used computers. Nevertheless, the reduction needs to be compared in terms of actual simulation variables. Thus, we present a comparison of the *original* model with the *reduced* model by comparing their supplied heat. In addition, the calculated mass flow rates at the supply and the calculated return temperatures at the supply are compared with a time step of 15 minutes. The total supplied heat of the *original* model is 82.7 GWh, while the total supplied heat of the *reduced* model is 82.6 GWh, showing good accordance.

The comparison in Figure 7 and Figure 8 shows the difference between the both results. It is shown that the overall maximum difference in mass flow calculation is $+0.7\frac{kg}{s}/-1.97\frac{kg}{s}$ (total average mass flow: $55.2\frac{kg}{s}$), while the maximum difference in return temperature is $0.1K/-0.2K$ (total average return temperature: 60 °C). For a better visualization of the small error, Figure 9 shows the error as a histogram, underlining the small errors between the two models, with an expected value of $-0.00032K$. Nevertheless, ones can see e.g. in Figure 8, that there are some spikes and thus a deviation between the two compared simulations. The reason for this behavior could be the heat loss calculation of the static pipe model. Leading to smaller heat losses and smaller temperature drops on nearly zero mass flows compared to the plug-flow model. Keeping in mind that these spikes at the central supply are very small compared to the absolute average temperature values of 60 °C, we further investigated the deviations between the *original* and the *reduced* model. Figure 10 shows the mean value of the calculated deviations of the flow temperature for one year at all substations. It is shown, that the majority of the deviation is smaller then 1 K. Nevertheless, there are higher deviations which need to be further investigated in future work. In the scope of this paper, the *reduced* model compares good enough to the *original* model in terms of accuracy to further investigate top level control strategies, while improving overall stability and simulation speed.

Control Strategies

As already described in section 3, the presented district heating network is used to apply an exemplary comparison of two control strategies for district heating networks. In the context of this paper and based on the simplified simulation model, two simple control strategies are compared, showing the exemplary usage and capability of the workflow and the simulation models. One simulation is performed with a fixed flow temperature of 120 °C in the flow line of the heating network to fulfill the temperature requirements of the buildings on cold days. A second simulation is performed with a variable flow temperature curve based on the ambient temperature. The results for the overall supplied heat, the overall heat loss in the network and the used pumping energy are shown in Table 2.

On the one hand, comparing the total supplied heat, the variable flow temperature reduces the amount of heat use by 1.3 GWh due to lower heat losses in the network. On the other hand, regarding the used pumping energy, the variable flow temperature increases the energy demand by 16.5 MWh. Compared to the reduced amount of heat input, the slight increase of pumping energy is small, that the variable flow temperature should be preferred against the fixed flow temperature.

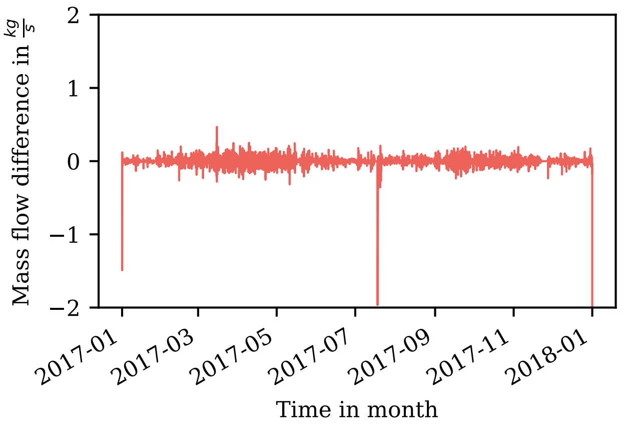


Figure 7. Mass flow difference between the results of the reduced and complex model at central supply unit

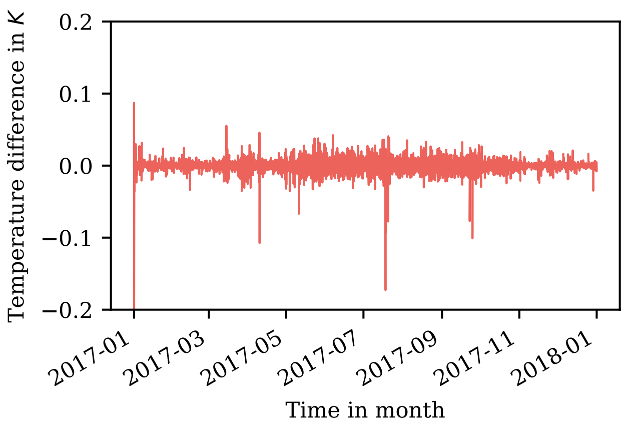


Figure 8. Return temperature difference between the results of the reduced and complex model at central supply unit

5 Conclusion and Limitations

The presented paper describes a workflow for automated generation and simplification of DHC network simulations with Python and Modelica. Two different type of simplifications are applied, on the one hand the network topology is optimized by reducing the total number of pipes in the model. On the other hand the model itself is reduced by using static pipe models for short pipe sections. The dynamic simulation models are based on internationally develop district heating and cooling components such as the IBPSA plug-flow pipe model. The automated workflow is applied to an existing district heating network of a research center in Jülich, Germany. The comparison of the simulation model reduction shows good accuracy between the *original* and *reduced* simulation model in terms of the simulation results, while reducing the simulation time by 45.2 %. To show the capability and usability of the presented workflow and simulation models, two different basic control strategies where examined, showing significant improvements for a variable flow temperature. The system model was not validated as part of this work. However, this is essential for the use of the model to optimize operation of DHC and will be done in future investigations. The paper shows the usability of the modeling language Modelica for DHC control optimization. Future work will include a comparison of the used models with even more simplified models, to elaborate which model detail is necessary for the evaluation of top level control strategies as well as the detail comparison of different models at different points at the network. Nevertheless, the paper present possible network simplification and their results of an Modelica user oriented view. However, future research needs to include the investigation of the impact on the numerical system, especially regarding the agebraic loops and blocks.

Acknowledgments

Grateful acknowledgement is made for financial support by BMWi (German Federal Ministry of Economic Affairs and Energy), promotional references 03ET1352A. This work was supported by the Helmholtz Association under the Joint Initiative “Energy System 2050 - A Contribution of the Research Field Energy”.

References

G. Florides and S. Kalogirou. Annual ground temperature measurements at various depths. In *8th REHVA World Congress, Clima, Lausanne, Switzerland*. 2005.

M. Fuchs, J. Teichmann, M. Lauster, P. Remmen, R. Streblow, and D. Müller. Workflow automation for combined modeling of buildings and district energy systems. *Energy*, 117:478–484, dec 2016. doi:10.1016/j.energy.2016.04.023. URL `https://doi.org/10.1016/j.energy.2016.04.023`.

Helge V Larsen, Benny Bøhm, and Michael Wigbels. A comparison of aggregated models for simulation and operational optimisation of district heating networks. *Energy Conversion and Management*, 45(7):1119 – 1139, 2004. ISSN 0196-8904. doi:https://doi.org/10.1016/j.enconman.2003.08.006. URL `http://www.sciencedirect.com/science/article/pii/S0196890403002140`.

H. Lund, S. Werner, R. Wiltshire, S. Svendsen, J. E. Thorsen, F. Hvelplund, and B. V. Mathiesen. 4th generation district heating (4gdh): Integrating smart thermal grids into future sustainable energy systems. *Energy*, 68:1–11, 2014.

M. Mayer. Mako templates for python, 2016. URL `http://www.makotemplates.org/`.

D. Müller, M. Lauster, A. Constantin, M. Fuchs, and P. Remmen. Aixlib-an open-source modelica library within the iea-ebc annex 60 framework. In *BauSIM 2016*, pages 3–9. 2016.

G. Schweiger, P.-O. Larsson, F. Magnusson, P. Lauenburg, and S. Velut. District heating and cooling systems-framework for modelica-based simulation and dynamic optimization. *Energy*, 137:566 – 578, 2017. ISSN 0360-5442. doi:https://doi.org/10.1016/j.energy.2017.05.115. URL `http://www.sciencedirect.com/science/article/pii/S0360544217308691`.

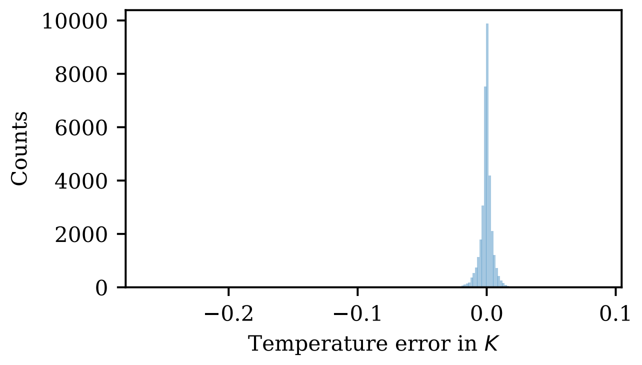


Figure 9. Histogram of temperature difference between the results of the reduced and complex model at central supply unit

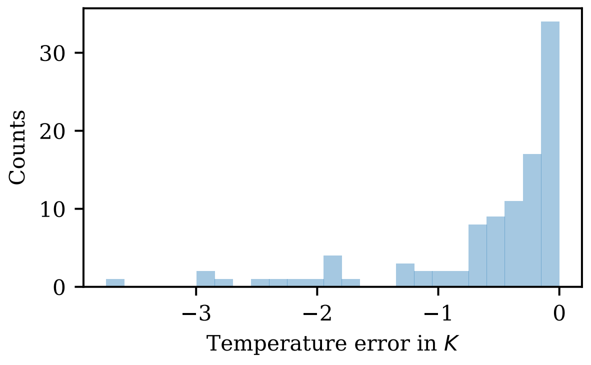


Figure 10. Histogram of the mean values of the quarter hourly deviations of one year for all substations

B. van der Heijde, M. Fuchs, C. Ribas Tugores, G. Schweiger, K. Sartor, D. Basciotti, D. Müller, C. Nytsch-Geusen, M. Wetter, and L. Helsen. Dynamic equation-based thermo-hydraulic pipe model for district heating and cooling systems. *Energy Conversion and Management*, 151:158 – 169, 2017. ISSN 0196-8904. doi:https://doi.org/10.1016/j.enconman.2017.08.072. URL `http://www.sciencedirect.com/science/article/pii/S0196890417307975`.

A. Vandermeulen, B. van der Heijde, and L. Helsen. Controlling district heating and cooling networks to unlock flexibility: A review. *Energy*, 151:103 – 115, 2018. ISSN 0360-5442. doi:https://doi.org/10.1016/j.energy.2018.03.034. URL `http://www.sciencedirect.com/science/article/pii/S0360544218304328`.

M. Wetter, C. van Treeck, and J. Hensen. New generation computational tools for building and community energy systems. *Energy in Buildings and Communities Programme. IEA EBC Annex*, 60, 2013.

M. Wetter, M. Fuchs, P. Grozman, L. Helsen, F. Jorissen, D. Müller, C. Nytsch-Geusen, D. Picard, P. Sahlin, and M. Thorade. Iea ebc annex 60 modelica library-an international collaboration to develop a free open-source model library for buildings and community energy systems. In *Proceedings of building simulation 2015*, 2015.

Session 2C: FMI 2

Non Linear Dimension Reduction of Dynamic Model Output

Claire-Eleuthèriane Gerrer[1] Sylvain Girard[1]

[1]Phimeca Engineering, France, {gerrer,girard}@phimeca.com

Abstract

Most advanced mathematical methods for the analysis of numerical model cannot cope with functional outputs of dynamic Modelica models. Principal component analysis is a well established method for dimension reduction, and can be used to tackle this issue. It relies however on a linear hypothesis that limits its applicability. We illustrate on a case study how the non linear method of auto-associative model overcomes this shortcoming and provides physically interpretable data representations.
Keywords: dimension reduction, functional data analysis, FMI, OtFMI, principal component analysis, auto-associative model, sensitivity analysis.

1 Introduction

The advent of the *functional mock-up interface* (FMI) and the emergence of associated tools considerably facilitated the analysis of Modelica models with advanced mathematical methods. Sensitivity analysis, model emulation, Bayesian inference and the like can now be performed routinely using scripting language such as Python (Girard and Yalamas; Girard et al., 2018) except for a substantial hurdle: many Modelica models are *dynamic*, and functional outputs are difficult to handle. Dimension reduction is a means to sidestep this difficulty. Principal component analysis (PCA) is by far the most prominent method for dimension reduction. This almost one century old statistical learning method (Hotelling, 1933) has been applied in virtually all fields where data are available. It is easy to implement and to understand, and relatively robust. It relies however on the hypothesis that the variables at hand can be aggregated into linear combinations, which unfortunately is not true for many dynamic model outputs. We illustrate this issue on a simple case study, and show how an alternative approach of more general applicability, the auto-associative model (AAM), allows to overcome it. Finally, we show how low dimension representations produced by AAM can be leveraged to get insights about the modelled physical phenomena.

2 Why reducing the dimension of dynamic model?

A computer experiment is the analysis of the output of a model obtained by varying its inputs according to a design of experiment. Modelica models are more often than not *dynamic*, namely their outputs are functions of time. Discretised time functions are high dimensional vectors which considerably obstructs the analysis. First, it is subjected to the "curse of dimensionality" (Houle, 2015), namely a variety of undesirable consequences of increasing the dimension. For instance, the volume of a cube increases exponentially with its dimension, and sample size required to densely fill it become quickly prohibitive. Distances in large dimension spaces loose their discriminating power, especially when the component variables are correlated, which is specially true for discretised time functions. Indeed, it is not straightforward to compare curves as it is with numbers. In statistical analysis, modelling the joint distribution of a set of more than 4 dependent variables, for instance using kernel estimation, is generally intractable.

Actually, the great majority of mathematical methods involved in computer experiments apply to models with scalar outputs. For instance, sensitivity analysis (Saltelli et al., 2008) aims at measuring the relative influence of the inputs on an output. Applying sensitivity analysis to each of them individually yields sensitivity indices that are functions of time: the output values at each chosen time step can be considered as distinct output variables. This approach to sensitivity analysis, sometimes deemed "*sequential*" (Girard, 2014, chapter 7) has its merits but is difficult to interpret.

Model emulation (also known as meta-modelling or surrogate modelling) is another technique that cannot cope with high dimensional outputs. It consists in substituting a CPU inexpensive mathematical approximation for a numerical model in order to achieve large sample size required for instance by some optimisation techniques, or for Bayesian parameter estimation, or to enable instantaneous interaction with the model. Kriging is an example of method for emulating numerical models (Roustant et al., 2012).

A common expedient to enable analysing functional outputs is to project them on a function basis (Campbell et al., 2006). When there is no obvious candidate, principal component analysis allows to automatically build an adapted basis.

3 Linear dimension reduction with principal component analysis

The geometric approach to PCA provides the most intuitive understanding of the method. The discretisation in d

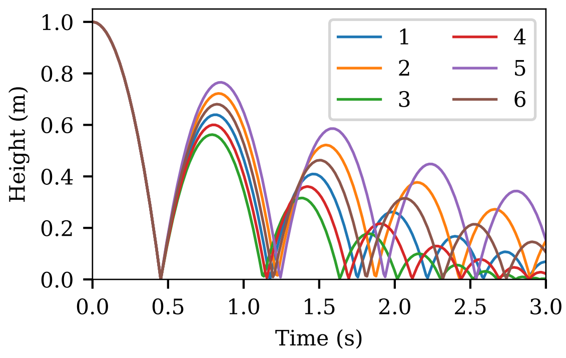


Figure 1. The first 6 trajectories of the training set when only the coefficient of restitution varies.

steps of N realisations of a functional model output can be seen as a point cloud of N points in $\mathbb{R}^d$. PCA then finds the axes along which data spread the most. These axes, called *principal directions*, have the property to minimise the distances between the points of the point cloud and their projection on the axes (Jolliffe and Cadima, 2016).

Each principal direction defines a linear combination of the initial d variables called *principal components*. The projection of the data points along the principal directions are called *scores*.

The principal directions of the data set are sequentially built, so as to be mutually orthogonal. The set of the principal directions form a new basis in the space $\mathbb{R}^d$. The k first principal directions, $k \in \{1, \ldots, d\}$, form a basis of the linear subspace of dimension k that best contains the scatter plot. Thus, PCA finds the linear subspace of given dimension (or hypercube, because the span of the data is usually limited) that best contains the point cloud.

3.1 PCA of the bouncing ball model

We applied PCA to a set of 128 trajectories of the famous bouncing ball model adapted by Tiller (2015)[1]: a ball is dropped from a given height and bounce back touching the ground with a fraction of the velocity it acquired during the fall determined by a fixed *coefficient of restitution*.

The trajectories were simulated with coefficient of restitution sampled between 0.7 and 0.9. following a Sobol' sequence (Sobol', 1979) so as to avoid redundancies. We used an FMU generated with OpenModelica, and the OtFMI Python module[2] (Girard, 2017) to carry out the simulations. Figure 1 displays the first 6 trajectories of this training set. All trajectories coincide before the first bounce at 0.45 s and increasingly deviate from one another at each subsequent bounce.

We simulated the next 896 ($= 1024 - 128$) trajectories of the Sobol' sequence to serve as a test sample for evaluating the performance of PCA. They were discretised at 300 evenly spaced time steps. Because the model has a single input, the intrinsic dimension of the set of trajectories, namely the smallest number of parameters required to fully parametrise it, is 1. The test trajectories were projected onto the first principal direction and compared to their original counterpart. The top panel of Figure 2 compares the worst reconstructed trajectory to the original. Here "worst" understands as resulting in the biggest root mean squared error (RMSE) between reconstruction and original. It must be noted however that the first 28 test trajectories sorted by decreasing RMSE are very similar to one another, as well as to those sorted by decreasing absolute error or relative error. Beyond that rank, the absolute error ranking diverges substantially from the two others. One principal component is clearly insufficient to capture the diversity of the trajectories: the reconstructed trajectory does not even touch the ground after the second bounce. Indeed, the middle and bottom panel show that the absolute and relative reconstruction errors with a single principal component are outsize. As expected, the error is null before the first bounces. It then displays a complex oscillatory pattern, ensuing from both the physical phenomenon and the sampling scheme. Interestingly, the absolute error globally increase as time goes by, despite the lessening of average height.

3.2 Time delays, a major stumbling block for PCA

What happens here is that the the point cloud of trajectories has a *linear* dimension much greater than 1. It is a one dimensional manifold extending in multiple directions in $\mathbb{R}^{300}$. As such, it cannot be "enclosed" in a line. Figure 3 illustrates the result of increasing the number of retained principal components (left panel). The decrease in all three error measures (absolute, relative and RMSE) is rather slow. For instance, a reduction to dimension 4 still results in a substantial number of relative errors greater than 50 %.

PCA attempts to catch the main temporal dynamics of functional outputs by linear combinations of the discretised values. Time shifts are non linear relationships involving time and an input variable. Fukunaga and Olsen (1971) illustrated this issue by considering a model whose output is a bump of fixed shape (they use a Gaussian bell curve) centred at variable time instants. In that case the principal directions spans the same vector space as the collection of bumps centred at each time step. Hence, the exact linear dimension grows with refinement of the time resolution of the discretisation. Non linear dimension reduction techniques are required to handle such situations.

4 Auto-associative models for non linear dimension reduction

The auto-associative model (AAM) proposed by Girard and Iovleff (2008)[3] approximates point clouds by implic-

[1]The adapted bouncing ball model is available at `http://book.xogeny.com/behavior/discrete/bouncing/`.

[2]`https://github.com/openturns/otfmi`.

[3]*Stéphane* Girard, not Sylvain.

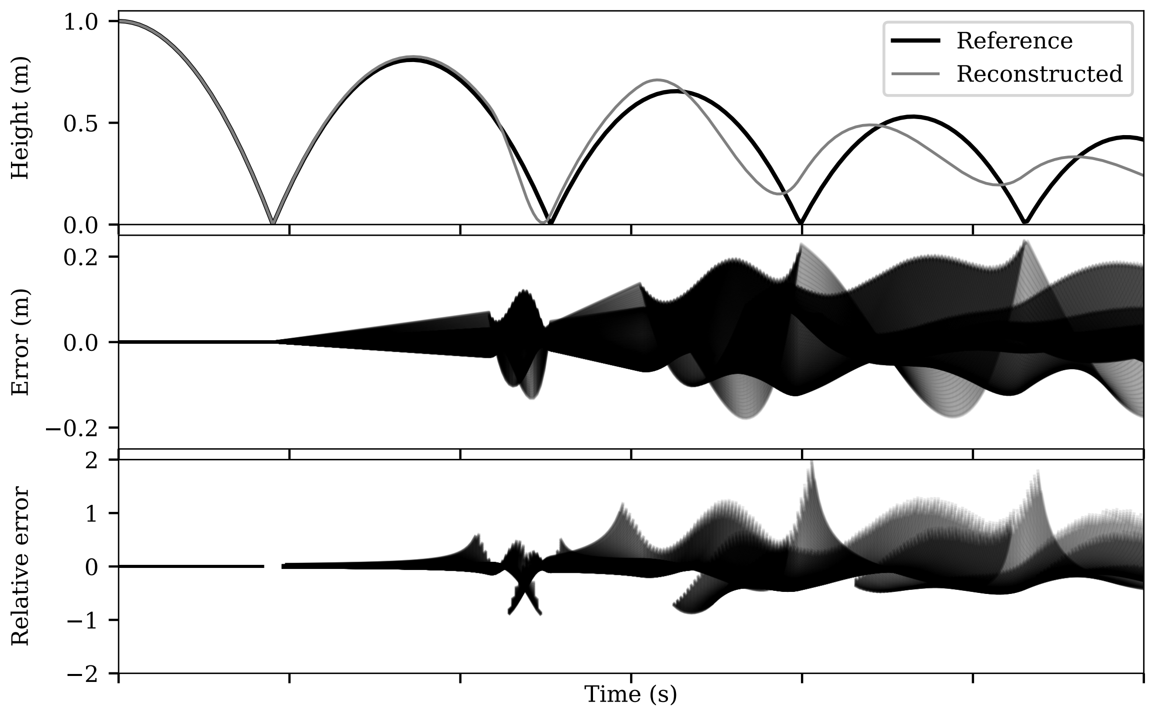


Figure 2. Reconstruction performance of PCA with a single principal component when only the coefficient of restitution varies. Top: comparison between original and reconstructed trajectories producing the worst RMSE. Middle: absolute reconstruction errors. Bottom: relative reconstruction errors. Intervals where the height was below 0.1 m were discarded. Lines are set to 0.1 opacity; darker tint indicate superposition of a large number of lines.

itly defined manifolds, instead of cubes like PCA does. It handles non linearity and can generally achieve reduction to dimension equal or close to the intrinsic dimension while preserving the fidelity of the reconstruction.

The algorithm for building AAM has 4 steps that are repeated until reconstruction is good enough:

1. *Direction computation* – A direction is computed by maximising an *index*, namely a function of the coordinates of the projection of the data points onto that direction. We used the index suggested by Girard and Iovleff (2008) that best preserves nearest neighbours.

2. *Projection* – The point cloud is projected onto the computed direction. The resulting coordinates will be called scores, by analogy with PCA terminology.

3. *Regression function estimation* – The regression function linking scores to the data points is estimated, here by spline regression.

4. *Update* – The point cloud from the current iteration is replaced by the *residuals*, namely the difference between data points and the output of the regression function estimated in step 3.

The algorithm terminates when the residuals are small enough. The final dimension is equal to the number of iterations.

PCA is a special case of auto-associative models where the regression functions are postulated to be linear. Its index is the variance of the projection of the point cloud. Its maximisation is equivalent to minimising the mean squared error between projections and data points. In that respect, it is a *global* index, contrary to the index we used for AAM based on nearest neighbour preservation, a local property.

4.1 AAM of the bouncing ball model

We fitted an AAM of dimension 1 on the same training set of 128 trajectories as before. We used a basis of 28 splines for the regression estimation. The number of splines was tuned manually, but this could be automatised for instance using cross validation.

Figure 4 illustrates the very good performance of the method. The worst reconstruction on the same test set as before is almost a perfect match, except for a tiny time delay and a blunting of the cusp at the last bounces. More than 90 % of the reconstructions have relative error below 10 % throughout the simulation, and more than 99 % of them have a maximal absolute error below 0.037 m.

Figure 3 shows that AAM performs better than PCA even if we keep a large number of principal components. In particular, the maximum absolute error of AAM is significantly smaller to that of PCA with 10 components.

Even better results were obtained in another similar experiment where the initial height, instead of the coefficient

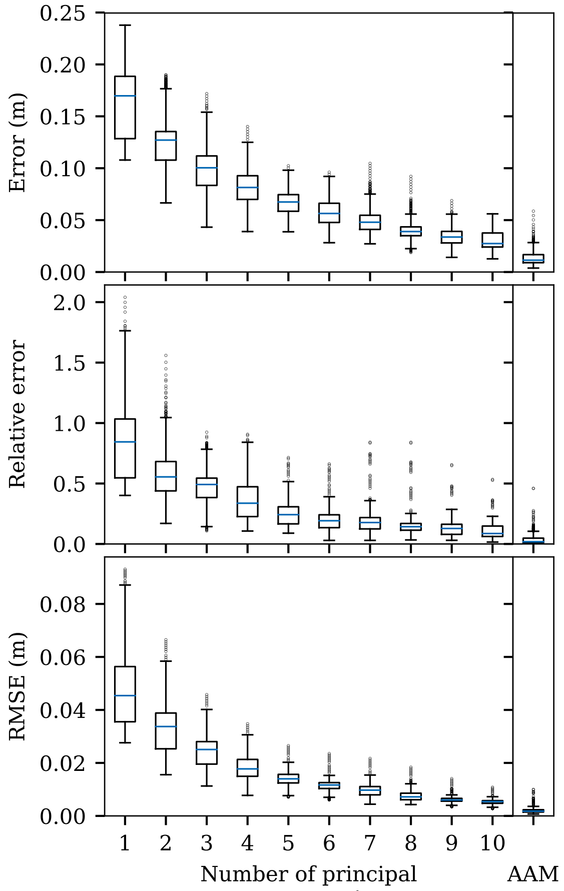


Figure 3. Distributions of time maximum absolute error, relative error and RMSE between test trajectories and reconstructions with increasing number of principal components (left), and reconstructions by a one dimensional AAM (right) when only the coefficient of restitution varies. Blue lines indicate the medians. Boxes span the interval between first, Q_1, and third, Q_3, quartiles. Whiskers reach the last data point above (resp. below) $Q_1 - 1.5 \times (Q_3 - Q_1)$ (resp. $Q_3 + 1.5 \times (Q_3 - Q_1)$). Dots are points outside the reach of the whiskers.

of restitution, varied (results not shown). In a third experiment, we simulated 512 training trajectories with both the coefficient of restitution and initial height varying, respectively between 0.7 to 0.9 and between 0.9 m to 1.1 m. Figure 5 shows the first 6 trajectories of this training set, whose size was augmented to $4096 - 512 = 3584$ trajectories. The effect of the two input variables combine: the higher the starting height, the higher the velocity at the first bounce. This is exemplified by the 5th trajectory (violet line) resulting from both high coefficient of restitution (0.875) and initial height (1.075 m): its second bounce is substantially away from the group of other trajectories (compare Figure 1). From visual inspection of the trajectories in Figure 1 and 5, we infer that the intrinsic dimension of the 2 input model is most likely equal to 2 because the two inputs have different effects on the output.

Figure 6 compares the performance of PCA with increasing number of principal components with that of AAM of dimension 1 and 2. Errors in reconstruction by PCA are globally much higher than in the single input experiment. Their distributions are leptokurtic (more "peaked") and positively skewed: there are a lot of important errors far away from the median and spanning a large interval. AAM performs not as good as in the single input experiment but is still much better PCA with 2 components, and roughly equivalent to PCA with 5 components.

4.2 Sensitivity analysis in AAM projection space

The gain in performance between the dimension 1 and 2 AAM, visible in Figure 6 (right panel), supports our guess that the intrinsic dimension of the model is 2. We confirmed that fact by analysing the sensitivity of the AAM scores to the coefficient of restitution and initial height. We computed first order and total Sobol' indices with the Monte Carlo algorithm proposed by Sobol' (2001) along with the "Jansen 1999" and "Saltelli 2010" estimators advocated by Saltelli et al. (2010).

The first AAM score is almost exclusively dependent on the coefficient of restitution (first order index: 94.2 %), with negligible interaction (second order joint index: 0.7 %). The second AAM score is dominated by the initial height (first order index: 78.5 %), with substantial contribution of the coefficient of restitution (first order index: 9.2 %), and interaction between the two (second order joint index: 12.2 %).

In order to interpret the physical meaning of these results, we reconstructed trajectories corresponding to locations evenly distributed along lines in the AAM projection plan. These "cross-sections" of the AAM plan space are shown in Figure 7. They illustrate what it means to have, say, "an average AAM first score and an high AAM second score". The first score mostly controls the bouncing instants. As a matter of fact, the middle plot of Figure 7 is pretty similar to Figure 1 showing the effect of the coefficient of restitution alone, which is coherent with the result of the sensitivity analysis stated above. The sec-

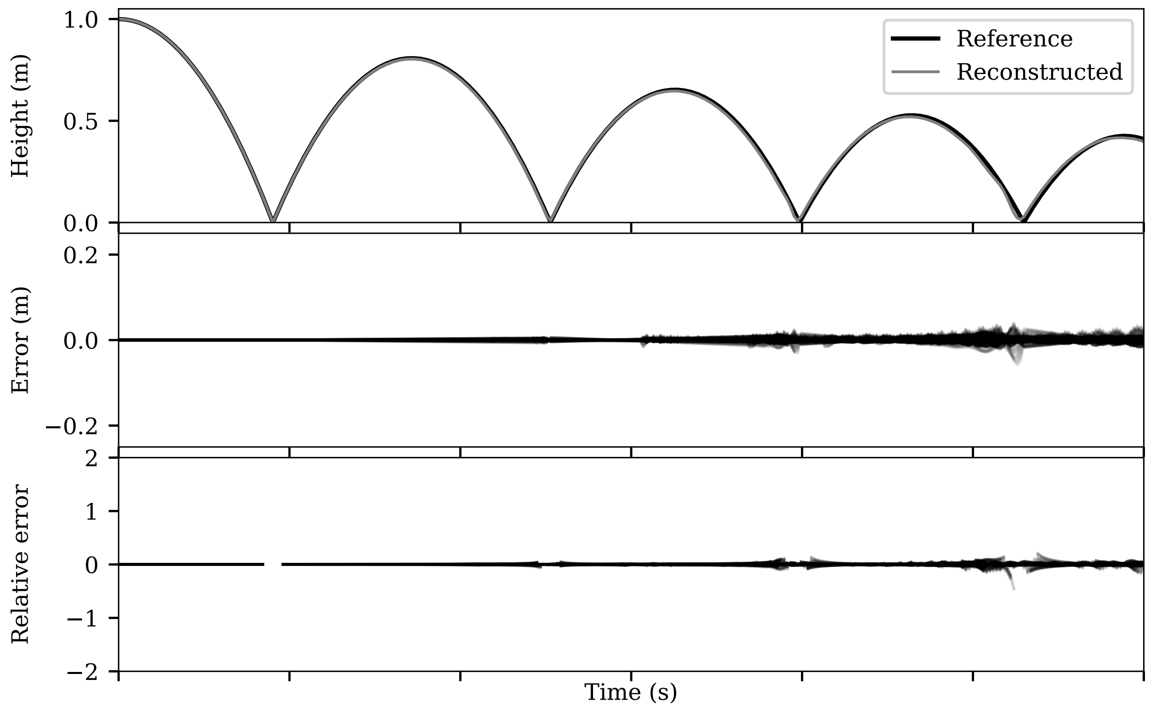


Figure 4. Reconstruction performance of a dimension 1 AAM when only the coefficient of restitution varies. Same graphical convention as in figure 2.

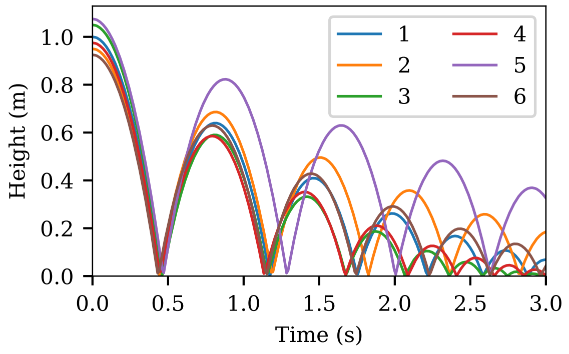


Figure 5. The first 6 trajectories of the training set with both the coefficient of restitution and initial height varying.

ond score affects the height of the peaks while keeping bouncing instants constant. It is similar to the effect of varying initial height alone (not shown), except that the latter alters bouncing instants. AAM actually automatically decomposed the influence of the input into a "time delay and damping" component, and a "height only" component. This level of legibility cannot be achieved with PCA whenever the linearity hypothesis does not hold.

It should be noted that the procedure detailed above is fully automatic. We treated the model as a black box, and did not took advantage of any insight about its physical or mathematical properties. This is particularly alluring as it forebodes routine usage by non specialists, and possible inclusion into graphical Modelica tools.

5 Conclusion and perspectives

Recent enrichments of the Modelica technological ecosystem enable straightforward implementation of advanced computer experiments with Modelica models. There remain however a major hurdle to overcome, namely adapting the panoply of mature mathematical methods to dynamic models with functional outputs. We showed on an example that linear dimension reduction with PCA may fall short of this objective, even for rather simple models. The recently developed non linear approach of AAM seems a very promising candidate to supplement, or even replace it altogether. It achieved very satisfying results on the presented case study and other more realistic ones not shown here. It is only little more complicated from the theoretical viewpoint, and almost as easy to use as PCA. "Degrees of freedom" in the algorithm are kept at a minimum, thus avoiding the need for elusive tuning skills.

Our implementation of the regression estimation is rather elementary. Hence, there is room for further performance enhancement. On the theoretical side, the question of how to define relevant metrics in the space of AAM scores is of great interest for sensitivity analysis or supervised importance sampling.

Acknowledgement

This work was partially supported by the Paris region through the FUI research project "Modeliscale", a collaboration with Dassault Systèmes, Inria, EDF, Engie, CEA INES, DPS, Eurobios and Phimeca Engineering.

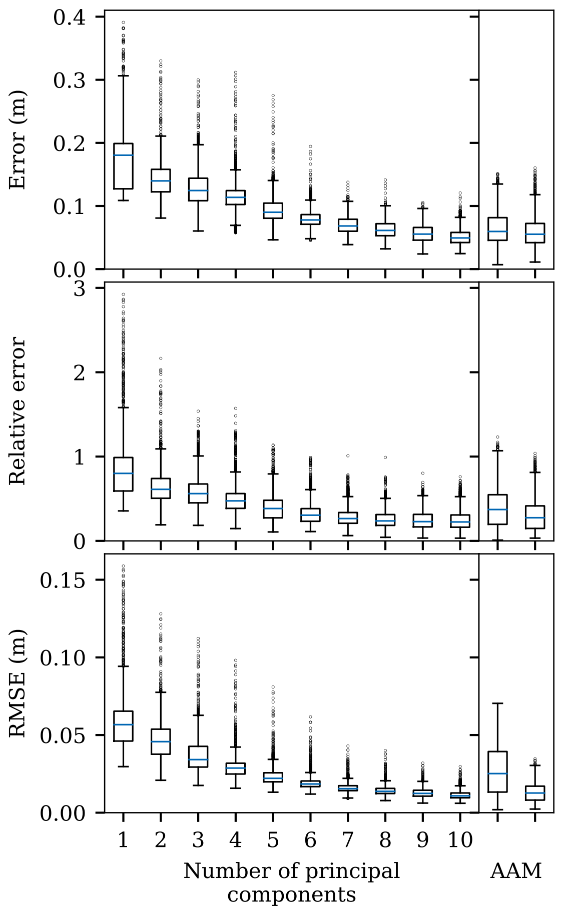


Figure 6. Distributions of time maximum absolute error, relative error and RMSE between test trajectories and reconstructions with increasing number of principal components (left), and reconstructions by a 1 and 2 dimensional AAM (right) with both the coefficient of restitution and initial height varying. Same graphical convention as in Figure 3.

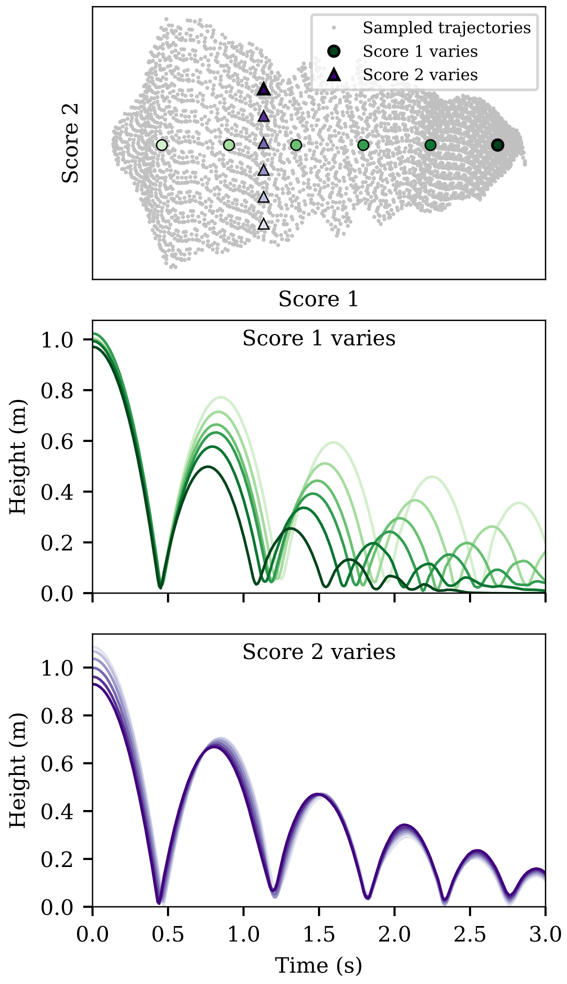


Figure 7. One dimensional cross-sections of the AAM projection space. Top: grey dots locate train and test trajectories in the AAM projection space; circles (resp. triangles) are located on arbitrary lines to illustrate the effect of varying the 1st (resp. 2nd) projection score. Middle (resp. bottom): reconstructed trajectories corresponding to the circle (resp. triangles) of same tint in the top plot.

References

Katherine Campbell, Michael D. McKay, and Brian J. Williams. Sensitivity analysis when model outputs are functions. *Reliability Engineering & System Safety*, 91(10-11):1468–1472, 2006.

Keinosuke Fukunaga and David R. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, C-20(2):176–183, feb 1971. doi:10.1109/t-c.1971.223208. URL `https://doi.org/10.1109/t-c.1971.223208`.

Stéphane Girard and Serge Iovleff. Auto-associative models, nonlinear principal component analysis, manifolds and projection pursuit. In *Lecture Notes in Computational Science and Enginee*, pages 202–218. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-73750-6_8. URL `https://doi.org/10.1007/978-3-540-73750-6_8`.

Sylvain Girard. *Physical and Statistical Models for Steam Generator Clogging Diagnosis*. Springer International Publishing, 2014. doi:10.1007/978-3-319-09321-5. URL `https://doi.org/10.1007/F978-3-319-09321-5`.

Sylvain Girard. Otfmi: simulate FMU from OpenTURNS: User documentation. Technical Report RT-PMFRE-00997-003, Phimeca, 2017.

Sylvain Girard and Thierry Yalamas. A probabilistic take on system modeling with modelica and python. URL `https://tinyurl.com/proba-system-model`.

Sylvain Girard, Thierry Yalamas, and Michael Baudin. Statistical learning and 0D/1D modelling: application to battery ageing. In *Lambda Mu 21 proceedings*. Institut de maîtrise des risques (IMdR), 2018.

Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

Michael E Houle. Inlierness, outlierness, hubness and discriminability: an extreme-value-theoretic foundation. *Technical Report NII-2015-002E*, 2015.

Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, mar 2016. doi:10.1098/rsta.2015.0202. URL `https://doi.org/10.1098/rsta.2015.0202`.

Olivier Roustant, David Ginsbourger, and Yves Deville. DiceKriging, DiceOptim: two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software*, 51(1):1–55, 2012.

Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global sensitivity analysis: the primer*. Wiley Online Library, 2008.

Andrea Saltelli, Paola Annoni, Ivano Azzini, Francesca Campolongo, Marco Ratto, and Stefano Tarantola. Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. *Computer Physics Communications*, 181(2):259–270, 2010.

Il'ya Meerovich Sobol'. On the systematic search in a hypercube. *SIAM Journal on Numerical Analysis*, 16(5):790–793, 1979. doi:10.1137/0716058. URL `http://dx.doi.org/10.1137/0716058`.

Il'ya Meerovich Sobol'. Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and Computers in Simulation*, 55:271–280, 2001.

Michael M. Tiller. *Modelica by Example*. Xogeny, 2015. URL `http://book.xogeny.com/`.

Relative Consistency and Robust Stability Measures for Sequential Co-simulation

Slaven Glumac[1] Zdenko Kovačić[2]

[1]AVL-AST d.o.o., Croatia, slaven.glumac@avl.com
[2]University of Zagreb Faculty of Electrical Engineering and Computing, Croatia, zdenko.kovacic@fer.hr

Abstract

The paper introduces a matrix co-simulation model of linear time invariant differential equations using general linear methods. This model is used to develop a calculation of relative consistency measure based on worst case defect calculation. It is shown how a robust stability measurement based on spectral radius can be used to measure robustness to slave parameter changes. Both consistency and stability measurements are calculated based on the linear model, and can be calculated prior to the co-simulation run. Finally, multi-objective optimization has been proposed to utilize introduced measurements for tuning the co-simulation master.

Keywords: co-simulation, master, robust stability, relative consistency, multi-objective optimization

1 Introduction

Co-simulation is a multi-method simulation of a coupled system, also known as simulator coupling (Kübler and Schiehlen, 2000). A co-simulation run can be quite difficult to set up in practice since models of the system are usually black boxes. When no previous information is available about co-simulation slaves, the procedure for the choice of a co-simulation master usually boils down to repeated trial and error. This is usually due to the fact that the system is more than the sum of its parts. The reason to co-simulate multiple simulators is to get the notion of the coupled system behavior. However, without any information about the behavior it is difficult to setup a co-simulation master.

In this paper co-simulation slaves are not completely black boxes. It is presumed that Jacobian matrices of slaves are available. (Åkesson et al., 2012). FMI 2.0 standard (FMI 2.0) defines an optional interface to Jacobian matrices of a slave. A linearized model is used to make the prediction of quality for a coupled system co-simulation. A goal of this paper is to introduce quality measures independent of internal states of the slaves.

Local error control is analyzed and shown to be a feasible method for bounding the global co-simulation error (Arnold et al., 2014). However, methods for local error estimation are usually expensive, e.g. Richardson extrapolation requires three times more co-simulation steps executed. There are predictor/corrector methods (Busch, 2012; Schweizer and Lu, 2015) which allow for run-time local error estimation as a side-effect. Defect control (Enright, 2000) presents an alternative to local error control. The defect control has been used to control the error of differential algebraic equations which makes it an ideal candidate for the use in co-simulation environments. This paper expand the idea of defect control by introducing a worst case defect calculation in order to enable a relative consistency estimate prior to the co-simulation run.

Stability is an important measure of system quality which does not depend on the initial states[1]. Zero-stability (Kübler and Schiehlen, 2000) is an important requirement for a co-simulation. However, a co-simulation cannot be more or less zero-stable, it can only be zero-stable or not. This leads to a search for a relative stability measure. In co-simulation there has been experimental work on determining stability regions for different co-simulation solvers (Busch, 2012; Schweizer et al., 2015). The authors have compared co-simulation masters based on the size of a plotted stability region. This paper tries to propose the use of stability radius estimate (Hinrichsen and Pritchard, 2005) in order to formalize this approach. Intuitively, robust stability is a particularly important property of a co-simulation. In practice, rapid prototyping is one of the main reasons for the use of co-simulation. During rapid prototyping, parameters or some parts of a single slave are expected to change. Robustness to such changes would mean that a user of a co-simulation does not have to adapt the master.

With quality measures defined the optimization becomes a feasible method for the choice of a co-simulation master. The optimization in co-simulation has been introduced as a means to improve the parameters of the simulated system in order to get a better signal response (Gedda et al., 2012). This paper introduces a problem of improvement of a co-simulation master as a multi-objective optimization problem (Kalyanmoy, 2001).

The next section introduces a co-simulation model used in this paper and underlying assumptions about co-simulation slaves. A presented matrix model of a co-simulation step enables the calculation of quality measures introduced in the following sections. The third section shows the example of modeling a two-mass oscillator. This example is used for the verification of quality mea-

[1]for linear systems

sures and demonstration of multi-objective optimization. The next section introduces a relative consistency and a robust stability measure. Both measures are minimized with the help of multi-objective optimization described in the fifth section. The section with conclusions and description of future work concludes the paper.

2 Co-simulation Model

2.1 Co-simulation Slave

A co-simulation slave according to the FMI 2.0 standard is a tuple:

$$G_i = (V_i, U_i, Y_i, D_i, v_i, set_i, get_i, doStep_i) \tag{1}$$

where V_i is a set of internal states of G_i, U_i is a set of input variables, Y_i is a set of output variables, $D_i \subset U_i \times Y_i$ is a set of output-input dependencies, v_i is an initial state of the FMU, set_i is a function which sets the value to an input variable, get_i is a function which returns the value of an output variable, and

$$doStep_i : (V_i, \mathbb{R}^+) \to V_i \tag{2}$$

is a function which implements a simulation step, i.e. it integrates the internal state[2]. Let v be the internal state, and t the local time of the slave. The function:

$$v' = doStep_i(v, h) \tag{3}$$

will change the internal state to v', and advance the local time of the slave to $t+h$.

In this paper it is assumed that the i^{th} co-simulation slave solves the following system of equations:

$$\dot{\mathbf{x}}_i(t) = \mathbf{A}_i \mathbf{x}_i(t) + \mathbf{B}_i \mathbf{u}_i(t) \tag{4a}$$

$$\mathbf{y}_i(t) = \mathbf{C}_i \mathbf{x}_i(t) + \mathbf{D}_i \mathbf{u}_i(t) \tag{4b}$$

where $\mathbf{x}_i : \mathbb{R}^+ \to \mathbb{R}^{|X_i|}$ is the state signal[3], $\mathbf{u}_i : \mathbb{R}^+ \to \mathbb{R}^{|U_i|}$ is the input signal, and $\mathbf{y}_i : \mathbb{R}^+ \to \mathbb{R}^{|Y_i|}$ is the output signal. A nonlinear differential equation:

$$\begin{aligned} \dot{\mathbf{x}}_i(t) &= \mathbf{f}_i(\mathbf{x}_i(t), \mathbf{u}_i(t)) \\ \mathbf{y}_i(t) &= \mathbf{g}_i(\mathbf{x}_i(t), \mathbf{u}_i(t)) \end{aligned} \tag{5}$$

can be transformed to (4) by linearization:

$$\mathbf{A}_i = \frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_i}, \quad \mathbf{B}_i = \frac{\partial \mathbf{f}_i}{\partial \mathbf{u}_i}, \quad \mathbf{C}_i = \frac{\partial \mathbf{g}_i}{\partial \mathbf{x}_i}, \quad \mathbf{D}_i = \frac{\partial \mathbf{g}_i}{\partial \mathbf{u}_i} \tag{6}$$

During co-simulation each slave performs the integration of its states (4a) in a $doStep_i$ function (2):

$$\begin{aligned} \mathbf{x}_i[k] &= \mathbf{x}_i(t_k) = \mathbf{x}_i(t_{k-1} + h) \\ &= \mathbf{x}_i(t_{k-1}) + \int_{t_{k-1}}^{t_k} \mathbf{A}_i \mathbf{x}_i(\tau) + \mathbf{B}_i \mathbf{u}_i(\tau) \mathrm{d}\tau \\ &= \mathbf{x}_i(t_{k-1}) + \int_{t_{k-1}}^{t_k} \mathbf{A}_i \mathbf{x}_i(\tau) + \mathbf{B}_i \mathbf{u}_i[k] \mathrm{d}\tau \end{aligned} \tag{7}$$

[2] The slave accepts any step size h, i.e. does not do a step rejection in case of an event.

[3] X_i is a set of internal state variables of the continuous system.

where h is the communication step size. This paper limits the analysis to zero-order hold as seen in the above equation, i.e. the value of input signal is assumed to be constant throughout a slave integration:

$$\tilde{\mathbf{u}}(\tau) = \mathbf{u}[k], \quad t_{k-1} < \tau \leqslant t_k \tag{8}$$

In this paper it is assumed that a linear model (4a) is integrated by a linear method:

$$\begin{aligned} \mathbf{v}'_i &= \mathbf{G}_i^{\mathbf{vv}} \mathbf{v}_i + \mathbf{G}_i^{\mathbf{vu}} \mathbf{u}_i \\ \mathbf{x}'_i &= \mathbf{G}_i^{\mathbf{xv}} \mathbf{v}'_i \end{aligned} \tag{9}$$

where $\mathbf{v}_i : \mathbb{R}^{|V_i|}$.

Integration of a linear system (4a) with any general linear methods can be formulated in the above manner. The derivation of matrices (9) can be done analogous to the derivation of the absolute stability matrix (Jackiewicz, 2009).

The function get_i should return the output values consistent to (4b):

$$\mathbf{y}'_i = \mathbf{C}_i \mathbf{x}_i + \mathbf{D}_i \mathbf{u}_i \tag{10}$$

This formulation is consistent with the mathematical model of co-simulation in the FMI standard (FMI 2.0, Section 4.1.2), although the definition of the k^{th} signal sample is a bit different, both for input signals (8) and the output signal:

$$\tilde{\mathbf{y}}(t_k) = \mathbf{y}[k] \tag{11}$$

In this paper the k^{th} signal sample refers to a last signal value sampled in the iteration of a co-simulation master. This is described in more detail in next sections.

2.2 Co-simulation Network

A network of co-simulation slaves is a tuple:

$$N = (G, \mathbf{L}) \tag{12}$$

where $G = \{G_1, G_2, \ldots G_n\}$ is a set of n co-simulation slaves, $U = U_1 \cup U_2 \cup \cdots \cup U_n$ is a set of input variables of all co-simulation slaves, and $Y = Y_1 \cup Y_2 \cup \cdots \cup Y_n$ is a set of output variables of all co-simulation slaves, $\mathbf{L} \in \mathbb{R}^{|U| \times |Y|}$ is a matrix representing output-input connections.

Let $\mathbf{u}(t)$, $\mathbf{y}(t)$, $\mathbf{x}(t)$ denote the column stacked values of all co-simulation slaves, respectively:

$$\mathbf{u}(t) = \begin{bmatrix} \mathbf{u}_1(t) \\ \mathbf{u}_2(t) \\ \vdots \\ \mathbf{u}_n(t) \end{bmatrix}, \quad \mathbf{y}(t) = \begin{bmatrix} \mathbf{y}_1(t) \\ \mathbf{y}_2(t) \\ \vdots \\ \mathbf{y}_n(t) \end{bmatrix}, \quad \mathbf{x}(t) = \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \\ \vdots \\ \mathbf{x}_n(t) \end{bmatrix} \tag{13}$$

Output-input connections are denoted as matrix $\mathbf{L}$:

$$\mathbf{u}(t) = \mathbf{L}\mathbf{y}(t) \tag{14}$$

Let system matrices of the co-simulation slaves (4) form a block diagonal matrix:

$$\begin{aligned}\mathbf{A} &= blockDiag(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n)\\ \mathbf{B} &= blockDiag(\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n)\\ \mathbf{C} &= blockDiag(\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n)\\ \mathbf{D} &= blockDiag(\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_n)\end{aligned} \tag{15}$$

The coupled system can be described by the following equations:

$$\dot{\mathbf{x}}(t) = \left(\mathbf{A} + \mathbf{BL}\left(\mathbf{I} - \mathbf{DL}\right)^{-1}\mathbf{C}\right)\mathbf{x}(t) \tag{16a}$$

$$\mathbf{y}(t) = \left(\mathbf{I} - \mathbf{DL}\right)^{-1}\mathbf{Cx}(t) \tag{16b}$$

An implicit assumption is that there are no algebraic loops in the system, i.e. $det\left(\mathbf{I} - \mathbf{DL}\right) \neq 0$. This system is described to give a reference for the exact solution of the co-simulation.

2.3 Co-simulation Master

A co-simulation master is an algorithm that executes a co-simulation of the network (12). The master presented in this paper (Algorithm 1) executes a non-iterative sequential co-simulation. It is determined by the communication step size h, extrapolation elements and the calling sequence of co-simulation slaves.

A calling sequence of co-simulation slaves determines the execution order of co-simulation slaves. Let the function $\sigma : I \to I$ define the calling sequence where $I = \{1, 2, \ldots, n\}$ is the set of slave indices. The calling sequence is defined by the following expression:

$$i = \sigma(r) \tag{17}$$

which states that the co-simulation slave G_i is executed r^{th} in the calling sequence.

The extrapolation is used to determine the values of continuous signal between[4] and beyond known communication points. This paper assumes the zero-order hold approximation of inputs during integration of co-simulation slave (8). The extrapolation elements of the co-simulation slave G_i are determined by the following equations:

$$\begin{aligned}\mathbf{w}_i' &= \mathbf{F}_i^{\mathbf{ww}}\mathbf{w}_i + \mathbf{F}_i^{\mathbf{wy}}\mathbf{y}\\ \mathbf{u}_i' &= \mathbf{F}_i^{\mathbf{uw}}\mathbf{w}_i'\end{aligned} \tag{18}$$

The extrapolation element can have access to all the outputs of all the slaves in the co-simulation network[5]. This allows for modeling of more advanced extrapolation techniques (Benedikt et al., 2013; Stettinger et al., 2014).

The assignment of time to a discrete signal value is done at the end of a co-simulation iteration. Let $\tilde{\mathbf{u}}(t)$ and $\tilde{\mathbf{y}}(t)$ denote the signals reconstructed from discrete samples provided by the co-simulation run.

In this paper the following equality holds:

$$\tilde{\mathbf{u}}(kh) = \mathbf{u}[k], \quad \tilde{\mathbf{y}}(kh) = \mathbf{y}[k] \tag{19}$$

This definition is important for consistency measurements introduced in the next sections.

Let the stacked values of the co-simulation system vectors be denoted as:

$$\mathbf{z} = \begin{bmatrix}\mathbf{x}^T & \mathbf{v}^T & \mathbf{y}^T & \mathbf{w}^T & \mathbf{u}^T\end{bmatrix}^T \tag{20}$$

This vector allows to model the co-simulation with a known model of linear time invariant co-simulation slaves (4), linear method of integration (9), and linear extrapolation methods (18). It allows to reformulate the equations of a co-simulation of black boxes (Algorithm 1) to a sequential multi-method integration of known models (Algorithm 2).

The integration of each co-simulation slave can be reformulated to:

$$\mathbf{z}' = \mathbf{G}_i\mathbf{z} \tag{21}$$

where equation (9) should be satisfied:

$$\begin{aligned}\mathbf{x}_i' &= \begin{cases}\mathbf{G}_i^{\mathbf{xv}}\mathbf{G}_i^{\mathbf{vv}}\mathbf{v}_i + \mathbf{G}_i^{\mathbf{xv}}\mathbf{G}_i^{\mathbf{vu}}\mathbf{u}_i, & j = i\\ \mathbf{x}_j, & j \neq i\end{cases}\\ \mathbf{v}_j' &= \begin{cases}\mathbf{G}_i^{\mathbf{vv}}\mathbf{v}_i + \mathbf{G}_i^{\mathbf{vu}}\mathbf{u}_i, & j = i\\ \mathbf{v}_j, & j \neq i\end{cases}\\ \mathbf{y}_j' &= \mathbf{y}_j, \quad \mathbf{w}_j' = \mathbf{w}_j, \quad \mathbf{u}_j' = \mathbf{u}_j\end{aligned} \tag{22}$$

The output update of a co-simulation slave can be reformulated to:

$$\mathbf{z}' = \mathbf{H}_i\mathbf{z} \tag{23}$$

where (10) should be satisfied:

$$\begin{aligned}\mathbf{y}_j' &= \begin{cases}\mathbf{C}_i\mathbf{x}_i + \mathbf{D}_i\mathbf{u}_i & , j = i\\ \mathbf{y}_j & , j \neq i\end{cases}\\ \mathbf{x}_j' &= \mathbf{x}_j, \quad \mathbf{y}_j' = \mathbf{y}_j, \quad \mathbf{w}_j' = \mathbf{w}_j, \quad \mathbf{u}_j' = \mathbf{u}_j\end{aligned} \tag{24}$$

The extrapolation elements can be reformulated to:

$$\mathbf{z}' = \mathbf{F}_i\mathbf{z} \tag{25}$$

where (18) should be satisfied:

$$\begin{aligned}\mathbf{x}_j' &= \mathbf{x}_j, \quad \mathbf{v}_j' = \mathbf{v}_j, \quad , \quad \mathbf{y}_j' = \mathbf{y}_j\\ \mathbf{w}_j' &= \begin{cases}\mathbf{F}_i^{\mathbf{ww}}\mathbf{w}_i + \mathbf{F}_i^{\mathbf{wy}}\mathbf{y}, & j = i\\ \mathbf{w}_j, & j \neq i\end{cases}\\ \mathbf{u}_j' &= \begin{cases}\mathbf{F}_i^{\mathbf{uw}}\mathbf{F}_i^{\mathbf{ww}}\mathbf{w}_i + \mathbf{F}_i^{\mathbf{uw}}\mathbf{F}_i^{\mathbf{wy}}\mathbf{y}, & j = i\\ \mathbf{u}_j, & j \neq i\end{cases}\end{aligned} \tag{26}$$

Such kind of reformulation allows for calculation of Algorithm 2 with the use of matrix operations. An iteration

[4] The term extrapolation is used loosely, it refers both to interpolation and extrapolation.

[5] Extrapolation matrices should be chosen with care to be consistent with the connection matrix **L** from (14).

Algorithm 1 Sequential co-simulation

```
Require: $N$, $\sigma$, $h$, $t_0$, $t_{end}$
  $k := 0$                                          ▷ Initialization phase
  $\mathbf{z} := \mathbf{R_v v}_0$
  while $\|\Delta\mathbf{z}\| > \varepsilon$ do
      for $r := 1$ to $n$ do
          $i = \sigma(r)$
          $\mathbf{w}_i := \mathbf{F}_i^{\mathbf{ww}}\mathbf{w}_i + \mathbf{F}_i^{\mathbf{wy}}\mathbf{y}$        ▷ Extrapolation
          $\mathbf{u}_i := \mathbf{F}_i^{\mathbf{ww}}\mathbf{w}_i$
          for $u_i \in U_i$ do
              $v_i := set_i(v_i, u_i, \mathbf{u}_{i[u_i]})$
          for $y_i \in Y_i$ do                          ▷ Read outputs
              $\mathbf{y}[y_i] := get_j(s_j, y_j)$
  $\mathbf{z}[0] = \mathbf{z}$
  while $t_0 + kh < t_{end}$ do                     ▷ Computation phase
      for $r := 1$ to $n$ do
          $i = \sigma(r)$
          $\mathbf{w}_i := \mathbf{F}_i^{\mathbf{ww}}\mathbf{w}_i + \mathbf{F}_i^{\mathbf{wy}}\mathbf{y}$        ▷ Extrapolation
          $\mathbf{u}_i := \mathbf{F}_i^{\mathbf{ww}}\mathbf{w}$
          for $u_i \in U_i$ do
              $v_i := set_i(v_i, u_i, \mathbf{u}_{i[u_i]})$
          $s_i = doStep_i(s_i, h)$                      ▷ Update internal state
          for $y_i \in Y_i$ do                          ▷ Read outputs
              $\mathbf{y}[y_i] := get_j(s_j, y_j)$
      $k := k + 1$
      $\mathbf{z}[k] := \mathbf{z}$                                  ▷ Signal assignment
```

of the computation phase of the algorithm can be reformulated as:

$$\mathbf{z}[k] = \mathbf{\Phi}\mathbf{z}[k-1] \tag{27}$$

The computation matrix $\mathbf{\Phi}$ can be calculated by tracing the steps of calculation phase of the algorithm:

$$\mathbf{\Phi} = \prod_{r=1}^{n} \mathbf{\Phi}_{\sigma(r)} = \mathbf{\Phi}_{\sigma(n)}\mathbf{\Phi}_{\sigma(n-1)} \cdots \mathbf{\Phi}_{\sigma(1)} \tag{28}$$

where $\mathbf{\Phi}_i$ represents the update of the overall co-simulation slave done by the co-simulation slave G_i:

$$\mathbf{\Phi}_i = \mathbf{H}_i\mathbf{G}_i\mathbf{F}_i \tag{29}$$

In this paper the initialization is done with the help of a fixed-point iteration. Initialization matrix $\mathbf{\Phi}_0$ is used to model an iteration of the initialization algorithm:

$$\mathbf{\Phi}_0 = \mathbf{\Phi}|_{\mathbf{G}_i = \mathbf{I},\ i \in I} \tag{30}$$

The initialization matrix is equal to computation matrix (27) with excluded solver updates[6] (function $doStep_i$), i.e. integration matrices $\mathbf{G}_i = \mathbf{I}$ equal to identity matrix. An iteration of the initialization phase of the algorithm can be reformulated as:

$$\mathbf{z}' = \mathbf{\Phi}_0\mathbf{z} \tag{31}$$

[6] A closer look at Algorithm 1 shows that the initialization and the computation phase are almost identical. The function call of $doStep_i$, i.e. the integration is missing in the initialization phase.

Algorithm 2 Sequential multi-method integration

```
Require: $N$, $\sigma$, $h$, $t_0$, $t_{end}$
  $k := 0$                                          ▷ Initialization phase
  $\mathbf{z} := \mathbf{R_v v}_0$
  while $\|\Delta\mathbf{z}\| > \varepsilon$ do
      for $r := 1$ to $n$ do
          $i = \sigma(r)$
          $\mathbf{z} := \mathbf{F}_i\mathbf{z}$                                ▷ Extrapolation
          $\mathbf{z} := \mathbf{H}_i\mathbf{z}$                                ▷ Read outputs
  $\mathbf{z}[0] = \mathbf{z}$
  while $t_0 + kh < t_{end}$ do                     ▷ Computation phase
      for $r := 1$ to $n$ do
          $i = \sigma(r)$
          $\mathbf{z} := \mathbf{F}_i\mathbf{z}$                                ▷ Extrapolation
          $\mathbf{z} := \mathbf{G}_i\mathbf{z}$                                ▷ Update internal state
          $\mathbf{z} := \mathbf{H}_i\mathbf{z}$                                ▷ Read outputs
      $k := k + 1$
      $\mathbf{z}[k] := \mathbf{z}$                                  ▷ Signal assignment
```

The initialization phase is started with a value assignment:

$$\mathbf{z} = \mathbf{R_v v}_0 \tag{32}$$

where:

$$\begin{aligned} \mathbf{v} &= \mathbf{v}_0 \\ \mathbf{x}_i &= \mathbf{G}_i^{\mathbf{xv}}\mathbf{v}_i \end{aligned} \tag{33}$$

need to be assigned to respective positions in the co-simulation value vector $\mathbf{z}$. The initialization phase of the algorithm can be modeled with:

$$\mathbf{z}[0] = \mathbf{\Phi}_0^{\infty}\mathbf{R_v v}_0 \tag{34}$$

where:

$$\mathbf{\Phi}_0^{\infty} = \lim_{m \to \infty} \mathbf{\Phi}_0^m \tag{35}$$

The conditions for the initialization phase to converge to the above limit are stated in the next sections.

3 Test System

3.1 Two-mass Oscillator

The example system is a two mass oscillator (Figure 1) The system consists of two co-simulation slaves, G_1 and G_2. Input variables of slaves G_1 and G_2 are:

$$U_1 = \{G_1.x, G_1.y\}, \quad U_2 = \{G_2.F\} \tag{36}$$

respectively. Output variables of slaves G_1 and G_2 are:

$$Y_1 = \{G_1.F\}, \quad Y_2 = \{G_2.x, G_2.y\} \tag{37}$$

respectively. The connection matrix is equal to:

$$\mathbf{L} = \begin{array}{c} \\ G_1.x \\ G_1.v \\ G_2.F \end{array} \begin{array}{c} \begin{array}{ccc} G_1.F & G_2.x & G_2.v \end{array} \\ \left[\begin{array}{ccc} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{array}\right] \end{array} \tag{38}$$

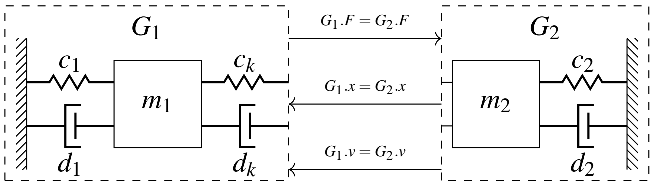


Figure 1. The example system used in the experiments throughout this paper is a two mass oscillator with force-displacement coupling. Slave G_1 provides force as an output, while slave G_2 provides the displacement and velocity of mass m_2.

Slave G_1 solves the following equations:

$$\mathbf{A}_1 = \begin{bmatrix} 0 & 1 \\ -\frac{c_1+c_k}{m_1} & -\frac{d_1+d_k}{m_1} \end{bmatrix} \quad \mathbf{B}_1 = \begin{bmatrix} 0 & 0 \\ \frac{c_k}{m_1} & \frac{d_k}{m_1} \end{bmatrix}$$
$$\mathbf{C}_1 = \begin{bmatrix} c_k & d_k \end{bmatrix} \qquad \mathbf{D}_1 = \begin{bmatrix} -c_k & -d_k \end{bmatrix} \tag{39}$$

Slave G_2 solves the following equations:

$$\mathbf{A}_2 = \begin{bmatrix} 0 & 1 \\ -\frac{c_2}{m_2} & -\frac{d_2}{m_1} \end{bmatrix} \quad \mathbf{B}_2 = \begin{bmatrix} 0 \\ \frac{1}{m_2} \end{bmatrix}$$
$$\mathbf{C}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{D}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{40}$$

Unless stated otherwise, the following system parameters are used in experiments throughout this paper:

$$\begin{aligned} &m_1 = 10, \quad c_1 = 1, \quad d_1 = 1, \quad c_k = 1, \quad d_k = 2 \\ &m_2 = 10, \quad c_2 = 1, \quad d_2 = 2 \end{aligned} \tag{41}$$

The co-simulation slaves use CVODE (Hindmarsh et al., 2005) solvers with a tight tolerance bound in order to provide a solution as close as possible to the exact one. For the purpose of analysis the integration of a co-simulation slave is assumed to be analytic (Hartman, 2002):

$$\begin{aligned} \mathbf{v}_i[k] &= e^{\mathbf{A}_i h}\mathbf{v}[k-1] + (e^{\mathbf{A}_i h} - \mathbf{I})\mathbf{A}_i^{-1}\mathbf{B}_i\mathbf{u}_i[k] \\ \mathbf{x}_i[k] &= \mathbf{v}_i[k] \end{aligned} \tag{42}$$

The integration matrices are defined as follows:

$$\mathbf{G}_i^{\mathbf{vv}} = e^{\mathbf{A}_i h}, \quad \mathbf{G}_i^{\mathbf{vu}} = (e^{\mathbf{A}_i h} - \mathbf{I})\mathbf{A}_i^{-1}, \quad \mathbf{G}_i^{\mathbf{xv}} = \mathbf{I} \tag{43}$$

where $i \in \{1, 2\}$.

3.2 Extrapolation

A zero-order hold (ZOH) element takes the last value and assigns it to the input:

$$\mathbf{F}_{i,ZOH}^{\mathbf{ww}} = \mathbf{0}, \quad \mathbf{F}_{i,ZOH}^{\mathbf{wy}} = \mathbf{I}, \quad \mathbf{F}_{i,ZOH}^{\mathbf{uw}} = \mathbf{I} \tag{44}$$

where $i \in \{1, 2\}$ and forms $\mathbf{F}_{i,ZOH}$ according to (26).

Nearly energy preserving coupling element (*NEPCE*) is an extrapolation element which tries to control the coupling error (Benedikt et al., 2013). Its implementation

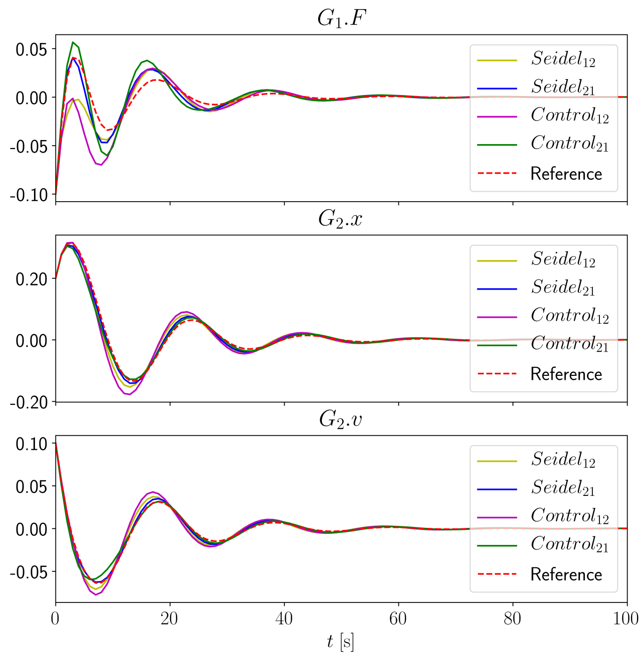


Figure 2. The plot shows output responses of four different co-simulation masters applied to the two-mass oscillator example system with co-simulation step size of $h = 1$.

consists of an implementation of an integral controller. The implementation in this paper is modeled as:

$$\begin{aligned} \mathbf{F}_{i,NEPCE}^{\mathbf{ww}} &= \boldsymbol{\alpha}_i \\ \mathbf{F}_{i,NEPCE}^{\mathbf{wy}} &= \mathbf{I} - \boldsymbol{\alpha}_i \\ \mathbf{F}_{1,NEPCE}^{\mathbf{uw}} &= \mathbf{I} \end{aligned} \tag{45}$$

where $i \in \{1, 2\}$ and forms $\mathbf{F}_{i,NEPCE}$ according to (26). It is interesting to note that a *ZOH* extrapolation element belongs to a subset of *NEPCE* extrapolation elements since:

$$\mathbf{F}_{i,ZOH} = \mathbf{F}_{i,NEPCE}|_{\boldsymbol{\alpha}_i = \mathbf{0}} \tag{46}$$

In later sections it is shown that tuning of *NEPCE* parameters can make a co-simulation more robust to parameter changes of the simulated system.

3.3 Sequential Co-simulation

Table 1. Co-simulation Masters

Master	σ	F_1	F_2
$Seidel_{12}$	σ_{12}	$\mathbf{F}_{1,ZOH}$	$\mathbf{F}_{2,ZOH}$
$Seidel_{21}$	σ_{21}	$\mathbf{F}_{1,ZOH}$	$\mathbf{F}_{2,ZOH}$
$Control_{12}$	σ_{12}	$\mathbf{F}_{1,NEPCE}$	$\mathbf{F}_{2,ZOH}$
$Control_{21}$	σ_{21}	$\mathbf{F}_{1,ZOH}$	$\mathbf{F}_{2,NEPCE}$

A calling sequence (17) for a sequential co-simulation of two slaves can be either:

$$\sigma_{12}(r) = \begin{cases} 1, & r = 1 \\ 2, & r = 2 \end{cases} \tag{47}$$

or:

$$\sigma_{21}(r) = \begin{cases} 2, & r = 1 \\ 1, & r = 2 \end{cases} \tag{48}$$

In the experiments presented in this paper, four co-simulation masters specified in Table 1 are used. *Control* master is named after the use of an error controller, namely $\mathbf{F}_{i,NEPCE}$. It is interesting to note that *Seidel* master is a subset of *Control* master:

$$Seidel = Control|_{\boldsymbol{\alpha}=\mathbf{0}} \tag{49}$$

This fact stems directly from (46).

The signal response for each of them applied to (41) is presented in Figure 2. Through this paper the analysis of results has been done with the help of NumPy (Oliphant, 2015) and Matplotlib (Hunter, 2007). The slaves and a reference configuration is available at repository BenchmarkFMUs (Glumac, 2017).

4 Relative Quality Measurements of Co-simulation

The goal of this section is to introduce measurements for the quality of co-simulation which do not require multiple co-simulation runs. The information from the previous sections should form the basis for such measurements.

4.1 Requirements for Initialization Phase

This paper uses fixed-point iteration (34) as an initialization algorithm for the co-simulation (Algorithm 1). In order for this algorithm to converge to an initial state, matrix $\mathbf{\Phi}_0$ should be stable and have only eigenvalues with value 1 on the unit circle. Matrix $\mathbf{\Phi}_0$ is stable if its spectral radius is:

$$\rho(\mathbf{\Phi}_0) \leqslant 1 \tag{50}$$

and its eigenvalues on the unit circle are semi-simple (Elaydi, 1996). The stability alone is not enough for the initialization to terminate. The above stability constraint (50) still allows for bounded oscillations of the fixed-point iteration. In order to prevent them the following constraint should be satisfied:

$$|\lambda| = 1 \Rightarrow \lambda = 1 \tag{51}$$

i.e. the only eigenvalues on the unit circle should only have the value of 1. By using Jordan decomposition it can be seen that a sequence of matrices $\mathbf{\Phi}^k$ has eigenvalues moving on the unit circle whenever eigenvalues are different from $e^{j0} = 1$, i.e. they have bounded oscillations.

The consistent initialization is defined in (Andersson, 2016), i.e. equations (14) and (16b) should be satisfied after the initialization phase:

$$\mathbf{y}[0] = \mathbf{L}\mathbf{u}[0] \tag{52a}$$

$$\mathbf{y}[0] = (\mathbf{I} - \mathbf{D}\mathbf{L})^{-1}\mathbf{C}\mathbf{x}[0] \tag{52b}$$

Conditions (50), (51) and (52) are introduced as a prerequisite for relative consistency measurement presented in the next subsection.

4.2 Relative Consistency

A local error control is a valid procedure to bound the global co-simulation error (Arnold et al., 2014). However, a local error depends on the initial state of the system and needs to be evaluated during co-simulation. Similar holds for the defect (Enright, 2000) which is used as the basis for the method proposed in this section. The goal of the method is to calculate the measure for consistency of the co-simulation with knowing only the structure of the system.

A connection defect is defined as:

$$\delta[k] = \|\mathbf{y}[k] - \mathbf{L}\mathbf{u}[k]\| \tag{53}$$

This definition of consistency is valid for a non-iterative co-simulation and can be measured during co-simulation with little or no extra cost. This measurement is useful as an indication whether or not to repeat a co-simulation run. However, an estimate for a single step of the co-simulation would be useful prior to the co-simulation run. The expression for the exact value of all the co-simulation values in the k^{th} step can be calculated following (34):

$$\mathbf{z}[k] = \mathbf{\Phi}^k \mathbf{\Phi}_0^\infty \mathbf{R}_\mathbf{v} \mathbf{v}_0 \tag{54}$$

The defect in the same step can be calculated:

$$\delta[k] = \left\|(\mathbf{S}_\mathbf{y} - \mathbf{L}\mathbf{S}_\mathbf{u})\,\mathbf{\Phi}^k \mathbf{\Phi}_0^\infty \mathbf{v}_0\right\| \tag{55}$$

where $\mathbf{S}_\mathbf{y}$ and $\mathbf{S}_\mathbf{u}$ are selection matrices defined by the following equation:

$$\begin{aligned} \mathbf{y} &= \mathbf{S}_\mathbf{y}\mathbf{z} \\ \mathbf{u} &= \mathbf{S}_\mathbf{u}\mathbf{z} \end{aligned} \tag{56}$$

The defect (55) still depends on the initial solver state $\mathbf{v}_0$. In order to obtain a worst case estimate for defect in k^{th} step, a matrix norm induced (Lancaster and Tismenetsky, 1985) by vector norm should be employed:

$$\|\mathbf{M}\| = \sup_{\mathbf{v}_0 \neq \mathbf{0}} \frac{\|\mathbf{M}\mathbf{v}_0\|}{\|\mathbf{v}_0\|} \tag{57}$$

From the above definition it immediately follows that a defect (55) is bounded by:

$$\|\mathbf{M}\mathbf{v}_0\| \leqslant \|\mathbf{M}\|\,\|\mathbf{v}_0\| \tag{58}$$

This guarantees that minimizing the following relative consistency measure:

$$\delta(\mathbf{\Phi}) = \left\|(\mathbf{S}_\mathbf{y} - \mathbf{L}\mathbf{S}_\mathbf{u})\,\mathbf{\Phi}^k \mathbf{\Phi}_0^\infty\right\| \tag{59}$$

will bound the defect with respect to internal solver state. The norm used for all of the experiments in this paper is 1-norm $\|\,.\,\|_1$.

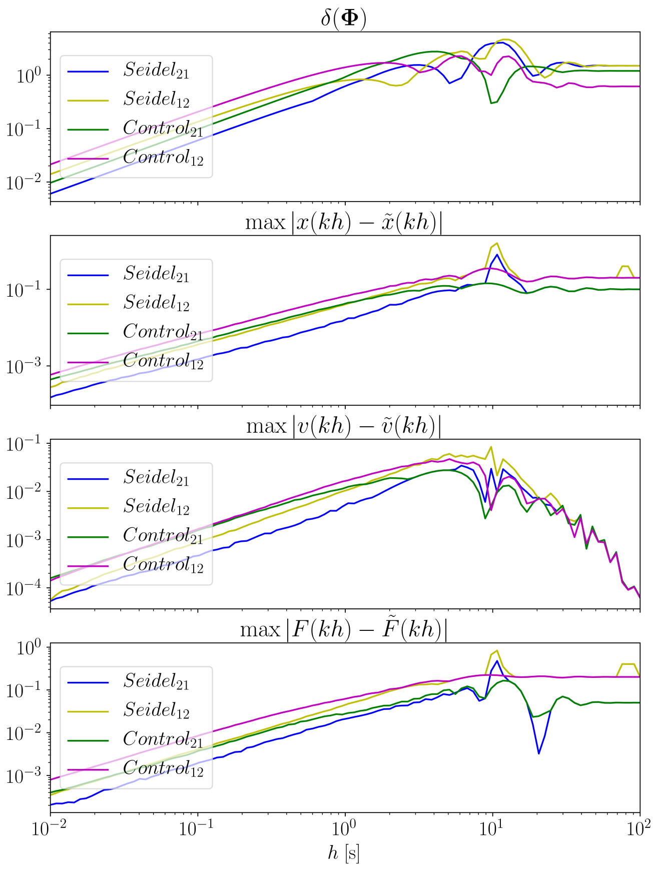


Figure 3. The figure compares the relative consistency measurement (first subplot from the top) and global errors of position, velocity and force outputs (second, third, and forth subplot from the top, respectively).

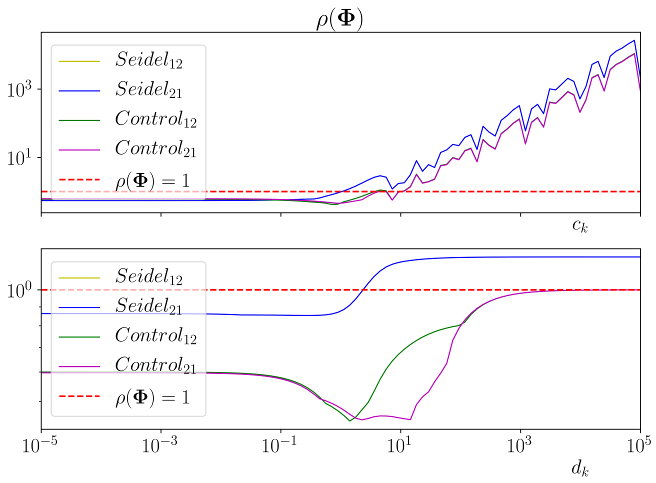


Figure 4. The figure shows the spectral radius of a two-mass oscillator co-simulation with respect to change of coupling stiffness c_k (upper plot) and coupling damping d_k (lower plot). The plots are used to indicate the size of stability region in terms of system parameters.

Figure 3 presents the analysis whether relative consistency provides a good indication of the global error. It can be seen that global error of each particular signal in the test examples follows the relative consistency measure. The experiment conducted gives confidence in using (59) as a minimization objective in order to improve the co-simulation master.

4.3 Robust Stability

The unstructured stability radius of a matrix (Hinrichsen and Son, 1989) is defined as the size of perturbations needed to bring the system to the verge of instability:

$$r(\mathbf{\Phi}) = \inf\{\|\mathbf{\Delta}\| : \rho(\mathbf{\Phi}+\mathbf{\Delta}) < 1\} \tag{60}$$

where $\rho(\mathbf{\Phi})$ is the spectral radius of a matrix defined as:

$$\rho(\mathbf{\Phi}) = \max_{\mathbf{\Phi v}=\lambda \mathbf{v}} |\lambda| \tag{61}$$

A stability radius calculates an important value for co-simulation. If a co-simulation master is set-up with some known linear time-invariant slaves, it is expected that this master is robust with respect to the changes of slave parameters. With a bigger stability radius the master is expected to work more reliably if a co-simulation slave is switched. In practice, this allows for a better prototyping of a complex system without a need to tune the co-simulation master.

In this paper a simplification of calculation has been considered. For the unstructured stability radius (60), the following relation holds (Hinrichsen and Son, 1989):

$$r(\mathbf{\Phi}) \leqslant 1-\rho(\mathbf{\Phi}) \tag{62}$$

The equality holds only for normal matrices. This is the reason of the assumption that the spectral radius can be considered a good robust stability measure[7].

In order to check whether a spectral radius (61) is a good pointer of a stability radius, the robustness of system (41) to change of stiffness c_k and damping d_k coefficients is analyzed. By inspecting the results visible in Figure 4, it can be seen that *Control* masters have larger stability intervals compared to *Seidel* masters, both with respect to stiffness:

$$\begin{aligned} \rho(Seidel_{12}) < 1, &\quad c_k \in [10^{-5}, 0.89] \\ \rho(Seidel_{21}) < 1, &\quad c_k \in [10^{-5}, 0.89] \\ \rho(Control_{12}) < 1, &\quad c_k \in [10^{-5}, 3.59] \\ \rho(Control_{21}) < 1, &\quad c_k \in [10^{-5}, 3.59] \end{aligned} \tag{63}$$

and damping:

$$\begin{aligned} \rho(Seidel_{12}) < 1, &\quad d_k \in [10^{-5}, 2.26] \\ \rho(Seidel_{21}) < 1, &\quad d_k \in [10^{-5}, 2.26] \\ \rho(Control_{12}) < 1, &\quad d_k \in [10^{-5}, 10^{5}] \\ \rho(Control_{21}) < 1, &\quad d_k \in [10^{-5}, 10^{5}] \end{aligned} \tag{64}$$

[7] This was done for simplicity of the exposition. In the work on stability radii there is a calculation of structured stability radius which will be a topic for future work.

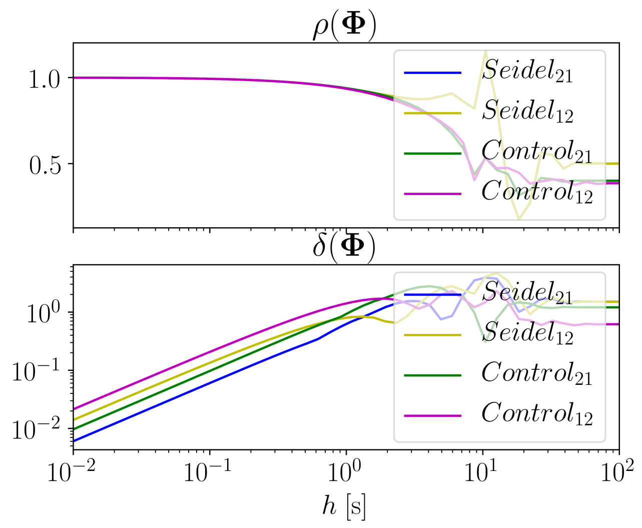


Figure 5. Comparison of the spectral radii (upper plot) and relative consistencies (lower plot) for different co-simulation masters with respect to the change of the communication step-size.

5 Optimal Choice of a Co-simulation Master

There can be multiple objectives defined for a co-simulation master. Two were defined in the previous section, the spectral radius (61) and the relative consistency (59):

$$J_1(\mathbf{\Phi}) = \rho(\mathbf{\Phi}) \\ J_2(\mathbf{\Phi}) = \delta(\mathbf{\Phi}) \tag{65}$$

In order to make an optimal choice of a co-simulation master with respect to the above objectives, multi-objective optimization (Kalyanmoy, 2001) should be employed. Multi-objective optimization is a technique for finding a non-dominated subset of solutions, i.e. the Pareto frontier $\mathscr{P}$:

$$\mathscr{P} = \{\mathbf{\Phi} \in \mathscr{S} : \neg(\exists \mathbf{\Phi}_{other}. \quad \mathbf{\Phi} \preceq \mathbf{\Phi}_{other})\} \tag{66}$$

The above definition states that solutions in the Pareto frontier $\mathscr{P}$ are not dominated by any other solution in the search space $\mathscr{S}$. A solution is dominated by another one $\mathbf{\Phi} \preceq \mathbf{\Phi}_{other}$ if it is worse or equal with respect to all objectives, and strictly worse with respect to at least one objective:

$$\begin{gathered} \mathbf{\Phi} \preceq \mathbf{\Phi}_{other} \\ \Updownarrow \\ J_1(\mathbf{\Phi}) < J_1(\mathbf{\Phi}_{other}) \wedge J_2(\mathbf{\Phi}) \leqslant J_2(\mathbf{\Phi}_{other}) \\ \vee \\ J_1(\mathbf{\Phi}) \leqslant J_1(\mathbf{\Phi}_{other}) \wedge J_2(\mathbf{\Phi}) < J_2(\mathbf{\Phi}_{other}) \end{gathered} \tag{67}$$

A multi-objective optimization is useful when objectives are conflicting [8], which seems to be the case in the studied problem (Figure 5). The figure presents the comparison of two optimization objectives on the search space defined by four co-simulation masters and the communication step-size h. The plots allow a choice of a step-size and a master with the estimation of trade-off. This is a form of brute-force multi-objective optimization as it samples all the search space and evaluates the objectives in each point. After the evaluation, a person is very likely to put some kind of goal on the first objective and search for the best value in the second one. This approach, however, is limited to a few parameters in the search space. In the case of Figure 5, only the communication-step size and the master type are varied. *Control* masters had been assigned control parameters before the experiment. The choice of control parameters is described in the rest of this subsection.

[8] Otherwise, a minimization of one objective would minimize the other. In this case a single-objective minimization could be more efficient.

Figure 6 shows the results of brute-force optimization of $Control_{21}$ co-simulation masters (Table 1). Unlike the previous approach, this one plots the Pareto frontier in the parameters space. The search space is defined as:

$$\mathscr{S} = \{Control_{21}|_{\alpha,h} : \alpha \in [0, 1.1], \\ h \in [10^{-3}, 10^1]\} \tag{68}$$

The parameter α is sampled uniformly in the search interval, while h is sampled uniformly in the log-scale of the search interval. The Pareto frontier is shown in the parameter space of the co-simulation master, i.e. both left-hand and right-hand heat-maps are spanned by the communication step-size h on the y-axis and the controller gain α on the x-axis. The shade of gray presents the value of an objective in a heat-map. The orange, yellow, and cyan mark points are the positions of the Pareto frontier solutions in the search space. The yellow color marks the $Seidel_{21}$ masters, while orange color marks the $Control_{21}$ masters with $\alpha \neq 0$. From (46), it immediately follows that a $Seidel_{21}$ master is a subset of possible $Control_{21}$ masters, i.e.:

$$Seidel_{21} = Control_{21}|_{\alpha=0} \tag{69}$$

A closer look at the Pareto frontier reveals that $Seidel_{21}$ masters (yellow points) dominate the rest of $Control_{21}$ for smaller communication step-sizes. The exception is a set of orange points, $Control_{21}$ masters at the top of the heat-maps. These points have smaller values of spectral radius (left heat-map) which corresponds to robustness to changes in system parameters (Figure 4). In addition, a communication step-size can be viewed as both a parameter and an objective. The bigger values of the communication step-size allow less burden for the communication network and allow more freedom for internal solvers to choose internal time-steps. In turn, this should probably decrease the CPU load. This is a rationale to prefer points with bigger values of communication step-size on the Pareto frontier. A smaller spectral radius and a bigger step-size are the reason to highlight one of the orange

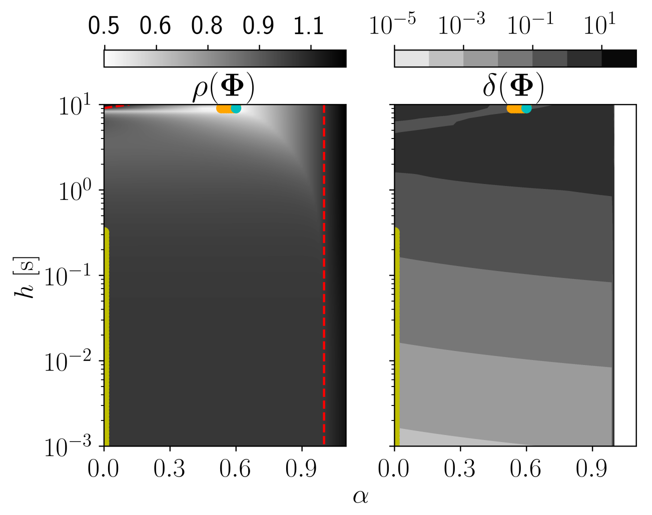


Figure 6. The Pareto frontier of $Control_{21}$ co-simulation is presented in the parameter space (α, h). The yellow points present $Seidel_{21}$ masters, while orange points present the rest of $Control_{21}$ masters. The cyan point represent $Control_{21}|_{\alpha=0.6,h=9.11}$ with objective values (70). The dashed red line represents a stability margin for the system, i.e. $\rho(\mathbf{\Phi}) = 1$.

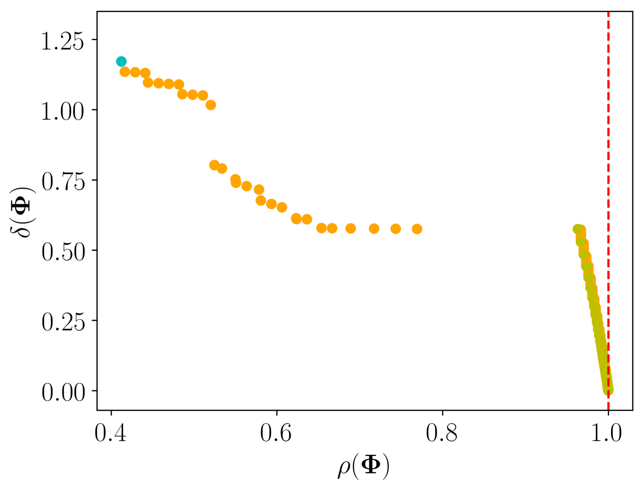


Figure 7. The Pareto frontier of $Control_{12}$ co-simulation is presented in the objective space (ρ, δ). The yellow points present $Seidel_{12}$ masters, while orange points present the rest of $Control_{12}$ masters. The cyan point represent $Control_{12}|_{\alpha_1=0.59,\alpha_2=0.36,h=8.9}$ with objective values (71). The dashed red line represents a stability margin for the system, i.e. $\rho(\mathbf{\Phi}) = 1$.

points as cyan ($\alpha = 0.6, h = 9.11$):

$$\begin{aligned} \rho(Control_{21}|_{\alpha=0.6,h=9.11}) &= 0.45 \\ \delta(Control_{21}|_{\alpha=0.6,h=9.11}) &= 0.78 \end{aligned} \tag{70}$$

This master has been used in previous images to present responses and objective values of all $Control_{21}$ masters.

The presentation in Figure 6 is feasible for search spaces with two real parameters. This is not the case for $Control_{12}$ masters which have three parameters α_1, α_2 and h. The Pareto frontier can be presented in the objective space (Figure 7). Again, the *Seidel* masters (yellow) seem to dominate other *Control* masters (orange) on a subset of the Pareto frontier. Orange points close to the yellow are also close in the parameter space, but they have $\alpha_1 > 0$. Again, the subset of *Control* masters (orange points on the left) has better stability properties for lower communication step-sizes. One of these points has been highlighted by cyan ($\alpha_1 = 0.59, \alpha_2 = 0.36, h = 8.9$):

$$\begin{aligned} \rho(Control_{12}|_{\alpha_1=0.59,\alpha_2=0.36,h=8.9}) &= 0.41 \\ \delta(Control_{12}|_{\alpha_1=0.59,\alpha_2=0.36,h=8.9}) &= 1.1 \end{aligned} \tag{71}$$

This master has been used in the previous images to present responses and objective values of all $Control_{21}$ masters.

The methods in this section are brute-force optimization techniques. They can easily become unfeasible as the search space scales poorly with the number of co-simulation slaves. The calling sequence (17) of co-simulation slaves is a permutation function, and as such the number of its configurations is factorial of the number of co-simulation slaves. However, the goal of this section is to demonstrate the use of objectives in choice of a co-simulation master. The formulation of objectives (65) enables the use of more advanced multi-objective optimization algorithms (Kalyanmoy, 2001) which may tackle high dimensional search spaces.

6 Conclusion and Future Work

This paper introduces relative consistency measure (59) and robust stability measure (61) of a co-simulation. Both measures have been used as objectives in multi-objective optimization (66) in order to increase the efficacy of tuning a co-simulation master.

The relative consistency measure is a worst case defect measurement with respect to unknown internal states of slaves. The experiments conducted show the global error follows the similar trend to the proposed consistency measure. This gives a suggestion that the relative consistency is a good measure to indicate how well a co-simulation master suites the coupled system it co-simulates. Since this measure is not dependent on an initial state, it gives the measure for any possible run of a linear system. This measure may be applicable for a non-iterative communication-step size control which may be one of the topics for future work.

The analysis in this paper has been restricted to sequential masters. Jacobi master does parallel updates of multiple co-simulation slaves which cannot be modeled with a sequential matrix multiplication. Parallel execution should be modeled as a single matrix. This matrix would align equations for *doStep* functions of multiple co-simulation slaves. The construction of such matrix should be a part of future work.

Since practical systems are usually not linear, a robust stability measure is introduced. It is argued that minimization of spectral radius (61) can be used to approximate unstructured stability radius (60). It is experimentally shown that a co-simulation with a lower spectral radius is more robust to changes of coupling stiffness and damping in the two-mass oscillator system. However, the spectral radius is only an approximation of the unstructured stability radius. Furthermore, it is not expected that the whole structure of the co-simulation is uncertain. The parameters of a co-simulation master are assumed to be known and certain, while the structured uncertainty should be concentrated on parameters of co-simulation slaves. For these reasons one of the main topics for future research will be to develop a calculation method of the structured stability radius (Hinrichsen and Pritchard, 2005) for the co-simulation.

References

Johan Åkesson, Willi Braun, Petter Lindholm, and Bernhard Bachmann. Generation of sparse jacobians for the function mock-up interface 2.0. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, pages 185–196. Linköping University Electronic Press, 2012.

Christian Andersson. *Methods and Tools for Co-Simulation of Dynamic Systems with the Functional Mock-up Interface*. PhD thesis, Lund University, 2016.

Martin Arnold, Christoph Clauß, and Tom Schierz. Error analysis and error estimates for co-simulation in fmi for model exchange and co-simulation v2. 0. In *Progress in Differential-Algebraic Equations*, pages 107–125. Springer, 2014.

Martin Benedikt, Daniel Watzenig, Josef Zehetner, and Anton Hofer. *NEPCE - A nearly energy-preserving coupling element for weak-coupled problems and co-simulation*, pages 1–12. ., 2013. ISBN 978-84-941407-6-1.

Martin Busch. *Zur effizienten Kopplung von Simulationsprogrammen*. kassel university press GmbH, 2012.

Saber N. Elaydi. *An Introduction to Difference Equations*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0-387-94582-2.

WH Enright. Continuous numerical methods for odes with defect control. *Journal of Computational and Applied Mathematics*, 125(1-2):159–170, 2000.

FMI 2.0. Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0. `https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf`, 2014. Accessed: 2018-10-26.

Sofia Gedda, Christian Andersson, Johan Åkesson, and Stefan Diehl. Derivative-free parameter optimization of functional mock-up units. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, pages 819–828. Linköping University Electronic Press, 2012.

Slaven Glumac. Benchmarkfmus. `https://github.com/sglumac/BenchmarkFMUs`, 2017. Accessed: 2018-11-04.

P. Hartman. *Ordinary Differential Equations*. Society for Industrial and Applied Mathematics, second edition, 2002. doi:10.1137/1.9780898719222. URL `https://epubs.siam.org/doi/abs/10.1137/1.9780898719222`.

Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

D. Hinrichsen and N. K. Son. The complex stability radius of discrete-time systems and symplectic pencils. In *Proceedings of the 28th IEEE Conference on Decision and Control,*, pages 2265–2270 vol.3, Dec 1989. doi:10.1109/CDC.1989.70573.

Diederich Hinrichsen and Anthony J Pritchard. *Mathematical systems theory I: modelling, state space analysis, stability and robustness*, volume 48. Springer Berlin, 2005.

John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

Zdzislaw Jackiewicz. *General linear methods for ordinary differential equations*. John Wiley & Sons, 2009.

Deb Kalyanmoy. *Multi objective optimization using evolutionary algorithms*. John Wiley and Sons, 2001.

R. Kübler and W. Schiehlen. Two Methods of Simulator Coupling. *Mathematical and Computer Modelling of Dynamical Systems*, 6(2):93–113, June 2000. ISSN 1387-3954. doi:10.1076/1387-3954(200006)6:2;1-M;FT093. URL `http://doi.org/10.1076/1387-3954(200006)6:2;1-M;FT093`.

Peter Lancaster and Miron Tismenetsky. *The theory of matrices: with applications*. Elsevier, 1985.

Travis E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition, 2015. ISBN 151730007X, 9781517300074.

Bernhard Schweizer and Daixing Lu. Predictor/corrector co-simulation approaches for solver coupling with algebraic constraints. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 95(9):911–938, 2015.

Bernhard Schweizer, Pu Li, and Daixing Lu. Explicit and implicit cosimulation methods: Stability and convergence analysis for different solver coupling approaches. *Journal of Computational and Nonlinear Dynamics*, 10(5):051007, 2015.

Georg Stettinger, Martin Horn, Martin Benedikt, and Josef Zehetner. *A model-based approach for prediction-based interconnection of dynamic systems*, pages 3286–3291. ., 2014. ISBN 978-1-4673-6088-3. doi:10.1109/CDC.2014.7039897.

Energy balance based Verification for Model Based Development

Kenji Sawada[1] Mamoru Sakura[2] Osamu Kaneko[3] Seiichi Shin[4]
Isao Matsuda[5] Toru Murakami[6]

[1]University of Electro-Communications, Japan, {knj.sawada[1],mamoru.sakura[2],o.kaneko[3],seiichi.shin[4]}@uec.ac.jp
GAIO Technology Co. Ltd., Japan, { matsuda.i[5], murakami.t[6]}@gaio.co.jp

Abstract

In model-based development (MBD), it is necessary to design a multiple physical model. Integration testing of such models is not easy because of its cross-sectional development. In this paper, we propose a new unit/integration test method “energy balance based verification” based on the law of energy conservation for MBD. The key idea is that the law of energy conservation will hold for no error models. The proposed method is composed of two diagrams. The first is a hierarchical diagram considering the type of energy. The second is an energy flow diagram based on the hierarchized diagram. Also, we develop model verification tools. Through the numerical experiments, we show that the proposed method and verification tools have the possibility of judging whether the model is normal or not. In the numerical experiments, we use a mild hybrid electric vehicle model that is developed via multiple CAE: MATLAB/Simlink®, MapleSim®, and IPG-CarMaker®.

Keywords: MBD, Model verification, Multiple CAE

1 Introduction

This paper focuses on a new model verification method for Model-Based Development (MBD) using cross-sectional tools. GAIO technology Co. Ltd. (GAIO) have pushed Model-Centered Development (MCD) which targets tool developments and services for MATLAB-based MBD and UML-based MDD (Model Driven Development) so far. Especially, MCD ver.1.0 concentrates on the controller models whose code is mainly described by MATLAB/Simulink. The next generation “MCD ver.2” targets not only the controller models but also the plant models. For example, MCD ver.2 targets co-simulation based on the plant model composed of various simulation tools such as MATLAB/Simulink, Mathematica, Maple/MapleSim, IMG-CarMaker and so on. This co-simulation results in the expansion of test area not only unit tests of Function Mockup Unit (FMU) but also integration tests of Functional Mockup Interface (FMI) (Blochwitz, Torsten, et al. 2012). So, GAIO needs to enter a new stage of model verification and have to introduce new test insights.

The accuracy of MBD depends on the accuracy of the model. Typically, the accuracy of the model is categorized by two. The first is the correctness of the program code which realizes the model. The second is the correcteness of the law of physics which is realized by the model. The former has been checked by typical program verification methods including unit tests and integration tests (Shokry, Hesham, et al. 2009, Rana, Rakesh, et al. 2013). The latter focuses on that ideal simulation models realize some physical laws. “The energy balanced based verification” method (Miyamoto et al. 2014) checks the energy balance of the model according to the fact that the law of energy conservation holds for no error models. That is, if the no error model has no internal loss energy, the total energy difference between the inputs and outputs will match the stored energy. Otherwise, the law of conservation does not hold. Even if the hybridization and electrification of automobiles make system structure complexity, the law of energy conservation itself does not change. This paper introduces the energy balance based verification as a new model test concept for MCD ver.2. This is the collaborative work between GAIO and the University of Electro-Communications (UEC).

This paper introduces a prototype system that streamlines the workflow of the energy balance based verification method and is composed of two verification tools. The method checks the input-output relation of each module consisting the model to calculate the energy quantity. In other words, the model expression considering the energy relation leads to efficient energy balance check of the model. Therefore, this paper proposes a hierarchical diagram and an energy flow diagram of the model. The former divides and categorizes the model according to the law of energy conservation. The category order is system, module, function, energy. The first verification tool is related to the hierarchical diagram. The latter expresses the energy flow relationship between the divided models. The second verification tool supports the energy balance based verification via the two diagram. To verify the validity of the proposed system, we consider the mild hybrid electric vehicle (MHEV) composed of MATLAB/Simulink, MapleSim, and IMG-CarMaker.

Further, we discuss the detectability of model bug using the developer tools.

2 Key Idea

2.1 Energy calculation

Energy calculation of the energy balance based verification is based on the numerical simulation of the model. We define the simulation time by

$$T_n = \Delta t \cdot n \tag{1}$$

where Δt is the sampling time and n is the sampled number. The simulation solver is the fixed step one. For example, the rotational kinetic energy is given by

$$\int_0^{T_n} \tau\omega dt \tag{2}$$

where torque τ[Nm] and rotational speed ω[rad/sec]. The translational energy is given by

$$\int_0^{T_n} Nvdt \tag{3}$$

where power N[N] and velocity v[m/sec]. The electric energy is given by

$$\int_0^{T_n} VIdt \tag{4}$$

where power voltage V[V] and current I[A]. The unit of energy is [J].

2.2 Energy evaluation policy

First, we consider four energy components: input energy, output energy, loss energy, and stored energy. At the time T_n, the sum of input energies for the model is

$$I_n = \sum_{k=1}^{k=M} i_{k_n} \tag{5}$$

where M is the number of input energy and i_{k_n} is the k-th entry of input energy at the time T_n. At the time T_n, the sum of output energies is

$$O_n = \sum_{k=1}^{k=M} o_{k_n} \tag{6}$$

where o_{k_n} is the k-th entry of output energy at the time T_n. At the time T_n, the sum of loss energy is

$$L_n = \sum_{k=1}^{k=M} l_{k_n} \tag{7}$$

where l_{k_n} is the k-th entry of loss energy at the time T_n. At the time T_n, the sum of stored energies is

$$C_n = \sum_{k=1}^{k=M} c_{k_n} \tag{8}$$

where c_{k_n} is the k-th entry of stored energy at the time T_n. If the model has an initial stored energy C_o, the following energy conservation law

$$C_n = C_0 + I_n - L_n - O_n \tag{9}$$

holds. If the simulation solver has the numerical error, the energy error

$$e_n = C_n - (C_0 + I_n - L_n - O_n) \tag{10}$$

is not equal to zero. In other words, if the acceptable error of the solver ε is guaranteed,

$$\max|e_n| < \varepsilon \tag{11}$$

holds for the model following the law of energy conservation. This is the first energy evaluation policy.

The second policy is the numerical error of the loss energy. The quantity of the loss energy is positive from the input-output energy. That is, if the acceptable error of the solver ε is guaranteed,

$$\min(l_{1_n}) > -\varepsilon, \min(l_{2_n}) > -\varepsilon, \dots, \min(l_{M_n}) > -\varepsilon \tag{12}$$

hold for all loss energy components.

The energy balance based verification the correctness of the model based on (11) and (12).

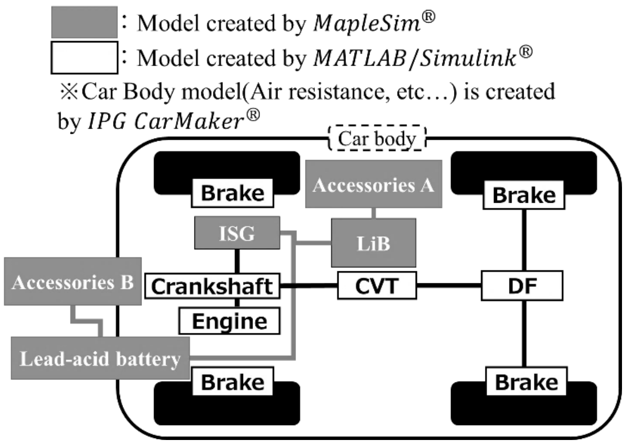
3 Hierarchical diagram

To calculate (11) and (12) efficiently, we introduce a hierarchical diagram. The diagram categorizes modules of the target models according to the following steps:

1. Separate electric systems, mechanical systems, and composite systems and label them S1, S2, …
2. Separate systems into modules and label them C1, C2, …
3. Separate modules into functions and label M1, M2, …
4. Separate functions into energies and label E1, E2, …

The category order is system, module, function, energy. Also, the diagram has three kinds of model information. The first is the energy calculation information in subsection 2.1. The second is the energy classification of each function in subsection 2.2. The third is the relation of connection between functions.

To show the concrete example of the hierarchical diagram, we show the MHEV model as shown in Fig. 1.



: Model created by MapleSim®
: Model created by MATLAB/Simulink®
※Car Body model(Air resistance, etc…) is created by IPG CarMaker®
Car body
Brake
Accessories A
Brake
ISG
LiB
Accessories B
Crankshaft
CVT
DF
Engine
Lead-acid battery
Brake
Brake

Figure 1. MHEV model.

This model is composed of MATLAB/Simlink®, MapleSim®, and IPG-CarMaker®. We apply step 1 and step 2 to the model, and we get its hierarchical diagram as shown in Fig. 2.

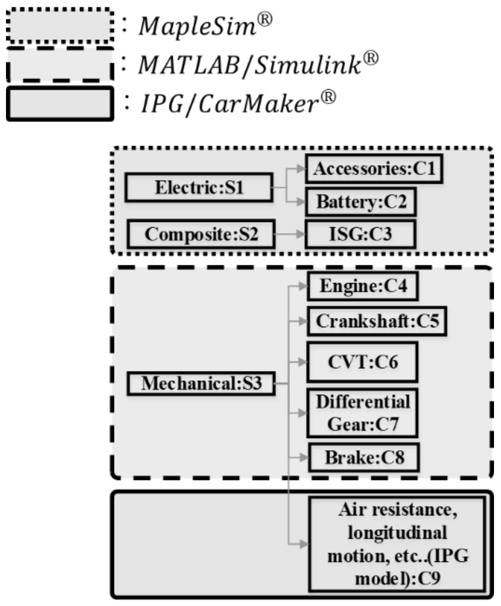


Figure 2. Hierarchical diagram of MHEV model

Fig. 3 is a part of the diagram obtained from step. 2 and step. 4.

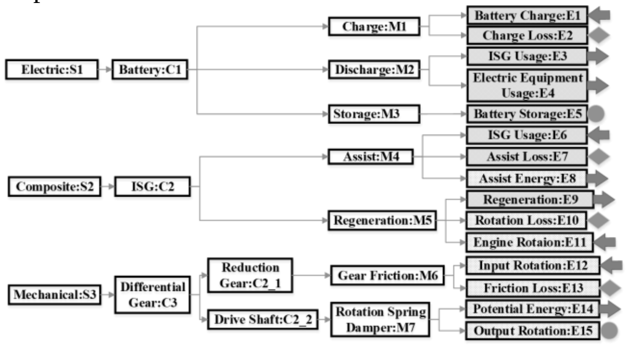


Figure 3. Detailed hierarchical diagram

In the diagram, the energy calculation information is expressed by the function block in Fig. 4.

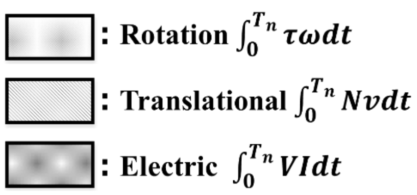

: Rotation $\int_0^{T_n} \tau\omega dt$
: Translational $\int_0^{T_n} Nvdt$
: Electric $\int_0^{T_n} VIdt$

Figure 4. Relationship between energy elements and formulas

The energy classification of each function is expressed by the mark as shown in Fig. 5.

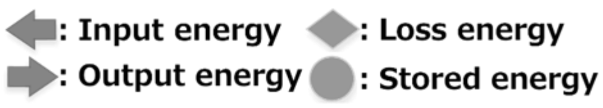


Figure 5. Energy classification in hierarchical diagram.

We design the diagram by *Microsoft Visio*®. Along with this, the relation of connection between functions is stored by each function block as meta information.

4 Energy flow diagram

The unit test of the energy balanced based verification checks (11) and (12) for each function. The integration test of the energy balanced based verification checks (11) and (12) for multiple functions. The former is corresponding to the test of FMU, and the latter is corresponding to the test of FMI. Both tests need the energy connection information. To carry out unit tests and integration tests efficiently, we introduce an energy flow (EF) diagram. The EF diagram visualizes the energy flow connection between functions and supports the energy calculations for multiple functions.

4.1 Diagram expression

We use a graph expression composed of place, transition, and arc in Figs. 6~10. The place is categorized into a unit test place, a loss place, an integration test place. Transition shows the testing or not. The arc shows the direction of the energy between function.

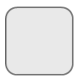


Figure 6. Unit test place

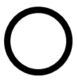


Figure 7. Loss place

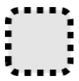


Figure 8. Integration test place

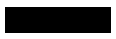


Figure 9. Transition (Non-testing)

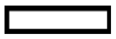


Figure 10. Transition(Testing)

We focus on C1 of Fig. 11 (deferential gear mode of MHEV) to show the EF diagram using place, transition, and arc.

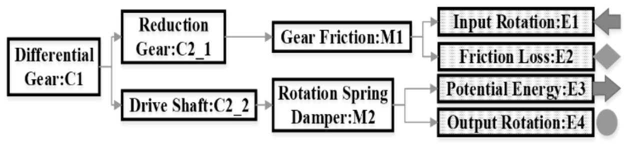


Figure 11. Hierarchy diagram of device C1

From Fig. 11, we obtain the energy relation:

$$E3_n = E3_0 + E1_n - E2_n - E4_n \tag{13}$$

Fig. 12 shows the energy flow transition of device C1. We see that the energy of the unit place changes from $E3_0$ to $E3_n$ and the energy of the loss place changes from 0 to $E2_n$. That is, we can apply (11) and (12) to each place as the unit test.

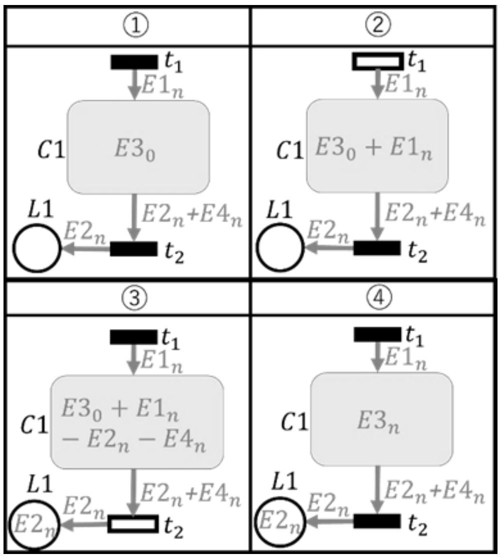


Figure 12. Energy flow transition of unit test of C1

To consider the integration test for multiple devices, we use the integration place in Fig. 8. The integration place connects unit tests as shown in Fig. 13. We consider transitions t1 and t2 as unit test A against function A, transitions t3 and t4 as unit test B against function B. p1 and p2 are for (11) and (12), and p4 is for (11) (unit test B has no loss energy). Place p3 connects function A with function B for the integration test. After unit tests A and B are carried out, the integration test checks the energy value of place p3 using (11).

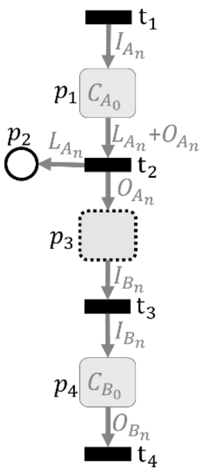


Figure 13. Energy flow between multiple functions.

4.2 Calculation via EF diagram

The EF diagram supports not only the visualization of unit and integration tests but also the energy calculation for (11) and (12). Consider the case of Fig. 13. The energy relation is given by

$$\begin{bmatrix} C_{A_n} \\ 0 \\ 0 \\ C_{B_n} \end{bmatrix} = \begin{bmatrix} C_{A_0} + I_{A_n} - (L_{A_n} + O_{A_n}) \\ L_{A_n} \\ O_{A_n} - I_{B_n} \\ C_{B_0} + I_{B_n} - O_{B_n} \end{bmatrix}. \tag{14}$$

The 1st, 3rd, 4th low of (14) are for equation (11). The 2nd low of (14) is for equation (12), We can extract (14) from the EF diagram by introducing the mathematical graph manipulation. Considering place set $P = \{p_1, p_2, \cdots, p_\alpha\}$ and transition set $T = \{t_1, t_2, \cdots, t_\beta\}$, we introduce the two matrices

$$\boldsymbol{B}^- = [b_{i,j}^-] \in \mathbb{R}^{\alpha\times\beta}. \tag{15}$$

$$\boldsymbol{B}^+ = [b_{i,j}^+] \in \mathbb{R}^{\alpha\times\beta}. \tag{16}$$

$b_{i,j}^-$ denotes the value of energy flow from place p_i to transition t_j. $b_{i,j}^+$ denotes the value of energy flow from transition t_j to place p_i. In this case, the connection matrix of the EF diagram is given by

$$\boldsymbol{B} = \boldsymbol{B}^+ - \boldsymbol{B}^- \in \mathbb{R}^{\alpha\times\beta}. \tag{17}$$

Also, we denote the energy values of place set $P = \{p_1, p_2, \cdots, p_\alpha\}$ as marking vector

$$\boldsymbol{m_n} = [m_n(p_1) \quad m_n(p_2) \quad \cdots \quad m_n(p_\alpha)]^{\mathrm{T}} \tag{18}$$

and we denote the testing or non-testing information of transition set $T = \{t_1, t_2, \cdots, t_\beta\}$ as the test vector

$$\boldsymbol{\omega_n} = [\omega_n(t_1) \quad \omega_n(t_2) \quad \cdots \quad \omega_n(t_\beta)]^{\mathrm{T}}. \tag{19}$$

We denote $\boldsymbol{m_0}$ as the initial energy values of place set $P = \{p_1, p_2, \cdots, p_\alpha\}$, we obtain the following relation:

$$\boldsymbol{m_n} = \boldsymbol{m_0} + \boldsymbol{B} \cdot \boldsymbol{\omega_n} \tag{20}$$

Equation (19) is corresponding to (13). In fact, from Fig, 14, we obtain

$$\boldsymbol{B}^- = \begin{bmatrix} 0 & L_{A_n} + O_{A_n} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & I_{B_n} & 0 \\ 0 & 0 & 0 & O_{B_n} \end{bmatrix} \tag{21}$$

$$\boldsymbol{B}^+ = \begin{bmatrix} I_{A_n} & 0 & 0 & 0 \\ 0 & L_{A_n} & 0 & 0 \\ 0 & O_{A_n} & 0 & 0 \\ 0 & 0 & I_{B_n} & 0 \end{bmatrix} \tag{22}$$

$$\boldsymbol{m_0} = [C_{A_0} \quad 0 \quad 0 \quad C_{B_0}]^{\mathrm{T}} \tag{23}$$

$$\boldsymbol{m_n} = [C_{A_n} \quad 0 \quad 0 \quad C_{B_n}]^{\mathrm{T}} \tag{24}$$

$$\boldsymbol{\omega_n} = [1 \quad 1 \quad 1 \quad 1]^{\mathrm{T}} \tag{25}$$

The matrix allows us to judge which low is for (11) or (12). If some row has all zero elements, the row is for the loss place and we apply (12) the corresponding row of (20). This expression is based on Petri net modeling (Murata 1989, Peterson 1981).

5 Numerical verification

The verification procedure based on energy balance is as follows:

I. Construct the hierarchical diagram of the target model.
II. Energy calculation via the simulation.
III. Construct the EF diagram based on the hierarchical diagram.
IV. Construct (11) and (12) based on (20).
V. Carry out the unit and integration tests.

We developed the prototype system automatically carries out from Step III to Step V: Tool α and Tool β. Tool α supports Step III and Tool β supports Step IV and V. We apply the tools to the MHEV model in Fig. 1. Fig. 14 shows the data flow between tools.

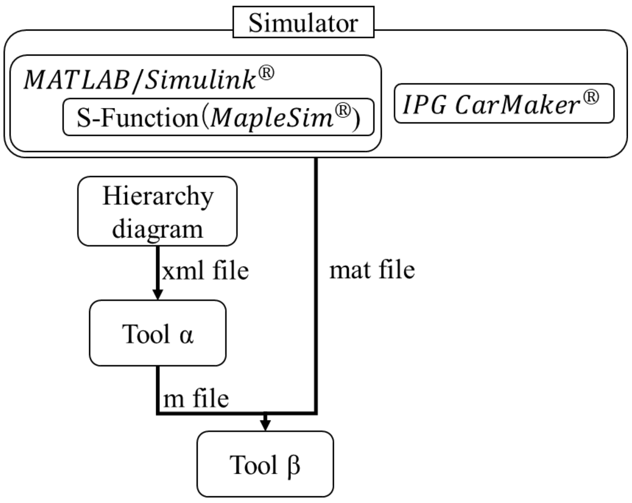


Figure 14. Data flow between tools.

In Step I, we construct the hierarchal model using Microsoft Visio to send XML files to Tool α. Tool α generates the connection matrix of the EF diagram for (20) and outputs m file of MATLAB. In Step II, The model of MATLAB/Simulink and the model of IPG CarMaker collaboratively via FMI. The simulation data is stored by mat file of MATLAB. Tool β receives m file and mat file and carries out unit and integration tests on MATLAB.

5.1 Verification

We consider a verification case where the MHEV model follows the energy conservation law. In Step II, we need to embed the energy calculation block in the target Simulink block as shown in Fig. 15. Fig. 16 shows the result of Tool β. If the model has no error, the tool outputs that the model follows the law of energy conservation.

Now, we are developing the auto-generation tool of the hierarchical diagram from the simulation model. For example, the Simulink model has XML data structure and then we can make effective use of XML. In this case, we embed energy classification information in the tag data of the Simulink file instead of embedding the energy calculation block.

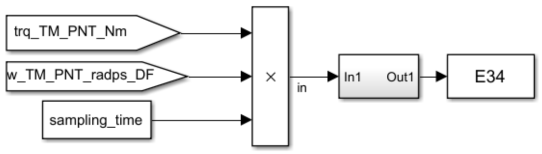
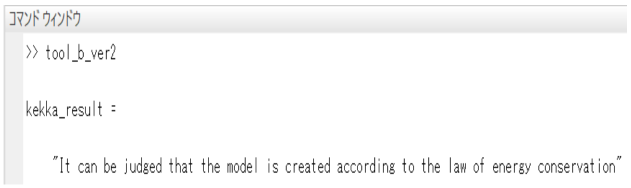


Figure 15. Energy calculation using Simulink.



```
コマンド ウィンドウ
>> tool_b_ver2

kekka_result =

    "It can be judged that the model is created according to the law of energy conservation"
```

Figure 16. Tool β result.

5.2 Bug detection

We consider the detectability of model bug using the developer tools. The model bugs of this paper are a mathematical bug and parameter bug. The former is that the equation expressing the model is wrong. The latter is that the parameter setting of the model is unrealistic. We consider the model bug of differential gear in Fig. 11.

First, we consider a mathematical bag. The following relation holds

$$G \cdot \tau_{in} = \tau_{out} \tag{26}$$

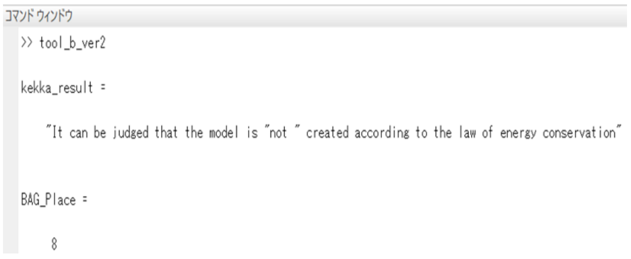
for gear ratio G, input torque τ_{in} and output torque τ_{out}. Also, we have the relation

$$\omega_{in} = G \cdot \omega_{out} \tag{27}$$

for input rotational speed ω_{in} and output rotational speed ω_{out}. Intentionally, we rewrite (27) as

$$G \cdot \omega_{in} = \omega_{out}. \tag{28}$$

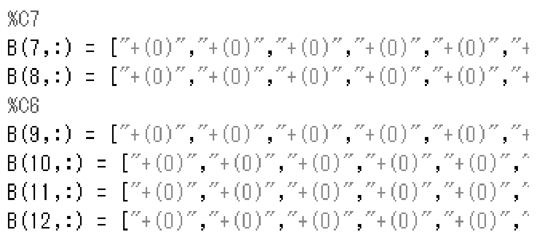
This bug causes that the output energy of deferential gear is bigger than the input energy and the model does not follow the law of energy conservation. Fig. 17 shows that the tool detects the model error.



```
コマンド ウィンドウ
>> tool_b_ver2

kekka_result =

    "It can be judged that the model is "not " created according to the law of energy conservation"


BAG_Place =

    8
```

Figure 17. Tool β result for error model.

"BUG_Place" shows the row number of the connection matrix (17) in which the law of energy conservation does not hold against the acceptable error of the solver. The corresponding information is also stored in the connection matrix of the EF diagram generated by Tool α. Fig. 18 shows the connection matrix for the MHEV model when m file by Tool α is carried out on MATLAB. The 8th row of the connection matrix belongs to C7.



```
%C7
B(7,:) = ["+(0)","+(0)","+(0)","+(0)","+(0)","+
B(8,:) = ["+(0)","+(0)","+(0)","+(0)","+(0)","+
%C6
B(9,:) = ["+(0)","+(0)","+(0)","+(0)","+(0)","+
B(10,:) = ["+(0)","+(0)","+(0)","+(0)","+(0)","
B(11,:) = ["+(0)","+(0)","+(0)","+(0)","+(0)","
B(12,:) = ["+(0)","+(0)","+(0)","+(0)","+(0)","
```

Figure 18. Connection matrix by Tool α.

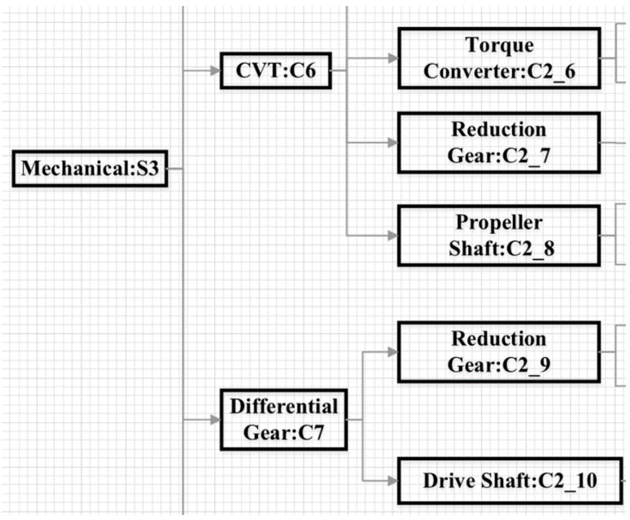


Figure 19. Hierarchical diagram focused on S3.

Fig. 19 shows the diagram focused on S3. We see that module C7 belongs to S3 and the tool detect the error of the differential gear.

Second, we consider a parameter bug. The loss torque τ_{loss} of the reduction gear (C2_9 in Fig. 19) satisfies

$$\tau_{loss} = S \cdot \tau_{in}. \tag{29}$$

where S is the loss rate and $0 \leq S < 1$. The initial value of MHEV is 0.02. In the bug parameter, we set -0.02. As a result, tool $\boldsymbol{\beta}$ detected this bug and identified the error of C7.

5.3 Discussion

The numerical validation shows that the developed tools support the energy balance based verification for multiple CAE and have detectability of model bug.

On the other hand, we need further discussion about the guarantee of verification standard. To guarantee the model accuracy via the tools, we need to guarantee the accuracy of energy calculation and energy classification on the hierarchical diagram and the EF diagram.

Also, we need to clarify what kind of model bug the energy balance based verification can detect. In subsection 5.2, the tools detected the bug module (C7 in Fig. 19), but, did not detect the bug function (C2_9 in Fig. 19). The detection accuracy depends on the structure of the connection matrix of the EF diagram generated from the hierarchical diagram, whereas some information of the latter is deleted in the former. The EF diagram is based on Petri net. That is, we need to introduce a Petri net modeling that reflects the hierarchical diagram. In this case, richer the embedded information in the hierarchical diagram become, better detection accuracy the tools have.

Further, we need to consider the simulation patterns such that the model bugs are revealed

6 Conclusion

This paper focused on MBD with multiple CAE, proposed a new unit/integration test method based on the law of energy conservation for MBD. The key idea is that the law of the energy conservation will hold for no error MBD models. Also, the paper developed the two tools supporting the energy balanced based verification. The proposed method consists of two steps. The first is a hierarchical model representation considering the type of energy. The second is an energy flow diagram based on the hierarchized model. In the numerical experiments, we used an MHEV model that is developed via multiple CAE: MATLAB/Simlink®, MapleSim®, and IPG-CarMaker®. Through the numerical experiments, we showed that the proposed method and verification tools have the possibility of judging whether the model is normal or not. Further, we discussed the detectability of model bug using the developed tools and open problems of the energy balance based verification.

References

Blochwitz, Torsten, et al. "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models." Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany. No. 076. Linköping University Electronic Press, 2012.

Shokry, Hesham, and Mike Hinchey. "Model-based verification of embedded software." (2009).

Rana, Rakesh, et al. "Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model based Development." ICSOFT. 2013.

Miyamoto, Kentaro, Kenji Sawada, and Seiichi Shin. "Energy balance based verification for hybrid vehicle models." Automatic Control Conference (CACS), 2014 CACS International. IEEE, 2014.

Tadao Murata, Petri nets: Properties, analysis and applications, Proceedings of the IEEE, Vol.77, No.4, 1989, pp. 541-580.

J. L. Peterson, Petri Net Theory and the Modeling of Systems, Englewood Cliffs, N. J.: Prentice Hall, 1981.

Parameterization Of A Simplified Physical Battery Model

Grimm Alexander[1] Prof. Haumer Anton[1]

[1]Ostbayerische Technische Hochschule Regensburg, Germany alexander.grimm@st.oth-regensburg.de, anton.haumer@oth-regensburg.de

Abstract

The importance of batteries is increasing, especially in the field of the high power requirement systems like electric driven vehicles. Mobile energy storage makes it possible to accelerate with incredible torque, without any accruing air pollution. Due to the high costs of real components, it is of great use to simulate battery driven systems before building them. Transient processes within a cell are highly dependent on the operating point of the complete system, which makes it difficult to create equations and model parameterizations. This paper shows which data is important for cell modeling and how to parameterize simplified physical cell models.

Keywords: simplified battery model, battery parameterization, physical battery model

1 Introduction

The main target of the master student's project was to get a method to generate a simplified model of a lithium ion cell. In general the physical models of batteries are generated with electronic elements like resistors, capacitors, or inductances.

The complexity of battery models can go up to infinity, so it is necessary to simplify the structure as much as possible. The simpler models contain a resistor, which represents the inner resistance of the cell and a resistor connected in parallel with a capacitor, which represents the capacitive behavior. Because of that, this model can only recreate the real battery behavior in a few situations and is also the least accurate one. To generate a more precise simulation, it is necessary to use more RC-elements (resistance connected in parallel with a capacitance). With more of these elements, it is possible to fit the Nyquist plot of batteries much more accurately.

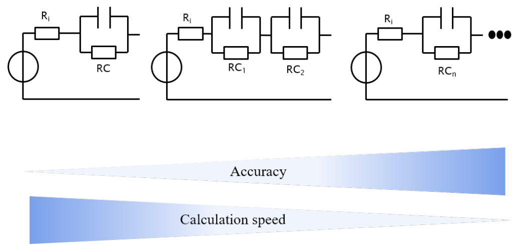


Figure 1. Rising calculation speed versus higher complexity

For high frequencies the impedance goes even inductive, so therefore a inductance should also be considered.

The disadvantage of more elements in the equivalent circuit is that more information about the cell is needed for parameterization and the CPU time for calculations is increased.

The most accurate models are completely numeric models in which all Elements are parametrized with interpolated tables, which are depending on temperature, SOC (state of charge) and flowing current. These models need much more measurement data than the ones with simple concentrated elements. All the parameters have to be recalculated in every state of the battery, which causes an extreme rise of the calculation time for the simulation. The measurements are very time consuming and expensive, so not many institutions have access to the necessary information and data for this kind of simulation models. That is the reason why in this paper no such model is discussed and presented.

2 Characteristic curves

2.1 Open circuit voltage

The open circuit voltage (short OCV) describes the cell voltage without any load, depending on the state of charge. With lower SOC the voltage of the cell is lower and vice versa. Mostly the OCV gets measured between SOC<95% and SOC>5% , so the cell does not get overloaded, or completely discharged, because both scenarios would harm the cell irreversibly. To get the information for the behavior higher, or lower than the measured values, the graph can be extrapolated.

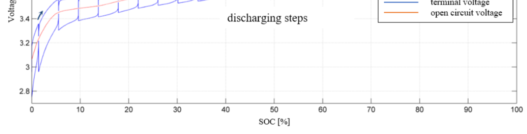


Figure 2. OCV of a battery with dis/charging steps (Peter Keil, p.3)

The OCV calculation is based on a charging and discharging process within the cell, where it gets charged/discharged with constant current impulses. The mean value out of both curves is the OCV.

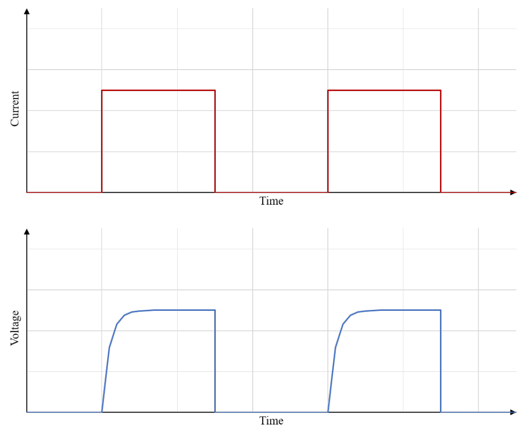


Figure 3. Schematic representation of a battery impulse response

2.2 Impulse response

The Impulse response shows the dynamic behaviour of the cell. To measure this specific curve, the battery gets electrically powered with predefined current impulses. It is possible to read a general time constant out of the graph for parametrization (Figure 3).

2.3 Electrochemical impedance spectroscopy

The electrochemical impedance spectroscopy (short EIS) shows the cell impedance depending on the current frequency and temperature. It is currently the most accurate method to describe the battery impedance behavior, but also the most elaborate and expensive with regards to the costs for measurements. Every single graph shows the behavior of the impedance for a fixed temperature and SOC. That means, that the EIS graph is different for every SOC and temperature.

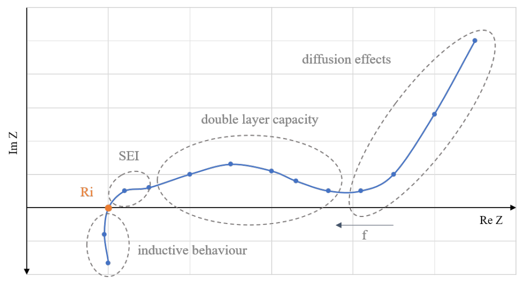


Figure 4. Schematic representation of an EIS

Figure 4 shows the graph of an EIS, which begins on the right hand side with $f \approx 0Hz$ and continue to the left with rising frequency. It can be divided into four sections, which show an individual characteristic of the cell. By low frequencies, the effect of the diffusion of the chemical ingredients influence the behavior the most. When the graph rises again, the effect of the double layer capacity, which is caused by the structure of a lithium ion cell, has the most impact. The section between the crossing point with the real part axis and the section for the double layer capacity shows the influence of the solid electrolyte interface on the impedance. The crossing point represents the inner resistance of the battery. When the frequency rise above this point, the battery begins to show inductive values, which can be simulated by inductances. It is important to know in which operating point the simulation will run mostly, so different equivalent circuits can be chosen for better results.

3 Parametrisation

There are many different models with different possibilities to parameterize them, but all of them have in common that they need the open circuit voltage for the basic voltage supply behavior. The data of the OCV can be interpolated and used for a voltage supply. In Modelica, there can be used a signal voltage and a CombiTable2D element and both are given in the Modelica Standard Library. The CombiTable2D can read the table of the OCV and provides the corresponding signal to the signal voltage source. If more OCV measurements within different temperatures are given, it is also possible to interpolate between the tables and provide a continuous behavior of the OCV within the temperature range. If the temperature can be seen as constant and it is precise enough for the individual application even data with different temperature, is not necessary.

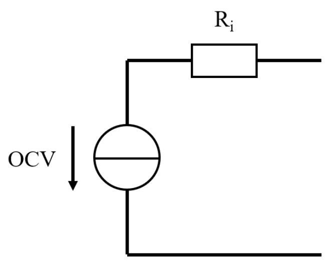


Figure 5. Equivalent circuit with inner resistance

The inner resistance can be calculated based on the voltage drop, which can be seen between the OCV and the voltage while charging/discharging the cell (figure 2).

$$R_i = \frac{\Delta U}{I_{step}} \tag{1}$$

3.1 Battery model with one RC-Element

There are two options to parameterize this model. It is possible to use the parameterization with the pulse response or with the electrochemical impedance spectroscopy. To do so with the pulse response is the much easier one and it does not need very expensive measurement systems like the other method. But like all other following models, it needs the OCV like the first and simplest model "Battery model with R_i".
The impulse response of a battery cell shows the inner resistance with the current and voltage in the first moments of the impulse. So the value of R_i (inner resistance)

can be calculated just by dividing voltage through the current. With the stationary current, that occurs after a short time ($\Delta I \approx 0$) the resistance of the RC-element can be calculated by:

$$R_{RC} = \frac{U}{I_{stat}} - R_i \tag{2}$$

With the information about R_{RC} and the impulse response, the capacitance can be calculated with the time constant τ of the rising current. τ needs to be read out of the impulse response.

$$C = \frac{\tau}{R_{RC}} \tag{3}$$

The inner Resistance can be again calculated with the OCV:

$$R_i = \frac{\Delta U}{I_{step}} \tag{4}$$

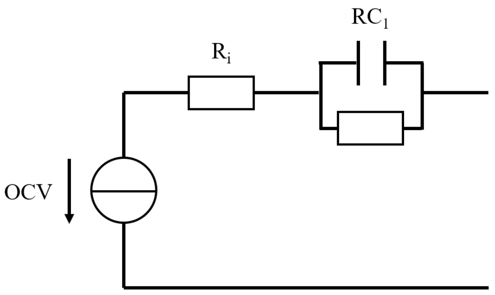


Figure 6. Equivalent circuit with one RC-Element

3.2 Parametrisation of any Model with the EIS

For this, the electrochemical impedance spectroscopy (short EIS = Nyquist plot of a battery) of lithium cells has to be analysed and evaluated. To use this method for parametrization correctly, it is important to know in which frequency band the simulation will run. As said in 2.3 the EIS itself shows the impedance depending on the frequency of the battery and it can be divided into four sections, which show the individual characteristic of the elements of the equivalent circuit.That means, that the individual segments of the curve can be used to identify the parameters for the significant elements, which should represent this characteristic in the equivalent circuit. Every individual section of the EIS can be fitted with an algorithm and the formula for the impedance for each element. Therefore optimization tools like the curve fitting toolbox, optimization toolbox for Matlab and the Solver for Microsoft Excel can be used. The following example shows how it is done.
For this example the equivalent circuit with inner resistance and two RC-Elements is chosen. This structure is a very common model for batteries and it should provide very high accuracy for low level frequencies (frequencies in which the battery behavior is not inductive). For the battery model application in cars, it can be said, that speed requirement (pedal position) does not change with high frequencies $f \approx kHz$ and so doesn't the current for the motor with additional loads.

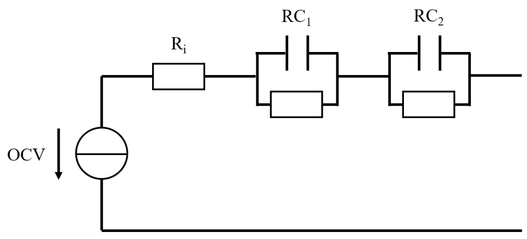


Figure 7. Equivalent circuit with two RC-Elements

To get the parameters for the other elements, it is necessary to know how the impedance for the structure is calculated. With the knowledge about the formula for the impedance it is possible to use a fitting algorithm to calculate the capacitances for the capacitors and the resistances for the resistors.

$$Z_G = R_i + Z_{RC1} + Z_{RC2} \tag{5}$$

$$Z_{RC1} = \frac{R_1}{(\omega R_1 C_1)^2 + 1} - \frac{j\omega R_1^2 C_1}{(\omega R_1 C_1)^2 + 1} \tag{6}$$

$$Z_{RC2} = \frac{R_2}{(\omega R_2 C_2)^2 + 1} - \frac{j\omega R_2^2 C_2}{(\omega R_2 C_2)^2 + 1} \tag{7}$$

This leads to the separated real and imaginary part of Z_G:

$$R_G = R_i + \frac{R_1}{(\omega R_1 C_1)^2 + 1} + \frac{R_2}{(\omega R_2 C_2)^2 + 1} \tag{8}$$

$$X_G = -\frac{j\omega R_1^2 C_1}{(\omega R_1 C_1)^2 + 1} - \frac{j\omega R_2^2 C_2}{(\omega R_2 C_2)^2 + 1} \tag{9}$$

The next part of the parameterization of a battery model with the EIS is to look up the measurement table of the EIS and choose the suitable data for the recreation of the graph. If the model has no inductance included, it is not possible to simulate inductive behavior. So all data with positive imaginary part is not suitable for the fitting and can be ignored.
The next step is to use the formula for the resistance (eq.:8) and the reactance (eq.:9) and minimize the deviation of the error squares:

$$(R_{measured} - R_{calculated})^2 \overset{!}{=} min \tag{10}$$

$$(X_{measured} - X_{calculated})^2 \overset{!}{=} min \tag{11}$$

Now it is time to use the fitting tool to calculate the four remaining parameters for the RC-Elements. The tool should find the minimum of the error squares by changing the parameters for the RC-Elements. Every solving algorithm needs well chosen start values for the calculation. Therefore the resistances should not be higher than realistic values. The resistances are commonly not higher than one Ohm for li-ion battery cells. The capacitances are more difficult to select. Every cell chemistry is different, even if it also is among li-ion batteries and therefore the capacitances vary massively. If there aren't many informations about the cells and the start value available, zero Farad can be a good value for the beginning of the fitting process. The solving algorithm tries to find the minimum for the error squares by changing the values of the RC-Elements and the formulas for the impedance. All of those algorithms are iterative solving methods which search for local minimums of the given functions and parameters. With the wrong start values the results will be false and need to be evaluated. This needs to be kept in mind.

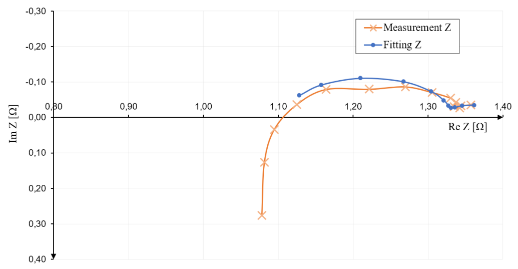


Figure 8. Fitting result with example measurement data.

Figure 8 shows an example with the resulting impedances for the given frequencies. The fitting can approximate the measured graph very closely, which means, that the behavior of the real cell can be reproduced very well.

4 Battery Model for long term energy investigations of electric vehicles

This project "finding a suitable battery model and the possibility to parametrize it", is based on the previous project to create a Modelica based simulation library for modeling and simulation of complete electrically driven vehicles. To keep it simple and reasonable for a electical physical model, the following considerations were made.

4.1 Chosen model

After intense research in papers and books the decision was to take the cell model with inner resistance and two RC-Elements to get high accuracy and precision for the simulation. The structure of the RC-Elements can be seen as the two transition layers of the electrodes and the inner resistance like shown in figure 10. The two RC-Elements have to be parametrized differently, because of the different chemical structure at each represented transition layer. The Resistors R_{i1} and R_{i2} represent the electrical resistance of the electrodes themselves. R_{el} reproduces the resistance of the electrolyte for the lithium-ions to go through.

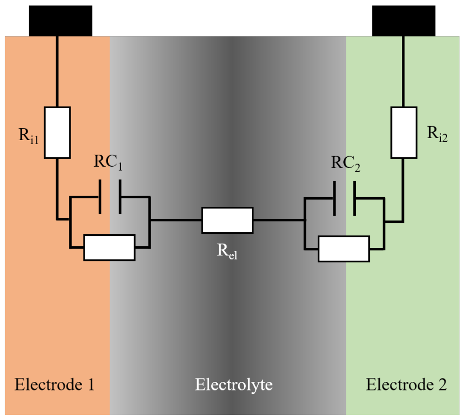


Figure 9. Consideration of the equivalent circuit.

The resistances R_{i1}, R_{i2}, R_{el} in sum result into R_i.

$$Z_G = R_{i1} + R_{i2} + R_{el} + Z_{RC1} + Z_{RC2} \quad (12)$$
$$= R_i + Z_{RC1} + Z_{RC2} \quad (13)$$

The model in the library needs additional elements for handling all the data and inclusion of the option to use all resistances for thermal management. An integrator converts the measured current into charge for calculation of the SOC. The SOC itself needs to be denormalized for the interpolation table of the OCV and all relevant signals are combined in the battery bus for a better overview. For a more realistic behaviour, the self discharge resistance R_{sd} is also considered and set by default to $1M\Omega$.

4.2 Simulation and modelling

Until now, this research project has no real EIS-measurements of cells from high voltage battery packs available. So it would make no sense to parameterize an model example of a car and try to simulate it, because it cannot be verified with any real measurements. The given graph in figure 8 at p.4 shows a value for the inner resistance that is way too high. With that data it is not possible to reproduce driving cycles with high acceleration and power requirements above $\approx 150kW$. Nevertheless the EMOTH battery model was tested with fictive values and was completely operational. It has to be kept in mind that the simulation will abort with a singularity error if the resistance is to high for the power requirement and the data for the OCV can only interpolate between the given data and not be extrapolated. If the temperature is rising above the maximum given value or falling below the minim given value, it will abort the simulation.

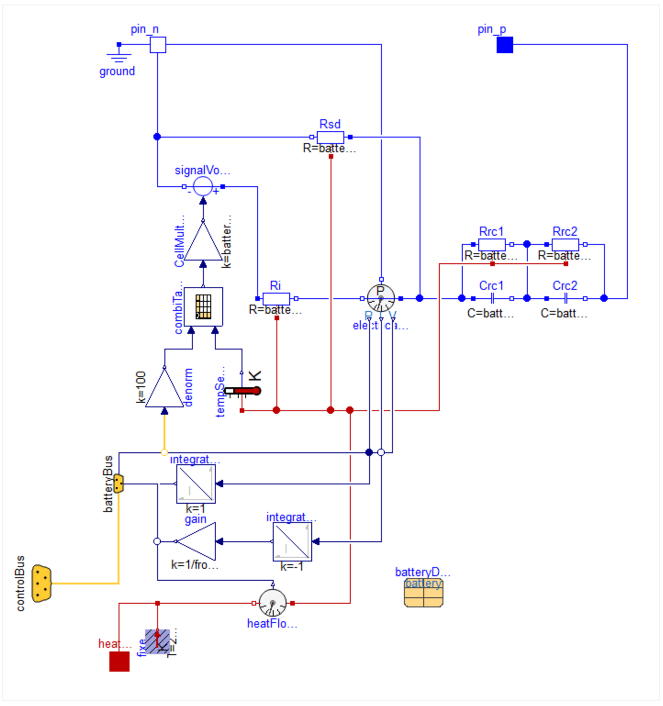


Figure 10. Equivalent circuit of the battery in Dymola.

For testing the functionality and the containing Modelica-elements, a simpler model was built and tested with different temperatures. The OCV was analysed and the results are shown in figure 11. The table2D interpolation was set to interpolate with continuous derivatives to smooth the graph. The parameters for the elements were set to the values obtained out of the measured EIS. Figure 11

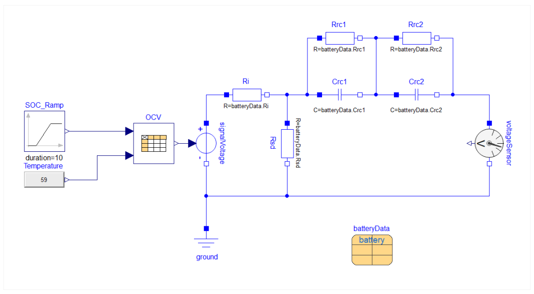


Figure 11. Battery model test for OCV behaviour

shows the structure of the test model. The SOC_ Ramp changes the SOC from 100% to 10% in 10 seconds with the fixed and predefined temperature, which is defined in the temperature block. All equivalent circuit parameters are summed up in the data record "batteryData". The voltage sensor measures the open circuit voltage depending on the changing SOC.

The graph in figure 12 shows the resulting interpolation from the example data from figure 8 p.4 for two different temperatures ($T_1 = -20°\mathrm{C}$ and $T_2 = 59°\mathrm{C}$). If there are real and useful electrochemical impedance spectroscopies available with different temperatures and state of charges, the previous discussed models and parametrization methods should provide very good simulation structures for accurate battery simulations.

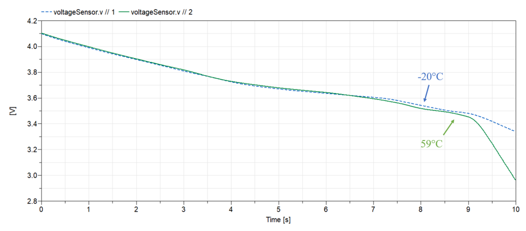


Figure 12. Recreated OCV in Dymola with different temperatures

5 Conclusion and outlook

There are many different ways to create a battery model, but the higher the level of complexity, the more calculation time and measurements are needed to be performed as basis for parametrization. Since this is very expensive and time consuming, the requirement of modeling is to keep the simulation structures as simple as possible.

In (Peter Keil) it is proven that even the equivalent circuit with only one RC-element can provide very accurate results. They used a model build up of one RC-Element and a predefined load profile for discharging the battery. Between the simulation and real measurements the difference was less than two percent, except for the operating points with a SOC lower than five percent, where the difference came up to 10%. The choice for the battery model depends on the type of measurements that can be proceeded and the available calculation power.

The next step is to use the obtained results and combine them with additional data to create and parametrize a complete model of the electrically driven bus of Regensburg.

References

Peter Fritzson. *Principles of object oriented modeling and simulation with Modelica 3.3*. John Wiley and Sons Inc., 2015. ISBN 978-1-1188-5897-4.

Rodrigo Garcia-ValleJoao A. Pecas Lopes. *Electric Vehicle Integration into Modern Power Networks*. Springer, New York, NY, 2013. ISBN 978-1-4614-0134-6.

Wolfgang Mielke. *Modellierung von Kennlinien, Impedanzspektren und thermischem Verhalten einer Lithium-Eisenphosphat-Batterie*. PhD thesis, 2011.

Andreas Jossen Peter Keil. *Aufbau und Parametrierung von Batteriemodellen*. PhD thesis.

Otto K. Dietleier Peter Kurzweil. *Elektrochemische Speicher*. Springer Fachmedien Wiesbaden GmbH, 2015. ISBN 978-3-658-10899-1.

D. V. Ragone. Review of battery systems for electrically powered vehicles. 1968.

Jan Philipp Schmidt. *Verfahren zur Charakterisierung und Modellierung von Lithium-Ionen Zellen*. PhD thesis, 2013.

Modeling of transformer-rectifier sets for the energization of electrostatic precipitators using Modelica

Mads Nannestad[1,2] Benoit Bidoggia[1] Zhe Zhang[2] Tiberiu-Gabriel Zsurzsan[2] Kasper Skriver[1]

[1]FLSmidth A/S, Denmark, bebi@flsmidth.dk
[2]Department of Electrical Engineering, Technical University of Denmark, Denmark

Abstract

Electrostatic precipitators (ESPs) are important parts of many industrial plants. They require high-voltage power supplies. In this paper, transformer/rectifier sets, a particular type of power supplies for the energization of ESPs, are presented and modeled using Modelica. The models have been validated with measurements from existing plants.

Keywords: Modelica, power supply, electrostatic precipitator, ESP

1 Introduction

Industrial plants—like coal-fired power plants and plants for the production of steel, pulp and paper, and cement—need to satisfy low and strict emission levels of pollutants in the emitted flue gases. One method to reduce the emission of solid pollutants suspended in gas streams is the use of electrostatic precipitators (ESPs) (Fig. 1). In ESPs, the particles of pollutants are electrically charged and pass through a strong electric field. The charged particles moving within the gas stream are therefore deflected towards collecting plates, to which they stick (Fig. 2). To create the ions that are required to charge the particles and to sustain the required strong electric field, high-voltage power supplies are required. ESPs are normally internally subdivided in electrically isolated sections, which are independently energized and controlled. Different topologies of power supplies which can be used to energize ESPs exist. One of them, also the most traditional, is called transformer/rectifier (T/R) set. T/R sets can be single phase or three phase (von Stackelberg, 2013). The aim of this paper is to model open-loop controlled, single- and three-phase T/R sets coupled to ESPs.

To understand the importance of the voltage shape and voltage level energizing ESPs, the concept of collecting efficiency η is introduced and defined as

$$\eta = \frac{\dot{m}_\mathrm{c}}{\dot{m}_\mathrm{i}} \tag{1}$$

where $\dot{m}_\mathrm{c}$ is the mass flow of collected particles and $\dot{m}_\mathrm{i}$ is the mass flow of incoming particles. The efficiency η depends on different geometrical parameters of the ESP—like the physical size and the shape of the electrodes and of the collecting plates—and of different physical parameters of the gas stream—like its density, temperature and velocity.

Figure 1. Example of electrostatic precipitator

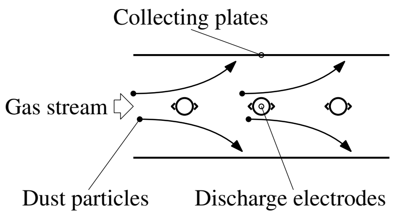


Figure 2. A schematic representation of a portion of an electrostatic precipitator

The influence of the geometrical and physical parameters can be expressed by

$$\eta = 1 - e^{-\frac{A}{Q}\omega} \tag{2}$$

where A is the total collecting area, Q is the flow rate of the gas stream and ω is called particle migration velocity. Under normal operating conditions, the particle migration velocity is proportional to the power P_c injected into the ESP and can be expressed as

$$\omega = k\frac{P_\mathrm{c}}{A} \tag{3}$$

where k is a constant that depends on the physical parameters of the particular process (Parker, 1997).

The efficiency η can also be expressed more directly in electrical terms by

$$\eta = 1 - e^{k_\eta \bar{v}\hat{v}} \tag{4}$$

where k_η is a constant (Bidoggia et al., 2018), and where $\bar{v}$ and $\hat{v}$ are respectively the mean and peak values of the ESP

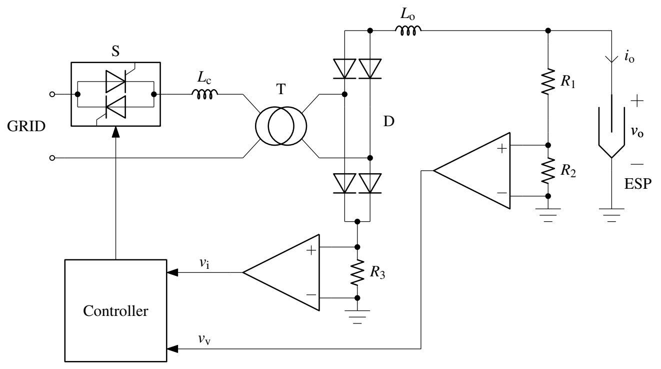


Figure 3. Schematic and block diagram of a single-phase T/R set connected to an electrostatic precipitator (ESP)

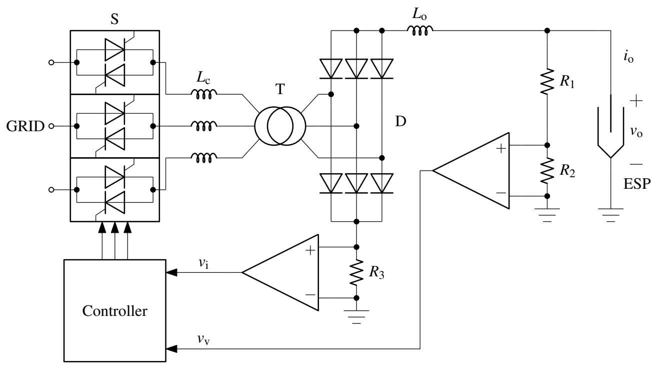


Figure 4. Schematic and block diagram of a three-phase T/R set connected to an electrostatic precipitator (ESP)

voltage. To maximize the collecting efficiency η of an ESP, the T/R-set controller acts on the firing angle of thyristors to keep the highest possible mean ($\bar{v}$) and peak ($\hat{v}$) voltage, typically at the limit of the dielectric breakdown voltage of the gas flowing through the ESP.

In this paper, after introducing the main components of single- and three-phase T/R sets (Sec. 2), the Modelica models which have been built will be described (Sec. 3). The simulation results and field measurements are presented and compared in Sec. 4. The conclusions are then presented in Sec. 5.

2 Energization of ESPs with T/R sets

In this section, the main components of single- and three-phase T/R sets are presented (Fig. 3 and 4). Because of the different output voltage they generate, their usage depends on the particular process of which the ESP is part.

All T/R sets are composed of a step-up transformer (T) and a full-wave rectifier (D). The positive connection of the rectifier is connected to ground to obtain a negative voltage across the ESP. The grid voltage is typically in the range 400 V to 690 V and the absolute value of the output voltage v_o can typically be in the range 80 kV to 150 kV. The output voltage is measured by the voltage divider made of $R_1$and R_2. The average value of the output current i_o is typically in the range 800 mA to 2000 mA and is measured by the shunt resistor R_o. Because sparks can occur inside an ESP under normal operation, an external inductor (L_c) is inserted to limit the resulting overcurrent; typical per-unit values for L_c are in the range 30 % to 40 % (Parker, 1997). The voltage and current signals, respectively v_v and v_i are measured by the controller, which regulates the firing angle of the thyristors (S) to regulate the power injected to the ESP. On the output side, the inductor L_o prevents high-frequency transients created by sparks from entering the T/R set. Because of the high voltage levels, the transformer T, the rectifier D, the resistor R_1 and the inductors L_o and L_c are immersed in an oil tank. The oil is also used for cooling purposes.

2.1 Single-phase T/R set

The output voltage waveform of single-phase T/R sets is characterized by a DC component with a superimposed relatively large ripple, with a frequency twice the grid frequency. This type of waveform is typically used in applications for which the resistivity of the particulate to filter falls in the medium-resistivity range ($1 \times 10^7\,\Omega\,\text{m}$ to $5 \times 10^9\,\Omega\,\text{m}$) (Bidoggia et al., 2018).

2.2 Three-phase T/R set

The output voltage waveform of three-phase T/R sets is characterized by a DC component with a negligible ripple, with a frequency six times the grid frequency. This type of waveform is typically used in applications for which the resistivity of the particulate to filter falls in the low-resistivity range ($< 1 \times 10^7\,\Omega\,\text{m}$) (Bidoggia et al., 2018).

The transformer is typically connected in a configuration with delta connection at the primary side and star connection at the secondary side.

3 Modeling

The Modelica Standard Library and the OpenModelica Connection Editor (OMEdit) have been used to model both single- and three phase T/R sets. The availability of interfaces between different physical domains makes Modelica ideal for modeling multiphysical systems like ESPs. For this work, the following physical domains have been used: *electrical*, for the power electronics stage; *blocks*, for the open-loop control of the power electronics stage; *thermal*, for the calculation of the losses.

In this paper, the model of an ESP has been simplified and represented by an R-C circuit, where the values of resistance and capacitance were based on real installations. With the aim of setting the base for a model as representative as possible of real systems, the voltage divider and the current sensor have also been modeled. The voltage divider is designed to provide a voltage ratio $V_n/10\,\text{V}$, where V_n is the nominal output voltage of the T/R set. The current shunt is designed to provide a current/voltage ratio $I_n/1\,\text{V}$, where I_n is the nominal output current of the T/R set.

The heating ports of the components from the elec-

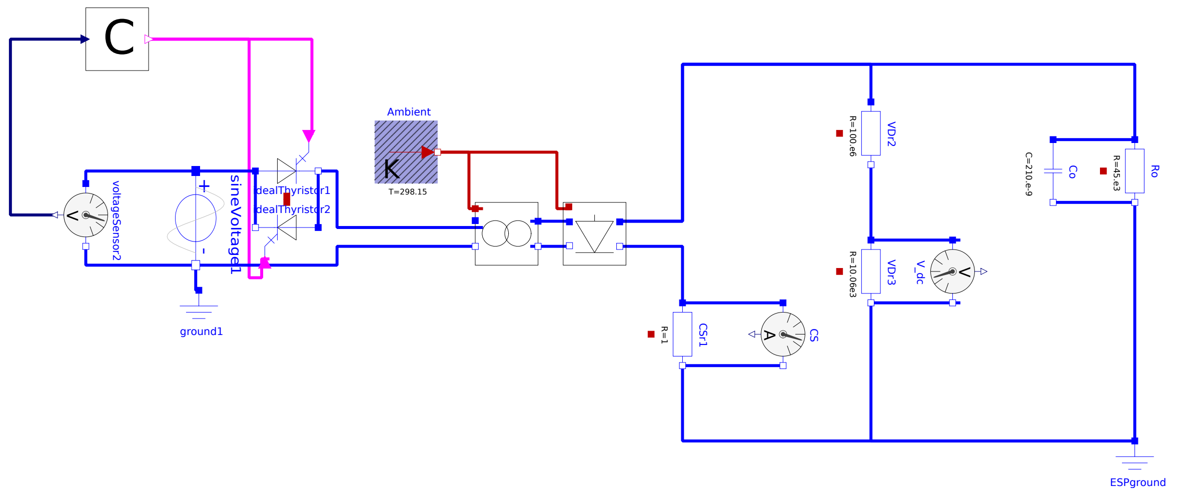


Figure 5. Modelica model of a single-phase T/R set connected to an electrostatic precipitator (ESP)

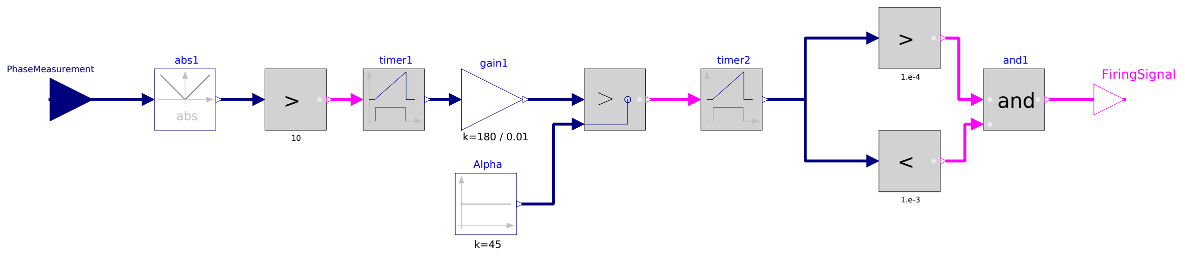


Figure 6. Submodel of a simple open-loop controller for T/R sets

trical domain with losses—such as diodes, resistors and thyristors—have been connected to a fixed ambient temperature of 298.15 K ≈ 25 °C.

The values of the parameters used in the models are provided by FLSmidth and based on real installations; they are reported in Table 1.

3.1 Single-phase T/R set

The derived model of a single-phase T/R set is shown in Fig. 5 with its submodels for the controller, the transformer and the rectifier (Tiller, 2017).

The controller is connected to two ideal thyristors which are fired by boolean signals (Otter et al., 1999). The submodel of the controller is shown in Fig. 6. The controller detects the zero crossing of the grid voltage, represented by an ideal voltage source, and then delays the firing of the thyristors of the diode. The firing delay is given as an angle (α) in the range 0° to 180°. The firing angle is compared to the output of the chain of the blocks timer1 and gain1, which represents the delay from the zero crossing of the voltage, expressed in degrees. Higher values of α correspond to lower output power. The block timer2 defines the duration of the firing pulse. Because two anti-parallel thyristors are present, the firing signal is given every half period.

The submodel of the full-wave rectifier is shown in Fig. 7. Because of the high-voltage levels applied to the rectifier, each diode represents a string of diodes connected in series. The three different models of diodes available at the Modelica Standard Library have been investigated (Clauss et al., 2000):

- HeatingDiode: while testing it with a three-phase rectifier and with the heating ports enabled, it seemed to lead to numerical problems;
- Diode2: because the relationship between voltage and current contains exponential expressions, it is not possible to extend the parameters of a diode to a string of diodes by simply manipulating the parameters themselves;
- IdealDiode: the model being linear, the parameters of a string of diodes can be simply derived from the parameters of a diode and the number of diodes in the string.

These models do not include the effect of the reverse recovery (Denz et al., 2014), which is not a concern for this application because of the low switching frequency (50 Hz to 180 Hz).

The submodel of the single-phase transformer is shown in Fig. 8 and it is made of the two inductances L_1 and L_m,

Table 1. Parameters for the models of the single- and three-phase T/R sets

	$R_{\{1\|2\|3\}}$ [mΩ]	$L_{\{1\|2\|3\}}$ [mH]	L_m [mH]	R_e [Ω]	n [−]	R_o [MΩ]	C_o [nF]
Single-phase	25	1.8	94	200	1/160	45	210
Three-phase	25	1.8	94	94	1/94	100	100

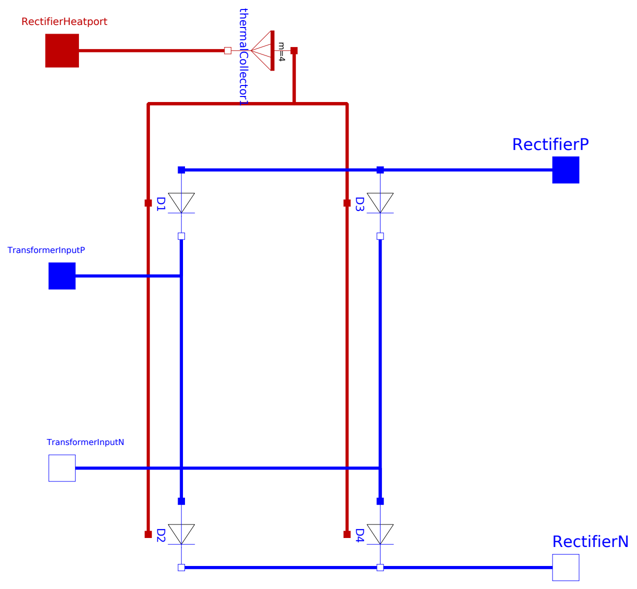


Figure 7. Submodel of the single-phase full-wave rectifier of a T/R set

the two resistances R_1 and R_e, and an ideal transformer with voltage ratio $n = n_1/n_2$.

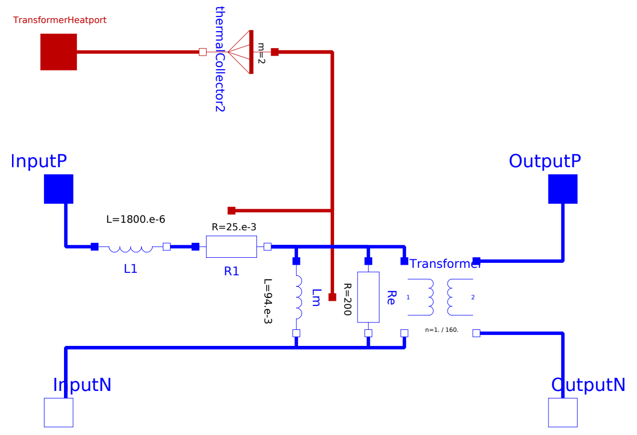


Figure 8. Submodel of the single-phase transformer of a T/R set

The inductance L_m represents the magnetizing inductance of the core of the transformer, while the inductance L_1 is defined as

$$L_1 = L_{1\sigma} + n^2 L_{2\sigma} + L_\mathrm{c} + n^2 L_\mathrm{o} \qquad (5)$$

where $L_{1\sigma}$ is the leakage inductances of the primary side, $L_{2\sigma}$ is the leakage inductance of the secondary side, L_c is the inductance of the external short-circuit inductor and L_o is the inductance of the output inductor.

The resistance R_e represents the losses in the core of the transformer, while the resistance R_1 is defined as

$$R_1 = R_{1\mathrm{w}} + n^2 R_{2\mathrm{w}} + R_\mathrm{c} \qquad (6)$$

where $R_{1\mathrm{w}}$ is the resistance of the winding of the primary side, $R_{2\mathrm{w}}$ is the resistance of the winding of the secondary side, and R_c is the resistance of the external short-circuit inductor.

3.2 Three-phase T/R set

The derived model of a three-phase T/R set is shown in Fig. 9 with its submodels for the controller, the transformer and the rectifier (Tiller, 2017).

The controller model is the same as for a single-phase T/R set but replicated three times.

The choice of the model for the diodes is the same as for the single-phase T/R set. The submodel of the rectifier is shown in Fig. 10.

The submodel of the high voltage transformer is shown in Fig. 11. The inductances and resistances appearing in the model are defined similarly to the homologous ones in the model for the single-phase transformer.

4 Results

In this section, the results of the simulation of the models described in Sec. 3 and the results measured on existing plants are presented. Because of the different parameters and the different nominal values of the simulated and real systems, the results have been normalized so that the waveforms of different systems can be compared.

Fig. 12a presents the primary current for both a single- and a three-phase T/R sets connected to an ESP and measured on existing plants. Fig. 12b presents the simulated primary current for both a single- and three-phase T/R set connected to an ESP.

Fig. 12c presents the secondary current and secondary voltage for a single-phase T/R set connected to an ESP and measured on a real plant. Fig. 12d presents the simulated secondary current and secondary voltage for a single-phase T/R set connected to an ESP.

Fig. 12e presents the secondary current and secondary voltage for a three-phase T/R set connected to an ESP and

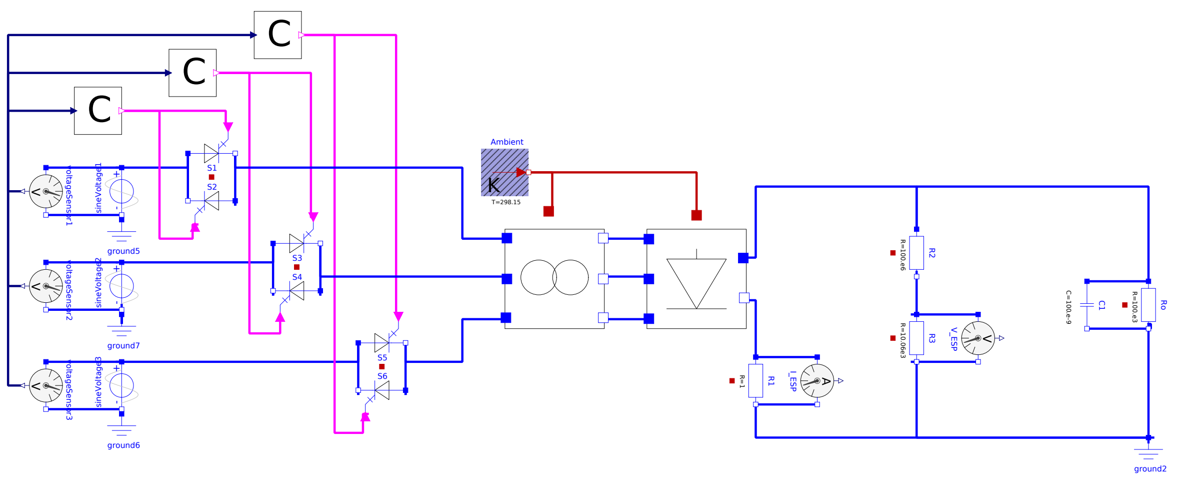


Figure 9. Modelica model of a three-phase T/R set connected to an electrostatic precipitator (ESP)

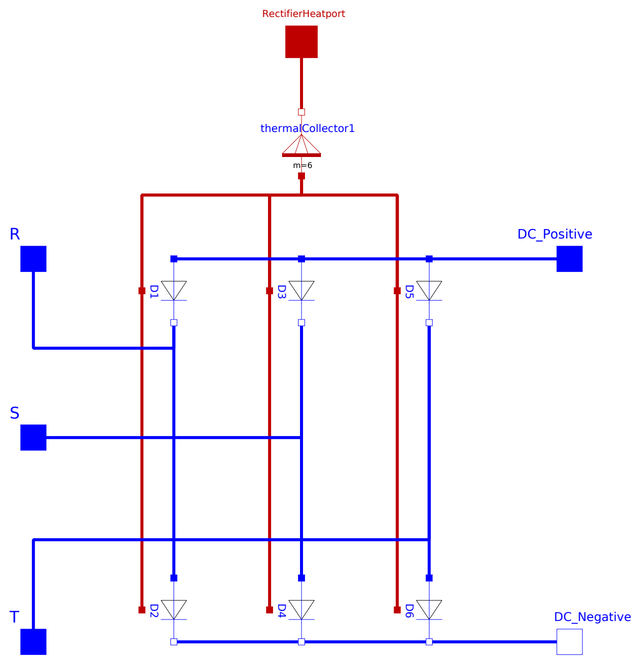


Figure 10. Submodel of the three-phase full-wave rectifier of a T/R set

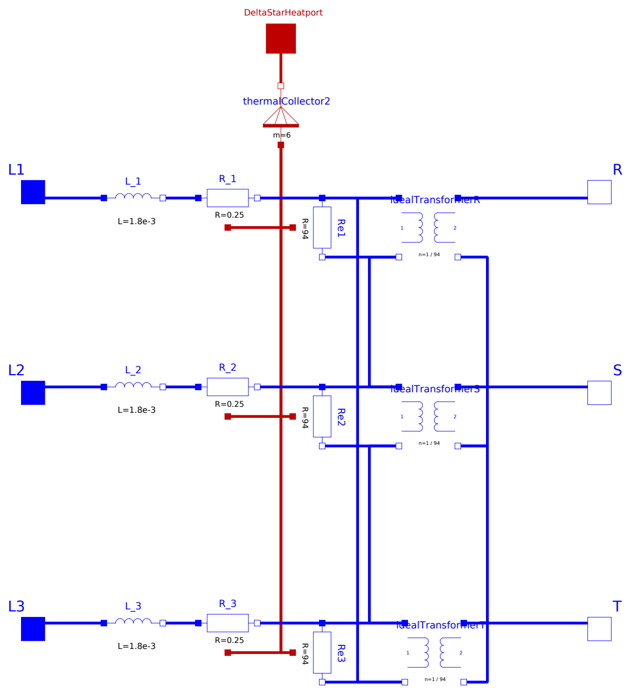


Figure 11. Submodel of the three-phase transformer of a T/R set

measured on a real plant. Fig. 12f presents the simulated secondary current and secondary voltage for a three-phase T/R set connected to an ESP. The amplitude of the voltage ripple is significantly smaller than in the case of the single-phase T/R set.

For all waveforms, the shape of the simulated waveforms is in agreement with the shape of the measured waveforms. The minor differences—in the primary current and in the secondary voltage and current—between the simulated and measured results are mostly related to the nature itself of ESPs. These are loads whose parameters are constantly changing as a function of the physical parameters of the gas streams that they are filtering. Thus, also modeling the ESP itself will be of paramount importance, especially in relationship with a closed-loop control system. Other minor differences—like for example in the primary current—are related to other electrical phenomena, which have not been taken into consideration yet, like the switching transients of the thyristors.

5 Conclusions

Electrostatic precipitators (ESPs) are important parts of many industrial plants and they require high-voltage power supplies. The goal of the project to which this work belongs is to model a complete ESP system. In this paper, a particular type of power supplies for the energization of ESPs—transformer/rectifier sets—has been presented and simple models have been derived using Modelica. The models have been validated with measurements from existing plants. However, further work will be required, like the derivation of the models of the ESP, of closed-loop controllers and of other topologies of power supplies. The derivation of a complete thermal model for these power supplies is planned.

6 Acknowledgment

We thank our colleagues from FLSmidth A/S and DTU, who provided insight and expertise that greatly assisted the research.

References

Benoit Bidoggia, Mads Kirk Larsen, Karsten Poulsen, and Kasper Skriver. Coromax micro-pulse power supplies (mppss) replace switch-mode power supplies (smpss) at an estonian power plant. In *Proceedings 15th International Conference on ESP (ICESP), Charlotte, NC, USA, 9–11 October (2018)*, 2018.

C. Clauss, A. Schneider, T. Leitner, and P. Schwarz. Modelling of electrical circuits with modelica. In *Workshop Modelica*, 2000.

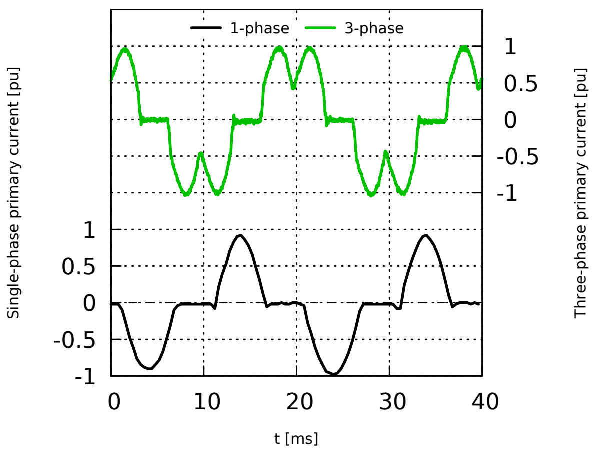
Patrick Denz, Thomas Schmitt, and Markus Andres. Behavioral modeling of power semiconductors in modelica. In *10th International Modelica Conference*, 2014.

Martin Otter, Hilding Elmqvist, and Sven Erik Mattsson. Modeling of mixed continuous/discrete systems in modelica. Technical report, 1999.

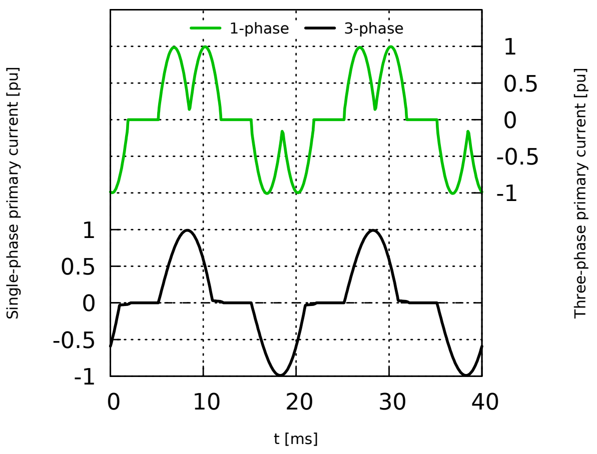
K.R. Parker, editor. *Applied electrostatic precipitation*. Chapman and Hall, 1997.

Michael M. Tiller. *Modelica by Example — Release v0.5.3-0-g4c1367c*. 2017.

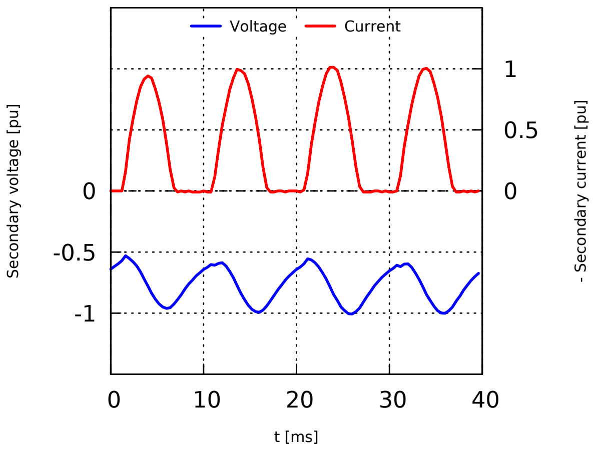
Josef von Stackelberg. The three-phase power supply for low ripple high voltage in conventional technology. In *ICESP XIII, September 2013, Bangalore, India*. Rico-Werk, Toenisvorst, Germany, 2013.



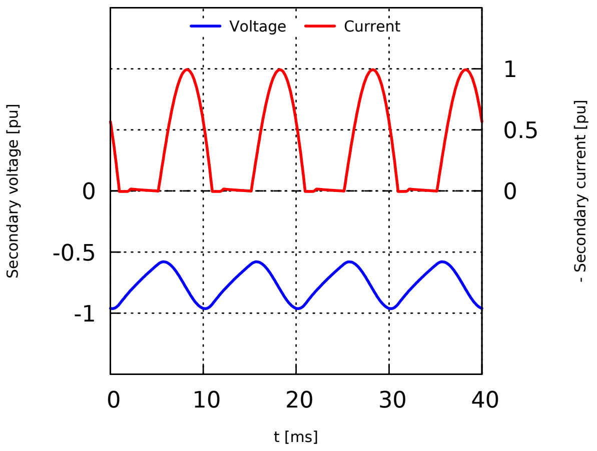
(a) Measured primary current for a single-phase (base current $I_b = 170\,\mathrm{A}$) and a three-phase (base current $I_b = 160\,\mathrm{A}$) T/R set



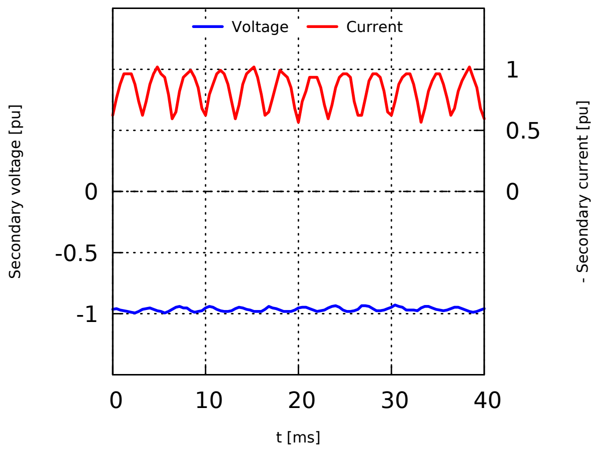
(b) Simulated primary current for a single-phase (base current $I_b = 150\,\mathrm{A}$) and a three-phase (base current $I_b = 85\,\mathrm{A}$) T/R set



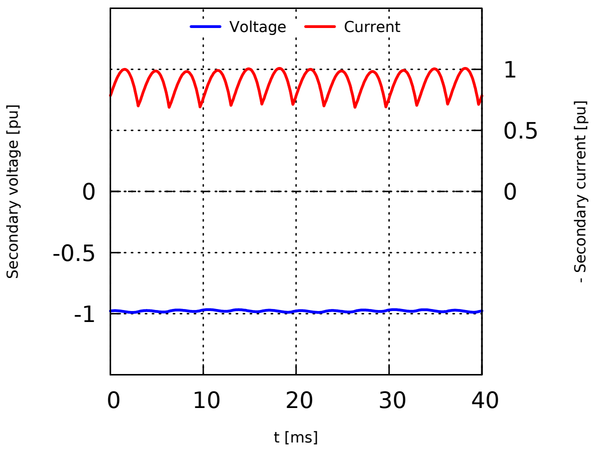
(c) Measured secondary voltage (base voltage $U_b = 76\,\mathrm{kV}$) and current (base current $I_b = 1800\,\mathrm{mA}$) for a single-phase T/R set



(d) Simulated secondary voltage (base voltage $U_b = 51\,\mathrm{kV}$) and current (base current $I_b = 2400\,\mathrm{mA}$) for a single-phase T/R set



(e) Measured secondary voltage (base voltage $U_b = 76\,\mathrm{kV}$) and current (base current $I_b = 1200\,\mathrm{mA}$) for a three-phase T/R set



(f) Simulated secondary voltage (base voltage $U_b = 51\,\mathrm{kV}$) and current (base current $I_b = 800\,\mathrm{mA}$) for a three-phase T/R set

Figure 12. Comparison between the main waveforms of measurements and simulation results for single-phase and three-phase T/R sets

A Model Predictive Control Application for a Constrained Fast Charge of Lithium-ion Batteries

Alberto Romero* Alejandro Goldar Emanuele Garone

Service d'Automatique et d'Analyse des Systèmes, Université libre de Bruxelles, Belgium,
{aromerof,agoldard,egarone}@ulb.ac.be

Abstract

The spread of electrical storage devices continues to be underpinned by the limited charging currents that can be applied. The limitation arises from the lack of sufficient high power charging stations, either at home or along roads and highways, and from the maximum admissible current that can be applied to the battery before undesirable degradation mechanisms are triggered. Accordingly, most traditional charging protocols limit the charging current as a function of the standing state of charge of the battery. These protocols are designed empirically and restrict the potential benefit of more flexible charging options. However, the alternative to traditional protocols must rely on a more precise knowledge of the operating constraints and on advanced control techniques to compute online the best operating plan. This work presents a model predictive control (MPC) application to minimize the charging time of a lithium-ion battery subject to electrochemical and thermal constraints. The satisfaction of these constraints ensures that the battery degradation is minimized, or at least mitigated. The programming language Modelica and the optimization and simulation framework JModelica.org is used in combination with Python language to assess the computing time and potential use of MPC and the developed cell models in commercial batteries.

Keywords: Fast-charge, nonlinear MPC, optimization, battery aging

1 Introduction

Lithium-based battery cells dominate the spectrum of electrical storage when it comes to portable electronic/-electric devices. These cells can withstand thousands of charge-discharge cycles before degradation makes them unusable. However, when the batteries are charged at high rates, the expected life is reduced. In general, the degradation rate depends on the charging current and the cell temperature. Empirical results show that operation above 40 °C can dramatically reduce the life of the cell (Wang et al., 2011; Ecker et al., 2012). The same results from operating at low temperatures (below 10°C).

High charging current increases the voltage of the battery due to the effect on the cell of the electrochemically induced overpotentials. This observable effect is a consequence of the ohmic, charge transfer, and diffusion resistances (Gerl et al., 2014). When these exceed certain thresholds then the side-reactions that cause cell degradation are triggered.

The overpotentials depend on the current applied I, the temperature T, and the state of charge (SOC). It is expected, therefore, that if certain operating constraints defined as a function of these variables are not violated, then the degradation mechanisms will be ceased or at least slowed down. How these constraints are identified and modeled has been covered recently by different authors (e.g. (Moura et al., 2013; Romagnoli et al., 2017)). It has been more common, however, to use empirical results of capacity and power fade (degradation) as a function of SOC, I and T, to subsequently obtain empirical degradation expressions by some fitting procedure. The resulting expressions can be used in combination with control schemes to maximize the benefit of utilizing the cells in the best possible way, often optimal under certain criterion.

In the recent years, Modelica has been chosen by many research institutions and companies to study the interaction of batteries with other systems such as power-trains, cooling devices, or power electronics. The component-oriented programming facilitated by Modelica can be used to compare multiple configurations in the design stage, as well as to optimize the size and operation of battery systems. Several libraries have been developed in the last years ((Dao and Schmitke, 2015; Uddin and Picarelli, 2014; Gerl et al., 2014; Bouvy et al., 2012; Brembeck and Wielgos, 2011; Einhorn et al., 2011; Janczyk et al., 2016)), which have been validated with experimental data. Applications have focused mostly on hybrids, plug-in hybrids, and full electric vehicles, with emphasis on fuel economy (Batteh and Tiller, 2009; Spike et al., 2015), thermal management (Bouvy et al., 2012), and battery aging (Gerl et al., 2014). Despite the significant effort, there is still some room for improvement in the area of constrained control techniques using Modelica to explicitly account for electrochemical and thermal operating boundaries.

This paper presents an optimal control strategy to charge a lithium-ion battery cell subject to electrochemical constraints. The model used to describe the cell dynamics and to draw the operational limits is the so-called Equivalent Hydraulic Model (EHM), which is linear on

*Corresponding author

the two electrochemical states (state of charge and critical surface concentration), and nonlinear on the output voltage. In addition to the electrochemical states, the cell temperature dynamics is also modeled and constrained. The optimal charging profile is obtained and applied over a receding horizon following a classical MPC approach, and is implemented in Python and JModelica.org. The optimal control strategy is compared with a commercial constant-current/constant-voltage (CCCV) charging protocol that is the standard in most applications, illustrating the benefits of the optimal constrained charging strategy.

The remainder of this work can be summarized as follows. Section 2 introduces the EHM model and its extension to include the temperature state. Section 3 describes the conventional and the optimal charging strategies, as well as the implementation details. Results and their discussion are covered in Section 4, followed finally by the conclusions and future work.

2 Battery modeling and control

This section presents a reduced order model of the battery cell taken from the literature, as well the description of the coupled thermal model and two control strategies that are later compared for battery charging.

2.1 Equivalent Hydraulic Model

The equivalent hydraulic model (EHM) was first proposed by (Manwell and McGowan, 1993) for lead acid batteries, but it has been recently applied to lithium-ion cells as a reduced-order electrochemical battery model. Figure 1 depicts the physical meaning of the model states and how they interact with the flow of lithium ions. The model captures the dynamics of an idealized two-layer single particle within which lithium accumulates. The dynamics can be represented with more traditional two-tank hydraulic model (**?**). In continuous-time, this model takes the form

$$\frac{d\mathrm{SOC}}{dt} = -\gamma I \tag{1}$$

$$\frac{d\mathrm{CSC}}{dt} = \frac{g(\mathrm{SOC}-\mathrm{CSC})}{\beta(1-\beta)} - \frac{\gamma}{1-\beta} I \tag{2}$$

where SOC and CSC are the state of charge and the critical surface concentration of lithium ions respectively, I is the applied current in $[\mathrm{A}\cdot\mathrm{m}^{-2}]$, and g, β and γ are constant parameters as defined in (Couto and Kinnaert, 2018). The convention of negative currents for battery charge is respected here.

The positive electrode dynamics are usually faster than the negative electrode ones, which motivates the assumption of fast dynamics for the former electrode, i.e.

$$\begin{aligned} \mathrm{CSC}^+ &= \mathrm{SOC}^+ \\ \mathrm{SOC}^+ &= \rho\mathrm{SOC} + \sigma \end{aligned} \tag{3}$$

where ρ and σ are also constant parameters. The output voltage of the battery is a nonlinear function consisting of the open-circuit voltage ΔU, the surface overpotential $\eta_s^\pm$ and the film resistance R_f given by

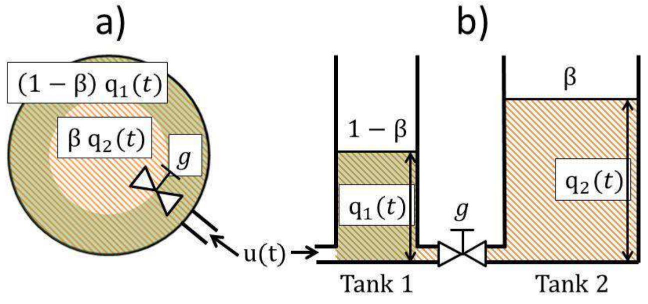


Figure 1. Equivalent hydraulic model and the spherical-solid particle representation (Couto et al., 2016). u(k), q_1 and q_2 are, respectively, the lithium flow, and the bulk and surface lithium concentration

$$V = \Delta U + \eta_s^+ - \eta_s^- - R_f \gamma I \tag{4}$$

where $\Delta U = f(\mathrm{SOC}, \mathrm{CSC})$ and

$$\eta_s^\pm = C \sinh^{-1}\left(\frac{\theta^\pm}{\sqrt{z(1-z)}} I\right),$$

where $z = \mathrm{CSC}^+ = \rho\mathrm{SOC} + \sigma$ for the positive component and $z = \mathrm{CSC}$ for the negative one, and C and $\theta^\pm$ are constant parameters. All parameters and functions of the EHM are chemistry dependent.

In order to avoid the main side reactions that compromise battery life and its safe operation, the battery overpotentials should be restricted through constraints. These constraints are in general nonlinear, which may result in solution spaces that are nonconvex (Romagnoli et al., 2017). However, it is possible to convexify the solution space with more conservative linear constraints of the form:

$$I \geq \alpha_i \mathrm{CSC} + \beta_i, \quad i = 1, \ldots, n_c \tag{5}$$

where α_i and β_i are the parameters associated to the linear approximations of the nonlinear side reaction constraints, and n_c is the considered number of linear constraints. In this work, only two linear approximations are used to describe the nonconvex solution space.

Besides electrochemical side reactions, there are operational limits of the electrode materials that need to be respected. Introducing or extracting more Li-ions that the allowed limit $\mathrm{SOC}_{\max}$ results in accelerated battery degradation (Tang et al., 2009; Hausbrand et al., 2015). Finally, a maximum input current $I_{\max}$ could also be imposed, which represents a given maximum C-rate, which is a multiple of the current that charges the battery in 1 hour. If the capacity of the battery is 34 $[\mathrm{Ah}\cdot\mathrm{m}^{-2}]$, then a C-rate 1C corresponds to -34 $[\mathrm{A}\cdot\mathrm{m}^{-2}]$, 3C to -102 $[\mathrm{A}\cdot\mathrm{m}^{-2}]$, and so on.

These constraints take the following form for the charging process

$$\mathrm{SOC} \leq \mathrm{SOC}_{\max}, \ \mathrm{CSC} \leq \mathrm{SOC}_{\max} \tag{6}$$

$$\mathrm{SOC}^{+} \geq \mathrm{SOC}^{+}_{\max}, \ \mathrm{CSC}^{+} \geq \mathrm{SOC}^{+}_{\max} \tag{7}$$

$$I \leq I_{\max} \tag{8}$$

2.2 Thermal Model

The previous reduced-order electrochemical model can be augmented to include the effect on the cell temperature of the heat generated when charging or discharging the battery. The augmented model is nonlinear because the heat generated depends on the product of voltage and current, both variables of the system. The first order differential equation governing the thermal dynamics can be written as

$$m_{\mathrm{cell}} C_{\mathrm{p,cell}} \frac{dT_{\mathrm{cell}}}{dt} = hA(T_{\mathrm{cell}} - T_{\mathrm{amb}}) + \dot{Q}_{\mathrm{gen}} \tag{9}$$

where T_{cell} is the cell temperature, m_{cell} is the cell mass, $C_{\mathrm{p,cell}}$ the specific heat, h is the overall heat transfer coefficient (accounting for convection and conduction), A is the heat exchange area, T_{amb} is the ambient temperature, and $\dot{Q}_{\mathrm{gen}}$ is the heat generated by the cell.

The thermal model for simulation is based on (Onda et al., 2003), where the total heat generated is defined as

$$\dot{Q}_{\mathrm{gen}} = I(\Delta U - V - T_{\mathrm{ref}} \frac{d(\Delta U)}{dT}) \tag{10}$$

where T_{ref} is a reference temperature, and $\frac{d(\Delta U)}{dT}$ can be calculated as the entropy change ΔS divided by the Faraday constant (F). The value of ΔS is based on a $\mathrm{LiNiCoO_2}$ pouch cell (Uddin et al., 2014).

The cell ohmic resistance R_{int} can also be used to calculate the heat generated as $\dot{Q}_{\mathrm{gen}} = I^2 R_{\mathrm{int}}$. In this work, and for the benefit of the predictive controller proposed in the following section, this expression of the heat is used in combination with the one-dimensional thermal model of the battery to approximate the heat exchange process.

Temperature constraints are motivated by the larger aging rates of batteries at higher temperatures as their lifespan roughly halves for each 13°C increase in average battery temperature (Keyser et al., 2017). The resulting upper constraint can be defined as follows:

$$T_{\mathrm{cell}} \leq T_{\mathrm{cell,max}} \tag{11}$$

A lower bound constraint is not considered in this work, but it would be necessary should ambient temperatures drop below 10°C.

2.3 Standard Charging Protocol

Standard charging protocols such as the constant current-constant voltage (CCCV) and its variations (Keil and Jossen, 2016) rely only on voltage measurements to reach a desired SOC. A typical CCCV protocol consists of a charge period under constant current (CC), followed by a constant voltage (CV) stage. The CV stage begins when a predefined voltage threshold is reached, and terminates when either a fixed maximum duration time or a minimum current threshold is achieved. The CV stage can be driven by a proportional feedback controller which in practice leads to a progressive reduction of the current as the reference voltage is reached; this is the approach followed in the present study.

2.4 Constrained Control

A widely used strategy to cope with constrained control problems is MPC (Camacho and Bordons, 2004). Model predictive controllers calculate the future control actions on the process, $u(t)$, by solving an on-line optimization problem subject to constraints that can be written as

$$\begin{aligned} \min_{u(t)} \quad & \int_{t_0}^{t_\mathrm{f}} (\mathrm{SOC}(t) - \mathrm{SOC}_{\mathrm{ref}})^2 dt \\ \text{s.t.} \quad & \text{model dynamics} \\ & \text{electrochemical constraints} \\ & \text{thermal constraints} \end{aligned} \tag{12}$$

The optimization problem (12) minimizes a cost function that depends on the predicted tracking error (given a desired reference $\mathrm{SOC}_{\mathrm{ref}}$) and the control effort. The former is the difference between the SOC and its reference. The latter depends on $u(t)$, which is the sequence of control actions on the system, i.e., the applied current. The time horizon $(t_\mathrm{f} - t_0)$ is discretized into a finite number of time steps in which the variables of the problem are defined. Constraints include the predictions performed with a simplified battery model, as well as upper and lower bounds on states, outputs, input, or a combination of these.

3 Case Studies and Implementation

Figure 2 shows the control scheme adopted. The controller, which is either based on the CCCV protocol or on MPC, can receive information regarding the reference, outputs and measured disturbances. It is assumed in this work that the effect of the disturbances is negligible.

Table 1. CCCV controller

Variable	*Value*	*Units*
I_{end}	-0.15	$[\mathrm{A \cdot m^{-2}}]$
V_{thres}	4.19	[V]
V_{ref}	4.2	[V]
$\mathrm{K_p}$	-5250	$[\mathrm{A \cdot V^{-1}}]$
$\mathrm{K_i}$	0	$[\mathrm{A \cdot (V \cdot s)^{-1}}]$
$\Delta t_{\mathrm{sample}}$	1	[s]
Δt_{sim}	0.1	[s]

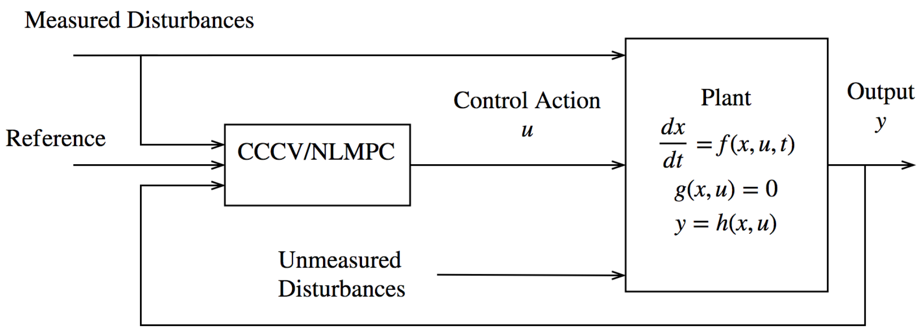


Figure 2. Feedback control loop including different controllers and the plant model.

Regarding the CCCV protocol, the stopping conditions and setup parameters for the proportional controller used are shown in Table 1. I_{end} and V_{thres} are the termination current and the voltage threshold respectively. Δt coincides with the simulation time step, and is the rate at which the controller is implemented. $\mathrm{K_P}$ and $\mathrm{K_I}$ are the proportional and integral coefficients for the PI controller, and V_{ref} define the voltage reference to calculate the error. In order to ensure a smooth transition between the CC and CV stages, the applied current in the later was filtered using the arithmetic mean of the calculated and a number of past applied currents (this number is 9 in this work).

Table 2. NMPC controller

Variable	*Value*	*Units*
t_{f}	200	[s]
$\mathrm{SOC_{ref}}$	0.665	[-]
$\mathrm{n_e}$	20	[-]
$\mathrm{n_{cp}}$	1	[-]
H	200	[s]
Δt_{MPC}	10	[s]
Δt_{sim}	1	[s]
solver	IPOPT	

The implementation in Modelica of both the plant model (Appendix I), used for simulation, as well as the prediction model (Appendix II, including constraints) are included at the end of the document. Both are based on experimental data obtained for a commercial LCO batteries (Turnigy Nano-tech, 160mAh (Turnigy, 2018)). The optimal control problem was solved with the programming language Python and JModelica.org. The latter is an open source platform for optimization, simulation and analysis of complex dynamic systems (Link et al., 2015). It interfaces the numerical solver IPOPT and CasADi, which is an open source symbolic framework for automatic differentiation and optimal control (Andersson et al., 2012). The Python program to compute the MPC problem is also included in Appendix III. Table 2 shows the settings for the NMPC algorithm, which is solved using the interior point method, where $\mathrm{n_e}$ is the number of finite elements,

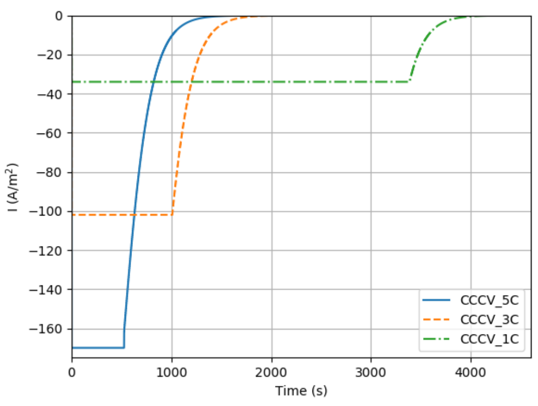


Figure 3. Current profiles for CCCV protocols (1C, 3C and 5C).

$\mathrm{n_{cp}}$ is the number of collocation points in each element, H is the control horizon, Δt_{MPC} is the sampling time and the length of each time interval for the MPC, Δt_{sim} is the simulation timestep.

4 Results

In this section, we present results regarding current, temperature, voltage and critical surface concentration. The values for the current applied are shown in the units of the model ($\mathrm{I \cdot m^{-2}}$), but are also referred to in terms of the equivalent C-rate.

4.1 CCCV Protocols

The results of the application of three CCCV charge profiles with increasing constant current are discussed first. Figure 3 presents the current profiles, calculated by the proportional controller and limited by the prescribed CC current.

As expected, the increase of the CC rate reduces the charging time: around 4165 s for 1C, 1975 s for 3C, and 1590 s for 5C. These values depend on the battery chemistry, the controller setup, and the termination conditions. Figure 4 shows the consequence of using charging protocols that overlook the existence of an electrochemically safe operating region. The grey area denotes the combination of I and CSC where side reactions are triggered.

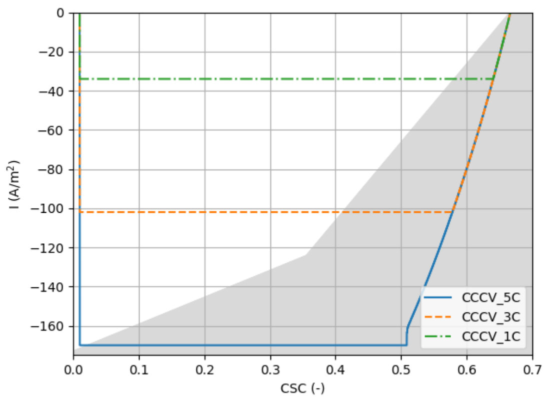


Figure 4. Charging trajectories in the I-CSC plane for CCCV protocols.

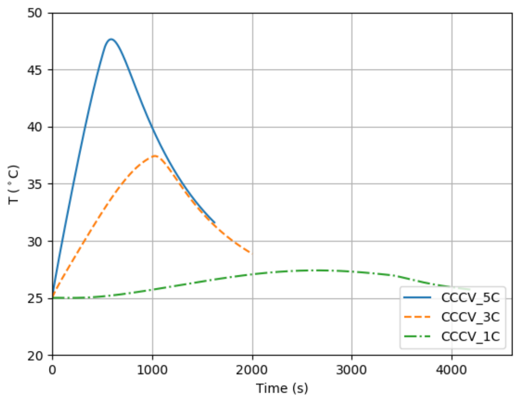


Figure 6. Temperature profiles for CCCV protocols (1C, 3C and 5C).

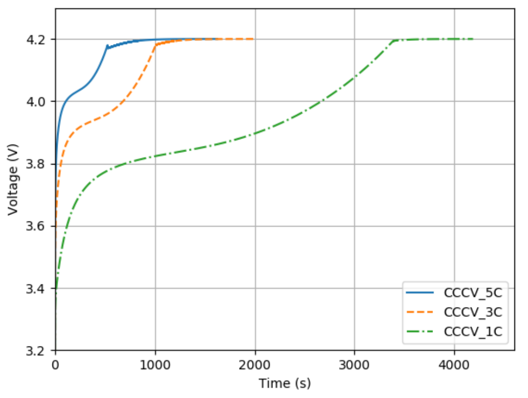


Figure 5. Voltage profiles for CCCV protocols (1C, 3C and 5C).

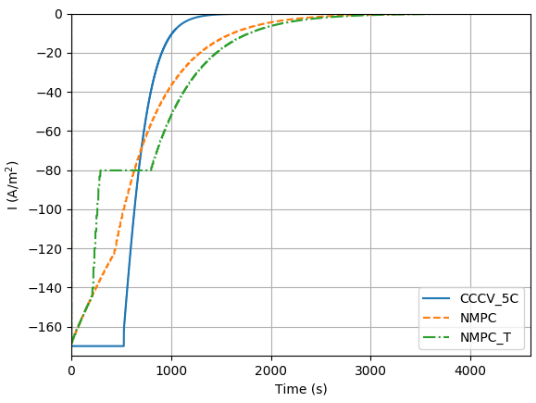


Figure 7. Current profiles for the nonlinear controllers and the 5C-CCCV protocol.

Figures 5 and 6 show the respective results for voltage and temperature. The first figure illustrates the effect of the proportional controller, which takes the same parameteres for the three CCCVs. Regarding the temperature, as expected, the maximum temperature achieved varies significantly with the current applied. However, an equally important fact is that the higher the Crate of the CC stage, the longer the cell's temperature remain above ambient temperatures. This means that calendar aging, i.e., the aging that takes place with zero current conditions and which depends on the cell temperature, will be higher for the higher C-rate CCCVs. For 1C CCCV the temperature increase remains below 5°C. Given the relationship between aging, current and temperature, it is not surprising that most manufacturers recommend the use of low C-rates for charging (below 1C).

4.2 Nonlinear MPC

In this section, we compare the previous results for 5C CCCV with the thermally unconstrained (NMPC) and constrained (NMPC_T) optimal controllers. For the latter, the maximum temperature allowed ($T_{\max}$) is 35°C. This choice is arbitrary, but well below the maximum temperature achieved by the 5C CCCV, and so suitable for illustrating the methodology. In practice, to properly select the boundary one should have an estimation of its impact in the battery's long-term performance, so to measure the economic gains from setting such a boundary. This issue is not addressed in the present paper.

Figure 7 shows the current profiles. Regarding the optimized profiles, they remain qualitatively close for most of the charge, except for an interval in which the current is decreased to enforce the temperature constraints for the NMPC_T, which slightly increases the charging time.

Figure 8 shows the admissible charging region, now with the NMPC and the NMPC_T profiles. Both remain electrochemically feasible throughout the charging process but, as a consequence of the temperature constraints, the NMPC_T departs from the electrochemical convex hull. In this figure, as well as in Fig. 7, it can be observed that the NMPC_T resembles a CCCV with boost charge. This charging protocol can reduce significantly the life of the cell, as shown by (Keil and Jossen, 2016). In their work, the authors present experimental results considering boost current of 5C that fills 40% of the capacity, including

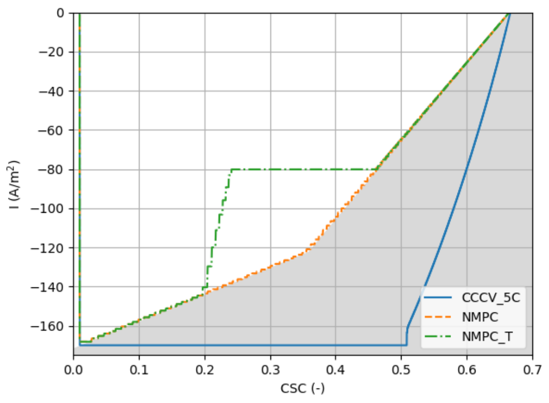


Figure 8. Charging trajectories in the I-CSC plane for nonlinear controllers and 5C-CCCV protocols.

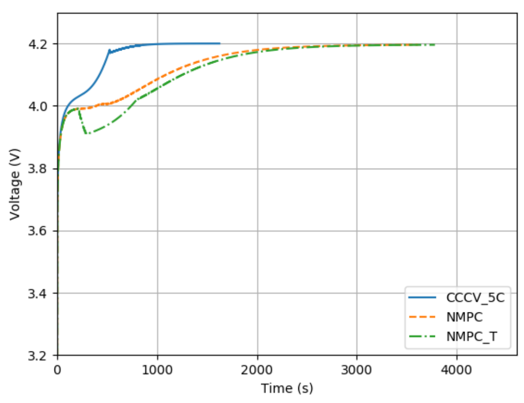


Figure 9. Voltage profiles for the nonlinear controllers and the 5C-CCCV protocol.

the following intervals: 0%-40%, 10%-50%, 20%-60%. The first interval results in the highest degradation, which is explained as a higher cell resistance at low SOC. Nevertheless, these three boost-charge profiles bring the 5C current into the region of side reactions, similar to what results with the 5C CCCV in the present paper (Fig. 5).

Finally, results for voltage and temperature are presented. As anticipated in the previous subsection, the electrochemical constraints limit the rate at which the voltage approaches the SOC reference. Regarding the temperature profiles, a $\Delta T = 4°$C reduction is obtained just by imposing electrochemical constraints with a further decrease when activating the thermal constraints. Taking into account that aging increases exponentially with temperature, these differences could improve the battery life at the expense of increasing the charging times. These are 3520 s for the thermally unconstrained, and 3720 s for the thermally constrained. It is worth noting that NMPC is charged up to 80% (SOC = 0.55) in 960 s, while the NMPC_T is charged up to this SOC in 1500 s. For the model used, CSC deviates only slightly from the SOC, and thus the charging times can be obtained comparing I-CSC and I-t plots.

Solution times when solving the NMPC problem at each time interval remain close to 0.15 s, while for the thermally constrained NMPC rise slightly up to 0.20 s. In any case the total time surpassed 0.3 s. Given the nonlinear controller update rate (10 s), this ensures that in absence of great disturbances, both controllers can be applied online, if the tools needed for estimation remain in the same order of magnitude.

5 Conclusions and Future Work

The fast and safe charge of lithium-ion batteries remains an open problem. Most charging protocols rely on empirically obtained parameters, and generally result too conservative, limiting the flexibility of operation. This work presents the modeling, simulation and control efforts to better understand the open challenges. In particular a NMPC scheme was implemented in Python and JModelica.org, which provides an excellent platform to compare the results of optimization-based control methods with more traditional charging protocols that rely on relay, proportional, or PI controllers.

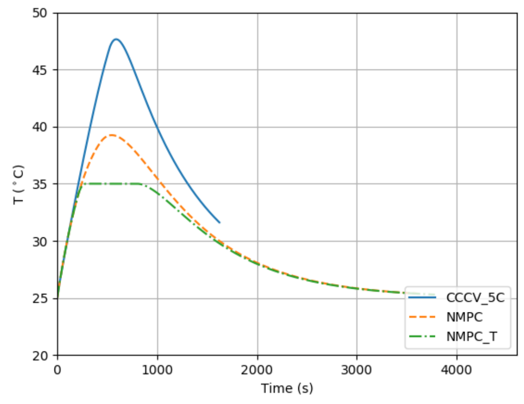


Figure 10. Temperature profiles for the nonlinear controllers and the 5C-CCCV protocol.

It can be concluded that the NMPC solver can be applied on-line, and that charging time can be reduced compared with CCCV protocols while complying with electrochemical constraints, which will presumably extend the battery life. However, adding thermal constraints can significantly limit the advantage of the optimal controller regarding the charging time. Arising from the simultaneous application of electrochemical and thermal constraints are boost-charge-like charging strategies, which make possible to charge up to 80% the battery in less than 30 min. Ongoing work focuses on time constraints to enforce a certain level of SOC within a given time, and on an optimization scheme to pre-compute electrochemically and thermal safe CCCV with boost charge.

References

Joel Andersson, Johan Åkesson, and Moritz Diehl. Casadi: A symbolic package for automatic differentiation and optimal control. In *Recent advances in algorithmic differentiation*, pages 297–307. Springer, 2012.

John Batteh and Michael Tiller. Implementation of an extended vehicle model architecture in modelica for hybrid vehicle modeling: development and applications. In *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, pages 823–832. Linköping University Electronic Press, 2009. 043.

Claude Bouvy, Sidney Baltzer, Peter Jeck, Jörg Gißing, Thomas Lichius, and Lutz Eckstein. Holistic vehicle simulation using modelica-an application on thermal management and operation strategy for electrified vehicles. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, pages 264–270. Linköping University Electronic Press, 2012. 076.

Jonathan Brembeck and Sebastian Wielgos. A real time capable battery model for electric mobility applications using optimal estimation methods. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*, pages 398–405. Linköping University Electronic Press, 2011. 063.

Eduardo F Camacho and Carlos Bordons. *Model predictive control. Advanced textbooks in control and signal processing*. Springer-Verlag, London, 2004.

L. D. Couto, J. Schorsch, M. M. Nicotra, and M. Kinnaert. Soc and soh estimation for li-ion batteries based on an equivalent hydraulic model. part i: Soc and surface concentration estimation. In *2016 American Control Conference (ACC)*, pages 4022–4028, July 2016. doi:10.1109/ACC.2016.7525553.

Luis D Couto and Michel Kinnaert. Partition-based unscented kalman filter for reconfigurable battery pack state estimation using an electrochemical model. In *2018 Annual American Control Conference (ACC)*, pages 3122–3128. IEEE, 2018.

Thanh-Son Dao and Chad Schmitke. Developing mathematical models of batteries in modelica for energy storage applications. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, pages 469–477. Linköping University Electronic Press, 2015. 118.

Madeleine Ecker, Jochen B. Gerschler, Jan Vogel, Stefan Käbitz, Friedrich Hust, Philipp Dechent, and Dirk Uwe Sauer. Development of a lifetime prediction model for lithium-ion batteries based on extended accelerated aging test data. *Journal of Power Sources*, 215:248–257, 2012. ISSN 03787753. doi:10.1016/j.jpowsour.2012.05.012.

M Einhorn, FV Conte, C Kral, C Niklas, H Popp, and J Fleig. A modelica library for simulation of electric energy storages. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*, pages 436–445. Linköping University Electronic Press, 2011. 63.

Johannes Gerl, Leonard Janczyk, Imke Krüger, and Nils Modrow. A modelica based lithium ion battery model. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, pages 335–341. Linköping University Electronic Press, 2014. 096.

R. Hausbrand, G. Cherkashinin, H. Ehrenberg, M. Gröting, K. Albe, C. Hess, and W. Jaegermann. Fundamental degradation mechanisms of layered oxide Li-ion battery cathode materials: Methodology, insights and novel approaches. *Materials Science and Engineering: B*, 192:3–25, 2015.

Leonard Janczyk, Klemens Esterle, Stephan Diehl, Michael Seibt, Arthur Gauthier, and Viry Guillaume. Validation of a battery management system based on autosar via fmi on a hil platform. In *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*, pages 87–94. Linköping University Electronic Press, 2016. 124.

Peter Keil and Andreas Jossen. Charging protocols for lithium-ion batteries and their impact on cycle life-An experimental study with different 18650 high-power cells. *Journal of Energy Storage*, 6:125–141, 2016.

Matthew Keyser et al. Enabling fast charging Battery thermal considerations. *Journal of Power Sources*, 367:228–236, 2017.

Kilian Link, Leo Gall, Monika Mühlbauer, and Stephanie Gallardo-Yances. Experience with industrial in-house application of fmi. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, pages 17–22. Linköping University Electronic Press, 2015. 118.

James F. Manwell and Jon G. McGowan. Lead acid battery storage model for hybrid energy systems. *Solar Energy*, 50(5):399 – 405, 1993. ISSN 0038-092X. doi:https://doi.org/10.1016/0038-092X(93)90060-2.

AB Modelon. Jmodelica. org user guide, verison 2.2, 2018.

Scott J Moura, Nalin A Chaturvedi, and M Krstić. Constraint management in li-ion batteries: A modified reference governor approach. In *American Control Conference (ACC), 2013*, pages 5332–5337. IEEE, 2013.

Kazuo Onda, Hisashi Kameyama, Takeshi Hanamoto, and Kohei Ito. Experimental study on heat generation behavior of small lithium-ion secondary batteries. *Journal of the Electrochemical Society*, 150(3):A285–A291, 2003.

R. Romagnoli, L. D. Couto, M. M. Nicotra, M. Kinnaert, and E. Garone. Computationally-efficient constrained control of the state-of-charge of a li-ion battery cell. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1433–1439, Dec 2017.

Jonathan Spike, Johannes Friebe, Chad Schmitke, Christian Donn, Michael Folie, Valerie Bensch, and Christine Schwarz. Holistic virtual testing and analysis of a concept hybrid electric vehicle model. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, pages 537–545. Linköping University Electronic Press, 2015. 118.

Maureen Tang, Paul Albertus, and John Newman. Two-Dimensional Modeling of Lithium Deposition during Cell Charging. *Journal of The Electrochemical Society*, 156(5): A390–A399, 2009.

Turnigy, 2018. URL `http://www.turnigy.com/batteries/nano-tech/`.

Kotub Uddin and Alessandro Picarelli. Phenomenological li ion battery modelling in dymola. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, pages 327–334. Linköping University Electronic Press, 2014. 96.

Kotub Uddin, Alessandro Picarelli, Christopher Lyness, Nigel Taylor, and James Marco. An acausal Li-ion battery pack model for automotive applications. *Energies*, 7(9):5675–5700, 2014. doi:10.3390/en7095675.

John Wang, Ping Liu, Jocelyn Hicks-Garner, Elena Sherman, Souren Soukiazian, Mark Verbrugge, Harshad Tataria, James Musser, and Peter Finamore. Cycle-life model for graphite-LiFePO4 cells. *Journal of Power Sources*, 196(8):3942–3948, 2011. ISSN 03787753. doi:10.1016/j.jpowsour.2010.11.134. URL `http://dx.doi.org/10.1016/j.jpowsour.2010.11.134`.

Appendix I. Battery plant model

```
model EHMcell

  // Heat transfer problem parameters
  parameter Modelica.SIunits.Resistance R =
       0.330 "Internal resistance";
  parameter
      Modelica.SIunits.CoefficientOfHeatTransf
       h = 10 "Heat transfer [W/m2.K]";
  parameter Modelica.SIunits.Area A = 4.7
      e-4 "Heat transfer area [m2]";
  parameter Modelica.SIunits.Mass M =
      0.0045 "Cell mass [kg]";
  parameter
      Modelica.SIunits.SpecificHeatCapacity
       Cp = 800 "Cell specific heat [J/kg.K
      ]";

  // Electrochemical parameters
  parameter Real gamma = 0.0000054581;
  parameter Real g = 0.042653404659871;
  parameter Real beta = 0.7;
  parameter Real Ccell = 0.16 "Battery
      capacity [Ah]";
  parameter Real p_oneC = 34 "1C current
      flux [A/m2]";
  parameter Real p_alpha = 0.5;
  parameter Real p_Faraday(unit = "C/mol")
      = 96487 "Faraday constant";
  parameter Real p_R = 8.314472;
  parameter Real p_T_ref = 298.15;
  parameter Real p_aFRT = (p_alpha*
      p_Faraday)/(p_R*p_T_ref);
  parameter Real p_thetap =
      0.003753150372049;
  parameter Real p_thetan
      =0.002390386732068;
  parameter Real p_Rf = 0.000846744769459;
  parameter Real p_ro = 0.798857289559742;
  parameter Real p_sigma =
      1.001138873133620;

  // Constraint parameters
  parameter Real alpha1 = 137.436438945007;
  parameter Real alpha2 = 858.977743406295;
  parameter Real beta1 = -171.795548681259;
  parameter Real beta2 = -532.566200911903;

  // The states and variables
  Real SOC(start = 0.01) "State of charge";
  Real CSC(start = 0.01) "Critical surface
      concentration";
  Real CSCp(start = 0.01) "CSC
      Overpotential positive electrode";
  Modelica.SIunits.Temp_C T(start = 25.) "
      Battery temperature";
  Modelica.SIunits.Voltage Up(start = 0.1)
      "Surface overpotential positive
      electrode";
  Modelica.SIunits.Voltage Un(start = 3.5)
      "Surface overpotential negative
      electrode";
  Modelica.SIunits.Voltage V(start = 3.5) "
      Battery voltage";
  Modelica.SIunits.MolarEntropy DS "Entropy
       change [J/mol.K]";

  // The control signal
  input Modelica.SIunits.ElectricCurrent I
      "Input current flux [A/m2]";

equation
  der(SOC) = - gamma * I;
  der(CSC) = g/beta/(1-beta) * SOC - g/beta
      /(1-beta) * CSC - gamma * I/(1-beta)
      ;
  der(T) = ((-I/(p_oneC/Ccell)*(V - (-Un +
      Up) + p_T_ref*DS/p_Faraday) ) - h*A*(
      T - 25))/M/Cp;
  DS = 0.5609*1e+3*(SOC/0.68)^5 -1.3440*1e
      +3*(SOC/0.68)^4 + 1.1877*1e+3*(SOC
      /0.68)^3 -0.6072*1e+3*(SOC/0.68)^2 +
      0.2378*1e+3*(SOC/0.68)^1 -0.0397*1e
      +3*(SOC/0.68)^0 ;
  CSCp = -p_ro * SOC + p_sigma;
  Up = (0.654807602368402*((1-CSCp)
      .^3.196972561445755))+ 3.85516954 +
      1.247319422*(1-CSCp) -
      11.15240126*(1-CSCp).^2 +
      42.8184855*(1-CSCp).^3 -
      67.71099749*(1-CSCp).^4 +
      42.50815332*(1-CSCp).^5 - 6.13244713
      e-4*Modelica.Math.exp(-7.657419995*(
      CSCp.^115.0));
  Un = 8.002296379 + 5.064722977*CSC -
      12.57808059*CSC.^(1/2) - 8.632208755
      e-4*CSC.^(-1) + 2.176468281e-5*CSC.
      ^(3/2) - 0.4601573522*
      Modelica.Math.exp(15.0*(0.06 - CSC))
      - 0.5536351675*Modelica.Math.exp
      (-2.432630003*(CSC - 0.92));
  V = -Un + Up -(p_Rf*I) - (1/p_aFRT)*(
      Modelica.Math.asinh(( 1*p_thetan*I)/
      sqrt(CSC*(1-CSC)))) + (1/p_aFRT)*(
      Modelica.Math.asinh((-1*p_thetap*I)/
      sqrt(CSCp*(1-CSCp))));

end EHMcell;
```

Appendix II. Battery optimization model

```
optimization EHMTV_Opt (objectiveIntegrand
    = (SOC - 0.665)^2,
                 startTime = 0,
                 finalTime = 200)

  // Heat transfer problem parameters
  // ...

  // Electrochemical parameters
  // ...

  // Constraint parameters
  // ...

  // The states and variables
  Real SOC(start = 0.01) "State of charge";
  Real CSC(start = 0.01) "Critical surface
      concentration";
  Real CSCp(start = 0.01) "CSC
      Overpotential positive electrode";
  Modelica.SIunits.Temp_C T(start = 25.) "
      Battery temperature";
  Modelica.SIunits.Voltage Up(start = 0.1)
      "Surface overpotential positive
      electrode";
  Modelica.SIunits.Voltage Un(start = 3.5)
      "Surface overpotential negative
      electrode";
  Modelica.SIunits.Voltage V(start = 3.5) "
      Battery voltage";
  Modelica.SIunits.MolarEntropy DS "Entropy
       change [J/mol.K]";

  // The control signal
  input Modelica.SIunits.ElectricCurrent I
      "Input current flux [A/m2]";

equation
  // ...
  der(T) = (R*(I/(p_oneC/Ccell))^2 - h*A*(T
       - 25))/M/Cp;
  // ...

constraint
  SOC <= 0.665 + 0*0.627319647304968;
  CSC <= 0.665 + 0*0.627319647304968;
  SOC >= 0.001;
  CSC >= 0.001;
  I >= alpha1 * CSC + beta1;
  I >= alpha2 * CSC + beta2;
  I >= -34*5;
  I <= 0.;
  T <= 300;
  V <= 4.5;

end EHMTV_Opt;
```

Appendix III. Python implementation of the MPC, based on JModelica.org User Guide (Modelon, 2018)

```
# // Program that demonstrate the native
    FMI interface for simulation
# // Integration takes place within python

import numpy as N
import pylab as plt
from pymodelica import compile_fmu
from pyfmi.fmi import load_fmu

# // Import the function for compilation of
     models and the load_fmu method
from pyjmi import
    transfer_optimization_problem

# // Compile model and load FMU
fmu_name = compile_fmu("EHMcell", "C:\...\
    EHMcell_old.mo")
model = load_fmu(fmu_name)

# // Transfer the optimization problem to
    casidi
# // This function transfers the
    optimization problem into Python
# //   and expresses its variables,
    equations, etc., using the
# //   automatic differentiation tool
    CasADi.
op = transfer_optimization_problem("
    EHMTV_Opt", "C:\...\EHMTV_Opt.mop")

# // Optimization options
opts = op.optimize_options()
opts['n_e'] = 20
opts['n_cp'] = 1
opts['solver'] = 'IPOPT'
opts['expand_to_sx'] = 'NLP'

# // Simulation times and model
    initialization
Tstart = 0
Tend = 3600+1000
model.time = Tstart
I0 = -1.
I02 = I0
model.set('I', I0)
h0 = 10.
model.set('h', h0)
model.initialize()

# // Data to be stored in the integration
    loop
# // Get continous States
x = model.continuous_states
# // Get the Nominal Values
x_nominal = model.nominal_continuous_states
# // Get the Event Indicators
event_ind = model.get_event_indicators()

# // Values for the solution
# // Retrieve the valureferences for the
    values 'SOC', 'CSC' and 'T'
states0 = [model.get_variable_valueref('SOC
    ')] + [model.get_variable_valueref('CSC
    ')] + [model.get_variable_valueref('T')
    ]
vol0 = model.get_variable_valueref('V')
voltage = [model.get_real(vol0)]
vol = model.get_real(vol0)
t_sol = [Tstart]
sol = [model.get_real(states0)]
I_sol = [I0]

# // Initialize integration time and define
     the step-size
time = Tstart
Tnext = Tend    # Used for time events
dt = 1 # Step-size
Iend = -0.15

# // Main integration loop using explicit
    Euler method.
# // This is the integration loop for
    advancing the solution one step at a
    time.
# // The loop continues until the final
    time has been reached or
# //   if the FMU reported that the
    simulation is to be terminated.

count = 0
while time < Tend and not
    model.get_event_info().
    terminateSimulation:

    # // Compute the derivative of
        theprevious step f(x(n), t(n))
    dx = model.get_derivatives()

    # // Advance
    h = min(dt, Tnext - time)
    time = time + h

    # // Set the time
    model.time = time

    # // Set the states at t = time
    # // Perform the step using x(n+1) = x(
        n) + h*f(x(n), t(n)))
    x = x + h*dx

        # // To make sure that the plant
            does not feed back infeasible
            states
    if x[2] >30 and x[2]<31:
        x[2]=30
    if x[2] > 31:
        break

    model.continuous_states = x

    # // Retrieve solutions at t = time for
         outputs
    # // model.get_real, get_integer,
        get_boolean, get_string(valueref)

    t_sol += [time]
    sol += [model.get_real(states0)]
    vol = model.get_real(vol0)
    voltage += [vol]

    if vol > 4.5:
        I0 = 0.;
        model.set('I', I0)

    op.set('SOC0',float(N.array(sol)[-1,0])
        )
    op.set('CSC0',float(N.array(sol)[-1,1])
        )
    op.set('CSCp0',float(N.array(sol)
        [-1,1]))
    op.set('T0',float(N.array(sol)[-1,2]))

    clause = count \% 10
    if  clause <= 1e-5:
        resopt = op.optimize(options = opts
            )
        Iopt = resopt['I']
        print Iopt[0]
        if count > 100:
            I0 = Iopt[0]
        else:
            I0 = Iopt[0]
            model.set('I', I0)

    I_sol += [I0]
    count=count+1
    if I0 >= Iend:
        break

# // Plots...
```

Session 3A: HVAC

Modeling Heat Pump Recharge of a Personal Conditioning System with Latent Heat Storage

Rohit Dhumane Jiazhen Ling Vikrant Aute Reinhard Radermacher

Center for Environmental Energy Engineering, Department of Mechanical Engineering, University of Maryland, College Park, MD, USA 20742

dhumane@terpmail.umd.edu; {jiazhen,vikrant,raderm}@umd.edu

Abstract

Roving Comforter provides personal cooling in the range of 150 W using vapor compression cycle (VCC) up to 4 hours. During this operation, the condenser heat is stored in a latent heat storage made of phase change material (PCM). This heat needs to be discharged before next cooling operation. A heat pump mode is considered and analyzed for this heat discharge in the present article. The cycle is modeled using `CEEEModelicaLibrary`, which is a commercial package for complex vapor compression systems. Equations and assumptions involved in modeling some of the components is presented. Programming and modeling decisions for PCM modeling are discussed in detail. A parametric study is conducted with the heat pump system model to identify merits and demerits of operating heat pump cycle at various compressor RPM's.

Keywords: Heat Pump, PCM, Latent Storage, Personal Comfort

1 Introduction

World Wildlife Fund recently reported "We are the first generation to know we are destroying our planet and the last one that can do anything about it." The lifestyle choices in the modern world are not sustainable and need a thorough scrutiny.

Space heating and cooling takes up significant portion of worldwide primary energy consumption. The current practice of maintaining conditioned space temperature within a narrow temperature range on the one hand, does consume a huge amount of energy to condition the total space in the building, including the unoccupied space; on the other hand, cannot always guarantee the comfort of 80% occupants due to the individual preference (Zhang, Arens and Zhai, 2015). The temperature range for comfort were developed based on predictive mean vote or adaptive models and is not ideal for individual comfort (Kim, Schiavon and Brager, 2018). Personal conditioning system allow building temperature set-points to be elevated without compromising thermal comfort. Space heating and cooling energy consumption can be reduced to the order of 10% per °C elevation in temperature set-points of buildings (Hoyt, Arens and Zhang, 2015).

Portable personal comfort systems have potential to provide improved thermal comfort, reduced energy consumption and ways to mitigate demerits of existing personal comfort systems (Dhumane *et al.*, 2017). Roving Comforter (RoCo) is being developed to provide better individual thermal comfort at much lesser energy consumption. The first prototype (Dhumane *et al.*, 2019) used paraffin based phase change material for latent heat storage. For recharge operation, a thermosiphon operation was used (Dhumane *et al.*, 2018). The technology can provide annual energy savings per person up to $130 (Heidarinejad *et al.*, 2018).

The current prototype (see Figure 1) provides 4 hours of cooling, but requires slightly more than 6 hours to discharge heat from the PCM storage by thermosiphon operation. A heat pump operation can provide faster recharge than thermosiphon, but consume significantly higher power. It is necessary to model this operation to understand it quantitatively and also identify ideal operating parameters. Modeling this heat pump operation is the objective of the present article.

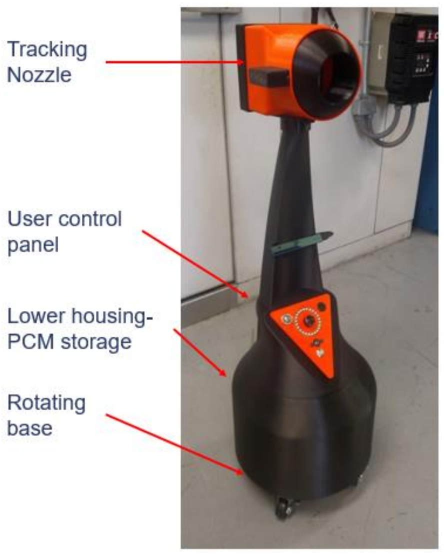


Figure 1. Current Prototype of Roving Comforter.

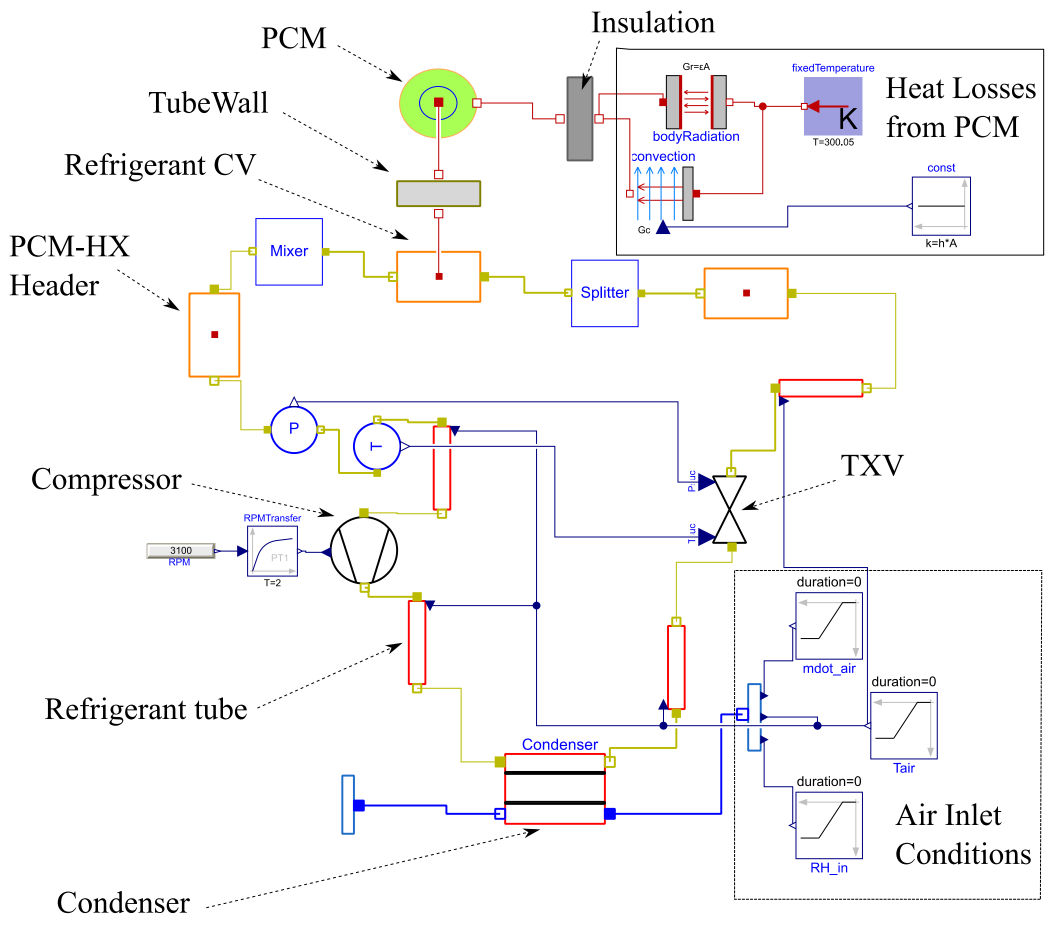


Figure 2. Model diagram for heat pump recharge

2 HEAT PUMP RECHARGE

The schematic of the heat pump operation of RoCo is shown in Figure 3. The schematic for cooling mode is shown in Figure 4. The grey portion of the refrigerant circuit does not have any refrigerant flow. A reversible four-way valve is used to control the refrigerant flow direction and switch the operating conditions between cooling and PCM recharging (solidification). To be more specific, the four-way valve directs the discharge refrigerant from the compressor to the PCM HX in the cooling while directs the discharged refrigerant to the air-to-refrigerant Condenser (as shown in the figure) in the PCM recharging. The compressor used in RoCo is of the variable-speed type and the recharge time may be controlled by adjusting its RPM. The recharge time can be reduced with higher RPM but the power consumption is, as a result of higher RPM, increased. There are two TXVs in the circuit for heat pump, each protected by a check valve which ensures only one TXV operates in each mode. The bulbs of the TXV are connected to outlet of either the air to refrigerant heat exchanger or the PCM-HX. During the cooling operation, the outlet of air to refrigerant heat exchanger (marked as condenser in Figure 3) is connected to the bulb of TXV, while the outlet of PCM-HX is connected to the bulb of the second TXV. This second TXV is operational during the heat pump operation.

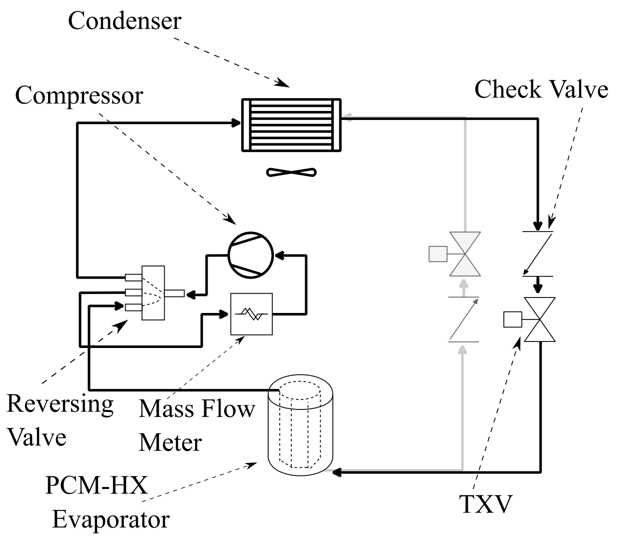


Figure 3. Schematic of Heat Pump Operation.

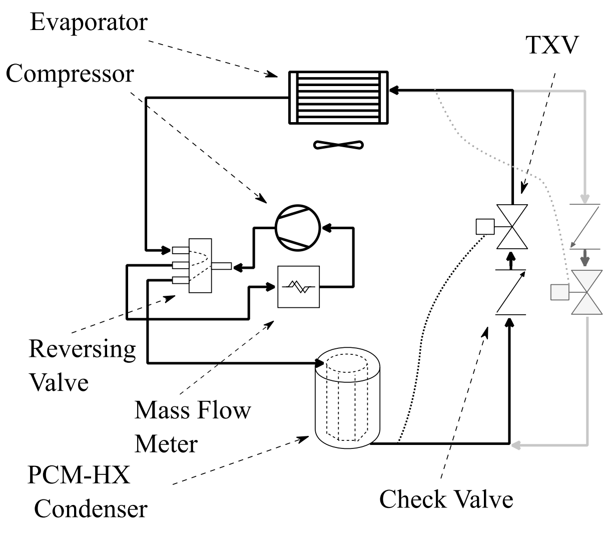


Figure 4. Schematic of Cooling Operation.

3 MODEL DEVELOPMENT

The model diagram for the heat pump model is shown in Figure 2. The component models are obtained from CEEEModelicaLibrary (CML), which has been discussed in detail in Qiao (2014). The components used in the present investigation have been updated to stream connectors (Franke *et al.*, 2009). The modifications for heat transfer coefficient and pressure drop characteristics as suggested by Dermont *et al.* (2016) have been incorporated. The PCM model is developed for the present investigation and is discussed in detail. Other components are described briefly and important input parameters for each of them are presented in this section.

3.1 Phase Change Material Storage

The PCM storage used for the prototype involves graphite enhancement to paraffin based PureTemp 37. The bulk density of the material is 183 kg/m^3. The PCM storage is cylindrical with diameter = 0.254 m and height = 0.305 m. Eight 0.0063 m copper tubes are inserted at a distance of 0.051 m from the outer diameter, with headers connecting at bottom and the top. The symmetric location of the refrigerant tubes is exploited to allow simplification in modeling.

The cross-section of the PCM storage is shown below in Figure 5. The graphite foam prevents circulation of liquid PCM during phase change and the control volume can be modeled as a pure conduction problem. Radial mode of heat transfer is assumed to be dominant to avoid discretization along the height of the PCM cylinder. The symmetric location of the refrigerant tubes allows further simplification in modeling. Only a single tube is modeled and the behavior of the entire thermal storage is captured by scaling the behavior of this single section by eight. This is done using the `Splitter` and `Mixer` components from `CML`. The PCM surrounding each refrigerant tube is in the shape of a 1/8 pie. However, the control volume is assumed to be cylindrical to allow modeling as a 1-D conduction problem. This assumption is shown in Figure 5.

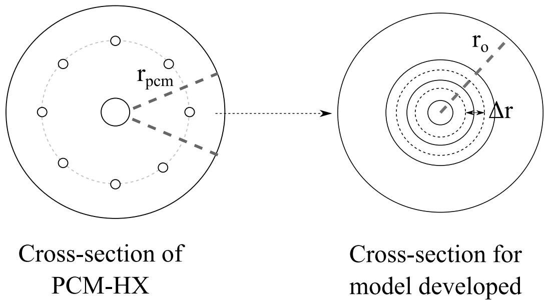


Figure 5. Cross section for modeling PCM.

The energy equation for 1-D cylindrical conduction with the assumptions mentioned above is shown in Equation 1. Here ρ is the average of the density of PCM in solid and liquid phases. The difference between these two values is less than 10% and this assumption allows treatment of density as a constant for the differential equation. Similarly, the thermal conductivity (k) is also treated constant for both phases. The difference is negligible in reality since the majority of heat transfer in graphite enhanced PCM is via graphite.

$$\rho \frac{\partial h}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(kr\frac{\partial T}{\partial r}\right) \tag{1}$$

The specific enthalpy (h) of the PCM is estimated using temperature transforming model of Cao and Faghri (1991). As shown in Equation 2, the specific enthalpy is written as a function of specific heat capacity (c), a source term (s) and temperature difference from the melting point (T_m). The paraffin based PCM typically melts over a temperature range and this allows values of specific heat capacity in two-phase region to be defined (not equal to infinity as in the case of pure materials like water). Temperature (T_m) is the mid-point of this temperature glide during phase change. The specific heat capacity and source terms are calculated using Equations 3 and 4.

$$h = c\left(T - T_m\right) + s \tag{2}$$

$$c = \begin{cases} c_s & \text{T-T}_m < -\delta\text{T} \quad (\text{solid}) \\ \dfrac{c_s + c_l}{2} + \dfrac{H}{2\delta T} & -\delta\text{T} \le \text{T-T}_m \le \delta\text{T} \quad (\text{two-phase}) \\ c_l & \text{T-T}_m > \delta\text{T} \quad (\text{liquid}) \end{cases} \tag{3}$$

$$s = \begin{cases} c_s\delta\text{T} & \text{T-T}_m < -\delta\text{T} \quad (\text{solid}) \\ \left(\dfrac{c_s + c_l}{2}\right)\delta\text{T} + \dfrac{H}{2} & -\delta\text{T} \le \text{T-T}_m \le \delta\text{T} \quad (\text{two-phase}) \\ c_s\delta\text{T} + H & \text{T-T}_m > \delta\text{T} \quad (\text{liquid}) \end{cases} \tag{4}$$

In these equations, H is the latent heat capacity of the PCM, δT is the temperature glide and c_s and c_l are solid phase and liquid phase specific heat capacity values.

Finite volume method is used to model the cylindrical control volume of the PCM. A staggered grid is adopted with the mass and energy of the PCM assumed to be concentrated at the center, while the heat fluxes calculated the boundary. The number of discretization for the control volume is 5.

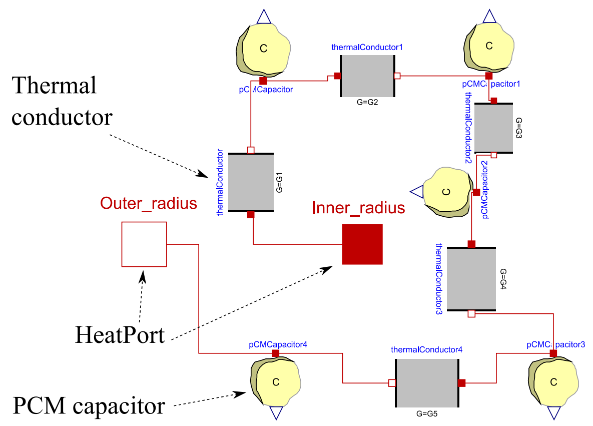


Figure 6. Model diagram for discretized PCM control volume.

The staggered grid is implemented using an alternating network of `Modelica.Thermal.HeatTransfer.Components.ThermalConductor` and `PCMConductor`. The arrangement is shown in Figure 6. Each of these components may also be created as vector arrays and connected using for loops for generality. The connections between these components are made using `HeatPort` connector.

3.1.1 Heat Transfer

The `ThermalConductor` requires an input of thermal conductance (G). For the first control volume (closest to the refrigerant tube in the center of cylinder), it can be calculated using Equation 5.

$$G_1 = \frac{k\left(2\pi\left(\frac{r_1 + r_{1.5}}{2}\right)L\right)}{\Delta r / 2} \tag{5}$$

The index i = 1, for the refrigerant tube outer diameter and for discretization of 5, the outer diameter of PCM cylinder control volume gets i = 6. The solid lines at the border of control volume get integer indices 2,3,4 and 5, while the dotted lines at center have index of 1.5, 2.5 ….. 5.5. The thermal conductance between the refrigerant tube and first control volume is evaluated by Equation 6.

$$G_i = \frac{k\left(2\pi r_i L\right)}{\Delta r} \qquad i = 2,3,4,5 \tag{6}$$

3.1.2 Heat Storage

The PCMConductor is a lumped control volume for heat storage. It is analogous to `Modelica.Thermal.HeatTransfer.Components.HeatCapacitor` but for PCM application. Equations 1 to 4 are written in this component. The melt fraction (λ) is calculated using Equation 7.

$$\lambda = \max\left(0, \min\left(1, \frac{h}{H}\right)\right) \tag{7}$$

The portion of the code performing this task is shown below.

```
if noEvent(T_star <= deltaT) then
  s = deltaT;
else
  s = c_s/c_l*deltaT + H/c_l;
end if;

T_star = T-T_m;

 if noEvent(T_star < -deltaT) then
   c = c_s;
 elseif noEvent(T_star <= deltaT) then
   c = (c_s + c_l)/2 + H/(2*deltaT);
 else
   c = c_l;
 end if;

h = c*(T-T_m+s);

lambda = max(0,min(1,h/H));

mass*der(h) = port.Q_flow;
```

3.1.3 Heat Losses

To model heat losses from PCM to the surroundings a PCM container is modeled. The container is modeled using a component similar to `Modelica.Fluid.Examples.HeatExchanger.BaseClasses.WallConstProps`. The heat storage is calculated using mass for the 1/8th pie. Thermal resistance for each of the pie is in parallel to each other. In this case, the total thermal resistance for each of the pie will be 8 times the thermal resistance of the entire cylindrical container.

Heat losses by both natural convection and radiation are calculated using `Modelica.Thermal.HeatTransfer.Components.Convection` and `Modelica.Thermal.HeatTransfer.Components.BodyRadiation`. The airside heat transfer coefficient for natural convection is calculated assuming the cylinder outer surface is a flat vertical plate where Churchill and Chu (1975) correlation is applicable. The heat transfer coefficient is calculated separately using average values of fluid properties and a fixed value is provided.

3.2 Splitter and Mixer

These components are useful for modeling symmetric circuits. The splitter component splits the refrigerant entering into equal portions, while the mixer merges it back to the complete value. Heat transfer is calculated at only one of the symmetric portions and then scaled back to the total number of symmetric portions. The component is assumed to be both isenthalpic as well as isobaric. The implementation of this component is shown below. The connections are made using `FlowPort` connector from `Modelica Standard Library`, with port A for inlet and port B for outlet.

```
if split then
  port_a.m_flow + nflow*port_b.m_flow = 0;
else
  port_b.m_flow + nflow*port_a.m_flow = 0;
end if;
  port_a.p = port_b.p;

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

3.3 Thermal Expansion Valve

The thermostatic expansion valve (TXV) is comprised of two sections: the throttling section, which regulates the refrigerant mass flow through the valve, and the sensor bulb section, which monitors the refrigerant temperature leaving the evaporator and converts the change in temperature into the change in pressure on the diaphragm, causing the needle to move upward or downward. Since the superheat is sensed by the bulb attached on the suction line, there is a delay between the sensed superheat and the actual superheat. This delay occurs due to the thermal inertia of the bulb and the heat transfer resistance between the substance in the bulb and the refrigerant flowing in the suction line. The sensor bulb is modeled as a lumped section in the present analysis, and its temperature variation with time is given by

$$m_b c_{p,b} \frac{dT_b}{dt} = \frac{T_{amb} - T_b}{R_{ab}} + \frac{T_w - T_b}{R_{wb}} \qquad (8)$$

where M_b is the mass of the sensor bulb, $c_{p,b}$ is the specific heat, T_b is the temperature of the bulb, T_w is the temperature of the tube wall to which the sensor bulb is attached, R_{ab} is the thermal resistance between the ambient and the bulb, and R_{wb} is the thermal contact resistance between the tube wall and bulb. Further details of the TXV model can be found in Qiao et al. (2012)

3.4 Compressor

The compressor is often treated as a quasi-steady-state component in transient simulations because the time scales associated with the variation of the compressor mass flow rate are very small compared to those associated with heat exchanger. The compressor is modeled by using three efficiencies: isentropic efficiency (η_{ise}), volumetric efficiency (η_{vol}), and motor efficiency (η_{motor}). Equations 9-11 describe the model.

$$h_{out} = h_{in} + \frac{h_{out,s} - h_{in}}{\eta_{ise}} \qquad (9)$$

$$\dot{m} = \eta_{vol} \rho_{in} V_d \frac{RPM}{60} \qquad (10)$$

$$\frac{\dot{m}\left(h_{out} - h_{in}\right)}{\eta_{motor}} = \dot{W} \qquad (11)$$

Here ρ_{in} is the density of entering refrigerant vapor, V_d is the displacement volume, $\dot{W}$ is the total work done by the compressor, h_{out} is the refrigerant outlet enthalpy, h_{in} is the refrigerant inlet enthalpy, and $h_{out,s}$ is the refrigerant outlet enthalpy for isentropic compression.

3.5 Condenser

The condenser is a fin and tube air cooled heat exchanger component, and is modeled using a segment-by-segment heat exchanger model (See Figure 7). Details of this component are presented in Qiao (2014) and only the assumptions and capabilities are discussed here.

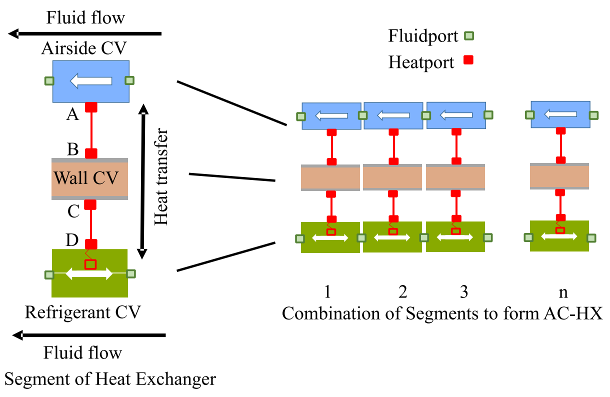


Figure 7. Schematic for Air Cooled Heat Exchanger.

The model consists of three control volumes: the refrigerant (green), finned walls (orange), and air stream (blue). Each of these is linked via `HeatPorts` and `FluidPorts`, from the `Modelica Standard Library`.

Tube walls and associated fins are modeled using a lumped capacitance method. In general, there is a temperature distribution on the fins. By applying the fin efficiency, however, one can lump the tube walls and fins together using one temperature. The axial conduction along the tube is neglected. T_{tube} is the temperature at a node halfway along the thickness (Δd) of the refrigerant tube. Energy balance for this control volume leads to Equation 12.

$$\left(m_{tube}c_{tube} + m_{fin}c_{fin}\right)\frac{dT_{tube}}{dt} = \dot{Q}_r + \dot{Q}_a \qquad (12)$$

The locations of `HeatPorts` A, B, C, and D are shown in Figure 7. `HeatPorts` B and C are used to evaluate the heat transfer within the refrigerant tubing control volume, while `HeatPorts` A and D are used for the airside control volume and refrigerant control volume, respectively. The thickness of the refrigerant tube is much smaller than its diameter, enabling evaluation of wall conduction by Equations 13 and 14 for the airside (Q_a) and the refrigerant side (Q_r).

$$\dot{Q}_r = \frac{kA}{\left(\Delta d/2\right)}\left(T_C - T_{tube}\right) \qquad (13)$$

$$\dot{Q}_a = \frac{kA}{\left(\Delta d/2\right)}\left(T_{tube} - T_B\right) \qquad (14)$$

The airside and refrigerant side cooling capacities are evaluated in their respective control volumes. The outlet state of the air temperature is evaluated using ε-NTU approach. The refrigerant side contains transient conservation equations, which are evaluated with pressure and enthalpy as state variables. Homogenous void fraction model is used for the current investigation. However, slip-flow based void fraction models are also available for analysis involving refrigerant charge.

4 RESULTS AND DISCUSSION

The compressor used for RoCo is of variable speed type. Increasing the compressor speed will lead to faster recharge but increased power consumption. Parametric study is conducted on compressor RPM of 2100, 2600, 3100 and 3600 for the system operation.

All the properties of PCM required for the model are not available experimentally. As a result, properties of a PCM with similar graphite concentration from an earlier study (Dhumane *et al.*, 2018) is used. These properties are given in Table 1.

Table 1. Properties of PCM used for modeling.

Parameter	*Value*
Bulk Density [kg m^{-3}]	143
Volume Fraction [%]	6.3
Solid Density [kg m^{-3}]	1005
Liquid Density [kg m^{-3}]	930
Specific Heat Capacity (Solid) [J kg^{-1} K^{-1}]	1997
Specific Heat Capacity (Liquid) [J kg^{-1} K^{-1}]	2335
Latent Heat [kJ kg^{-1}]	178
Thermal Conductivity [W m^{-1} K^{-1}]	20.2

The refrigerant used for the current system is R134a. The values of compressor efficiency are taken from the validated study of cooling mode (Dhumane *et al.*, 2019). Nominal values are provided for calculating heat transfer and pressure drop in heat exchangers. These values are calculated using correlations available from literature. Summary of the correlations used for different phenomena is given in Table 2.

Table 2. Correlation Summary.

Phenomena	Heat Transfer	Pressure Drop
Refrigerant boiling	(Shah, 1982)	(Müller-Steinhagen and Heck, 1986)
Refrigerant condensation	(Shah, 2016)	(Müller-Steinhagen and Heck, 1986)
Refrigerant single phase	(Dittus and Boelter, 1985)	(Blasius, 1913)
Condenser airside	(Wang and Chi, 2000)	Neglected
Refrigerant tube heat losses to ambient by natural convection	(Churchill and Chu, 1975)	Neglected

The model is simulated using Dymola 2018 with Radau-IIa solver, tolerance of 1e-6. Equidistant time steps option is unchecked from solver settings. The computer used has Intel Xeon Processor with 3.5 GHz speed, 16 GB RAM, 64-bit Windows Operating System and x64 based Processor. The simulation speed for the system with 3100 RPM is 17.2 seconds. Runtime of other cases is comparable.

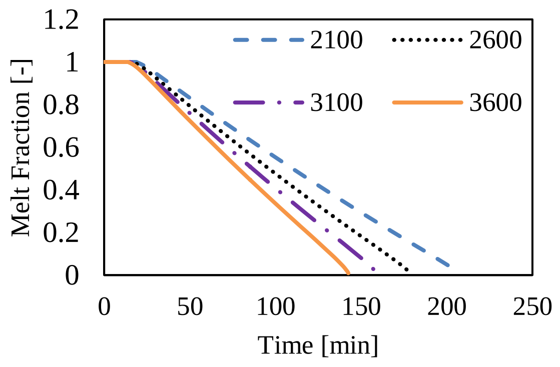


Figure 8. Rate of Solidification at different RPM.

The melt fraction heat pump operation at four different compressor RPM is shown in Figure 8, while the power consumption from compressor is shown in Figure 9. It can be observed that higher RPM leads to faster recharge, but the power consumption by the compressor to deliver the high RPM is large. Higher compressor speed leads to larger refrigerant mass flow rate and larger rate of heat release from the condenser during the heat pump operation. Since the time duration of recharge is different, it is necessary to compare integrated power consumption during the recharge time duration.

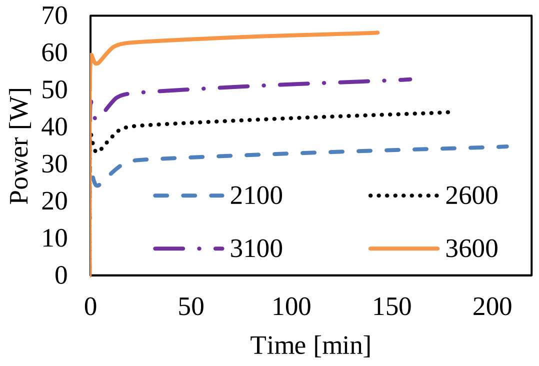


Figure 9. Compressor Power Consumption at different RPM.

A fan is also operational during the entire recharge cycle and so a 7-W power usage needs to be added to the power consumed by the compressor. This is shown in Figure 10, with a quadratic fitting trend line. The total recharge time for different heat pump cycles is plotted in Figure 11. A quadratic trend line is drawn to interpolate recharge time for RPM between simulated cases.

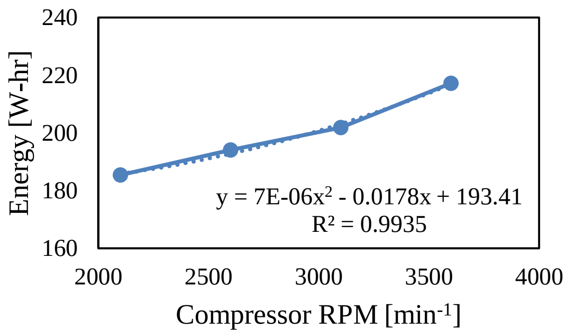


Figure 10. Integrated power consumption by heat pump operation at different compressor RPM.

From experiment data (Qiao *et al.*, 2018), the COP for just the cooling operation is obtained to be 4.25, with 754.0 Wh of cooling delivered at energy consumption of 177.4 Wh. For a complete cycle COP, the energy consumption of recharge cycle also needs to be added to the energy consumption during cooling operation. This is done using Equation 15, where the numerator is the total cooling capacity during cooling operation and denominator is the sum of power consumption in cooling operation and heat pump operation. Subscript ‘c’ stands for cooling operation, ‘r’ for heat pump operation.

$$COP_{cyc} = \frac{\int_0^{t_c} \dot{Q}_c dt}{\int_0^{t_c} \dot{W}_c dt + \int_0^{t_r} \dot{W}_r dt} \tag{15}$$

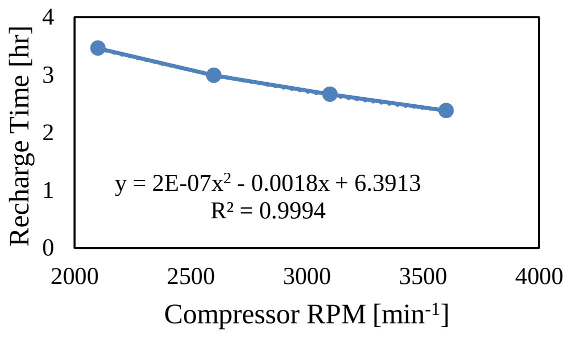


Figure 11. Recharge Time with Heat Pump Operation.

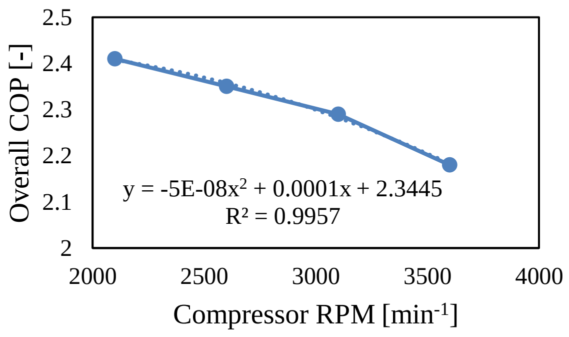


Figure 12. System COP with Heat Pump Operation.

Figure 10 illustrates the relationship between compressor RPM and RoCo charge time. As the compressor RPM increases from 2100 to 3600, the vapor compression cycle capacity increases which leads to a one-hour reduction in recharge time. By examining the fitted equation, one can conclude that the relationship is not linear. This can be explained by the fact that although the refrigerant mass flow rate increases linearly with RPM, the increase of capacity is still restricted by the air sink. The higher RPM makes the heat exchangers undersized, and therefore, the slope of the curve in Figure 11 becomes smaller as the RPM increases.

Figure 12 demonstrates the relationship between compressor RPM and system COP. The COP tells the system efficiency under various compressor RPM cases. As previously explained, the higher RPM makes the heat exchangers undersized and therefore system less efficient. The reduction in COP worsens as the compressor rotates faster which can be spotted by the increased slope of the curve.

The recharge period for the case with 3600 RPM is 55% lesser than that from thermosiphon, which is a significant improvement. Increase in RPM from 2100 to 3600 results in increased energy consumption of only 30 Wh. So, running the recharge at high RPM is definitely beneficial. The increase in energy consumption of RoCo will be in the order of a few hundred Wh, which may be easily offset from the savings from temperature set point elevation of building where energy consumption is in the order of thousand Wh. The model will be validated with experiment data in future to see how closely it predicts the heat pump cycle.

5 CONCLUSIONS

A physically based model of the graphite enhanced PCM is developed and then used in a system simulation for a reversible heat pump based recharge operation to investigate its potential benefits. The compressor RPM is varied to understand the heat pump operation at different operation conditions. Empirical correlations are generated to enable evaluation of heat pump performance based on simulation. The recharge period from heat pump operation is observed to be 55% lesser than that from thermosiphon. Heat pump recharge shows promise and needs experimental investigation.

Acknowledgements

This research was supported by the Advanced Research Projects Agency - Energy (ARPA-E) under Award DE-AR0000530 and Center for Environmental Energy Engineering (CEEE). The authors acknowledge the support of the team members of the Roving Comforter Project.

References

Blasius, H. (1913) 'Das aehnlichkeitsgesetz bei reibungsvorgängen in flüssigkeiten', in *Mitteilungen über Forschungsarbeiten auf dem Gebiete des Ingenieurwesens*. Springer, pp. 1–41.

Cao, Y. and Faghri, A. (1991) 'Performance characteristics of a thermal energy storage module: a transient PCM / forced convection conjugate analysis', *International Journal of Heat and Mass Transfer*, 34, pp. 93–101.

Churchill, S. W. and Chu, H. H. S. (1975) 'Correlating equations for laminar and turbulent free convection from a vertical plate', *International Journal of Heat and Mass Transfer*. Elsevier, 18(11), pp. 1323–1329.

Dermont, P., Limperich, D., Windahl, J., Prolss, K., Kubler, C., (2016) 'Advances of Zero Flow Simulation of Air Conditioning Systems using Modelica', in *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*. Linköping University Electronic Press, pp. 139–144.

Dhumane, R., Ling, J., Aute, V., Radermacher, R., (2017) 'Portable personal conditioning systems: Transient modeling and system analysis', *Applied Energy*. Elsevier, 208, pp. 390–401. doi: 10.1016/j.apenergy.2017.10.023.

Dhumane, R., Mallow, A., Qiao, Y., Gluesenkamp, K.R., Graham, S., Ling, J., Radermacher, R., (2018) 'Enhancing the thermosiphon-driven discharge of a latent heat thermal storage system used in a personal cooling device', *International Journal of Refrigeration*. Elsevier, 88, pp. 599–613.

Dhumane, R., Qiao, Y., Ling, J., Muehlbauer, J., Aute, V., Hwang, Y., Radermacher, R., (2019) 'Improving System Performance of a Personal Conditioning System integrated with Thermal Storage', *Applied Thermal Engineering*. Elsevier, 147(25 January 2019), pp. 40–51.

Dittus, F. W. and Boelter, L. M. K. (1985) 'Heat transfer in automobile radiators of the tubular type', *International Communications in Heat and Mass Transfer*, 12(1), pp. 3–22.

Franke, R., Casella, R., Otter, M., Sielemann, M., Elmqvist, H., Mattson, S.E., Olsson, H., (2009) 'Stream connectors-an extension of Modelica for device-oriented modeling of convective transport phenomena', in *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*. Linköping University Electronic Press, pp. 108–121.

Heidarinejad, M., Dalgo, D.A., Mattise, N.W., Srebric, J., (2018) 'Personalized cooling as an energy efficiency technology for city energy footprint reduction', *Journal of Cleaner Production*. Elsevier, 171, pp. 491–505.

Hoyt, T., Arens, E. and Zhang, H. (2015) 'Extending air temperature setpoints: Simulated energy savings and design considerations for new and retrofit buildings', *Building and Environment*, 88, pp. 89–96. doi: 10.1016/j.buildenv.2014.09.010.

Kim, J., Schiavon, S. and Brager, G. (2018) 'Personal comfort models–A new paradigm in thermal comfort for occupant-centric environmental control', *Building and Environment*. Elsevier, 132, pp. 114–124.

Müller-Steinhagen, H. and Heck, K. (1986) 'A simple friction pressure drop correlation for two-phase flow in pipes', *Chemical Engineering and Processing: Process Intensification*. Elsevier, 20(6), pp. 297–308.

Qiao, H., Xu, X., Aute, V., and Radermacher, R. (2012) Modelica based transient modeling of a flash tank vapor injection system and experimental validation. the *14th International Refrigeration and Air Conditioning Conferenc*. West Lafayette, IN.

Qiao, H. (2014) *Transient Modeling of Two Stage and Variable Refrigerant Flow Vapor Compression Systems with Frosting and Defrosting*. Ph.D. Dissertation, University of Maryland, College Park

Qiao, Y., Mallow, A., Muehlbauer, J., Hwang, Y., Ling, J., Aute, V., Radermacher, R., Gluesenkamp, K.R., (2018) 'Experimental Study on Portable Air-Conditioning System with Enhanced PCM Condenser', in *17th International Refrigeration and Air Conditioning Conferenc*. West Lafayette, IN.

Shah, M. M. (1982) ‘Chart correlation for saturated boiling heat transfer: equations and further study’, *ASHRAE Trans.;(United States)*, 88(CONF-820112-).

Shah, M. M. (2016) ‘Comprehensive correlations for heat transfer during condensation in conventional and mini/micro channels in all orientations’, *International journal of refrigeration*, 67, pp. 22–41. doi: 10.1016/j.ijrefrig.2016.03.014.

Wang, C.-C. and Chi, K.-Y. (2000) ‘Heat transfer and friction characteristics of plain fin-and-tube heat exchangers, part I: new experimental data’, *International Journal of heat and mass transfer*, 43(15), pp. 2681–2691. doi: 10.1016/s0017-9310(99)00332-4.

Zhang, H., Arens, E. and Zhai, Y. (2015) ‘A review of the corrective power of personal comfort systems in non-neutral ambient environments’, *Building and Environment*, 91, pp. 15–41. doi: 10.1016/j.buildenv.2015.03.013.

Real-time optimization of intermediate temperature for a cascade heat pump via extreme seeking

Wenyi Wang[1] Yaoyu Li[1]

[1]Department of Mechanical Engineering, University of Texas at Dallas, Richardson, TX 75080, U.S.A.
{Wenyi.Wang1, yaoyu.li}@utdallas.edu

Abstract

Improving the energy efficiency of air-source heat pump (ASHP) has been a critical issue for heating operation in cold climates. The cascade heat pump system has been developed as a more advantageous solution over the single-stage heat pump. However, the increased complexity of cascade heat pump systems has presented great challenge for online optimization for the energy efficiency, as model based control/optimization methods incur costly modeling and calibration under different operation and equipment conditions. We propose to use the extremum seeking control as a model-free real-time optimization strategy for efficient operation of cascaded heat pump. The intermediate temperature setpoint is used as the manipulated input for maximizing the system efficiency while satisfying the heating load demand. A Modelica model of an R134a/410a cascade heat pump is developed, and control simulations are conducted for validating the system performance under different ambient conditions.

Keywords: Cascade heat pump, intermediate temperature, extremum seeking control, real-time optimization, Modelica.

1 Introduction

Heat pump has been a mature technology for providing the space or water heating in building operation and industrial processes. However, the performance of the single-stage heat pump can be significantly limited under cold climate operation, which is manifested by decrease in heating capacity and coefficient of performance (COP) as well as increase in discharge temperature. To deal with such challenge, various techniques have been developed to improve the performance of heat pump by enabling higher temperate lift and wider range of ambient temperature, e.g. vapor injections and cascade configuration (Bertsch and Groll 2008; Park et al. 2015; Arpagaus et al. 2016).

In particular, the cascade heat pump can be a viable solution to the limitation of single-stage heat pump (Bansal and Jain 2007). Compared to the single–stage refrigeration cycle, the cascade cycle has a smaller compression ratio for each cycle and exhibits a better compression efficiency (Dopazo et al. 2009; Wang et al. 2009). Then, for the operation of the heat pump system thus implemented, a significantly higher pressure ratio can be achieve even under a low ambient temperature, i.e. resulting in large gap between the condensing pressure and the evaporating pressure.

There have been intensive efforts on optimizing the design and operation of cascade system (Jung et al. 2013; Park et al. 2013; Jung et al. 2014). Kim et al. (2014) conduct experimental and numerical studies on an air-to-water cascade heat pump with R134a/R410A, aiming to find the optimal charge amount. Chae et al. (2015) evaluate the impact of high-temperature cycle refrigerant charge on the performance of a cascade heat pump. Kim et al. (2014) experimentally study the effect of water temperature lift (i.e. the increase of condenser outlet water temperature from the inlet) on the performance of a cascade heat pump with R134a/R410A as the refrigerant. Ma et al. (2018) propose a study for a high-temperature cascade heat pump, using a near-zeotropic mixture BY-3 and R245fa as the working fluids in the low-temperature and high-temperature cycles, respectively. Experimental results show that the cascade heat pump system could reach water outlet temperature of 142°C and the maximum lift of water temperature could reach 100°C. The pressure ratio in the high and low-temperature cycle was 3.4 and 3.9, respectively, with the system COP of 1.72.

Among all the operational parameters for the cascade system, the intermediate temperature, which is the evaporating temperature of a high temperature cycle or the condensing temperature of a low temperature cycle, is deemed the most important. It has direct impact on the compression ratio and compressor isentropic efficiency of each cycle, and thus the COP of the whole system. Therefore, optimizing the intermediate temperature has been a primary focus for the cascade refrigeration or heat pump systems (Lee et al. 2006; Wang et al. 2009; Bhattacharyya 2008; Dopazo et al. 2009). Park et al. (2013) develop a thermodynamic model of a cascade heat pump water heater with R134a and R410A in order to obtain the optimal intermediate temperature. Their numerical analysis was later verified experimentally by Kim et al. (2013).

The existing work on experimental and numerical research has resulted a number of correlations for the optimal intermediate temperature, as for various system configurations. However, such models and the optimal intermediate temperature thus determined are based on elaborate calibration for specific equipment, ambient and load conditions. For practical operation, due to the diverse combinations of operating conditions and equipment status, the model calibration efforts involved can be cost prohibitive. Therefore, real-time control or optimization strategies with least dependency of model knowledge will be highly beneficial.

In this paper, we propose to use the Extremum Seeking Control (ESC) to optimize the setpoint of intermediate temperature (Li et al. 2010; Mu et al. 2015; Hu et al. 2015). More specifically, the low temperature cycle (LTC) condensing temperature is taken as the manipulated input of ESC, while the total power consumption of the system is the only feedback. As for the inner loop control, the LTC compressor speed is used to regulate the intermediate temperature, and the water outlet temperature is regulated by the high temperature cycle (HTC) compressor speed.

The remainder of this paper is organized as follows. The Modelica based dynamic simulation model of the cascade heat pump system is described in next section. Section 3 reviews the ESC principle and design guidelines. Simulation results are presented in Section 4. Section 5 concludes the paper with future work discussed.

2 Dynamic modeling of a cascade heat pump water heater

A cascade heat pump consists of two independently operated single-stage cycles as shown in Fig. 1. The low temperature cycle (LTC) absorbs the heat through evaporator from the ambient air and then transfers the heat to the high temperature cycle (HTC) through an intermediate heat exchanger which is called the cascade heat exchanger. Then the refrigerant in the HTC evaporates in the cascade heat exchanger and then is compressed before entering the condenser where the refrigerant rejects the heat to the water. The cascade heat exchanger works as a condenser for the LTC and evaporator for the HTC. Compared to a single-stage system, the cascade system has a smaller compression ratio and higher compression efficiency for each stage of compression. In this paper, we adopt the typical combination of R410a-R134a for both cycles. Among the most HFC refrigerants, R134a shows a higher critical temperature that is beneficial for making the high temperature hot water (Bertsch & Groll 2008).

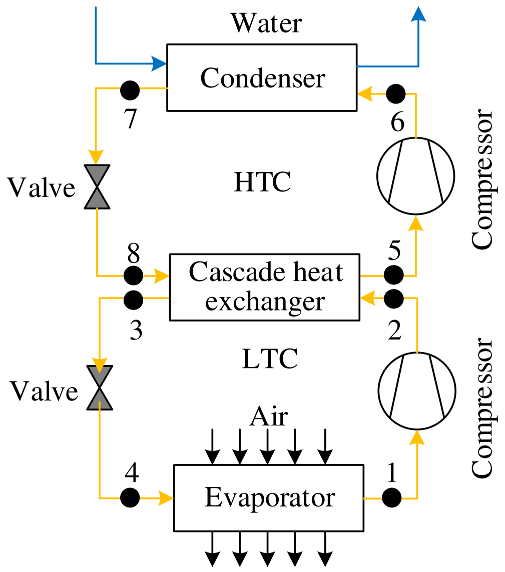


Figure 1. Schematic of a cascade heat pump water heater

Based on these analysis, a Modelica based dynamic simulation model of cascade heat pump is developed using Dymola (Dassault Systems 2017) and TIL Library (TLK-Thermo 2017), as shown in Fig. 2.

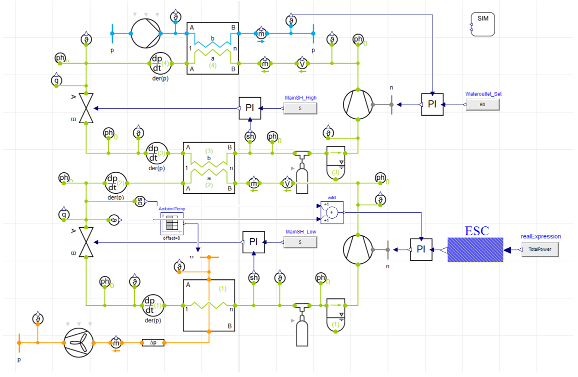


Figure 2. Dymola layout for a cascade heat pump.

In this model, the cascade heat pump system mainly consists of two compressors, three heat exchangers and two-expansion valves. The two compressors for the low and high temperature cycle are both modelled by the scroll compressor module in the TIL Library *TIL.VLEFluidComponents.Compressors.ScrollCompressor* with different displacements. The evaporator for the LTC is modeled with the fin-and-tube cross-flow heat exchanger module *TIL.HeatExchangers.FinAndTube.MoistAirVLEFluid.CrossFlowHX*. The air-side and tube-side convective heat transfer coefficient h_{air} and h_{tube} is calculated by the following equations:

$$Nu_{air} = 0.31\,\mathrm{Pr}^{0.333}\,\mathrm{Re}^{0.625}\left(\frac{4V_{fin}R_{void}}{D_{tube}A_{fin}}\right)^{0.333} \quad (1)$$

$$h_{tube} = \left(\frac{1.8}{N^{0.8}}\right)\left(\frac{\lambda_{liquid}}{h_{hyd}} * 0.023\,\mathrm{Re}^{0.8}\,\mathrm{Pr}^{0.4}\right) \quad (2)$$

The cascade heat exchanger separates the low temperature refrigerant and high temperature refrigerant, and realizes heat transfer between them. The temperature difference in a cascade heat exchanger is generally considered the most critical parameter that affects the system performance. A large temperature difference in a cascade heat exchanger can lead to degradation of system performance due to the higher irreversibility. To reduce the temperature difference for the cascade heat exchanger, a plate heat exchanger is adopted, modeled by the *TIL.HeatExchangers. Plate.VLEFluidVLEFluid.ParallelFlowHX* module. In the cascade heat exchanger, the low temperature refrigerant evaporates and the high temperature refrigerant condenses. The specific correlation for the evaporating side and condensing side is as follows:

$$h = 0.023\,\mathrm{Re}^{0.5}\,\mathrm{Pr}^{0.4}\left[(1-x)^{0.8} + \frac{3.8x^{0.76}(1-x)0.04}{(p/p_{crit})^{0.38}}\right] \quad (3)$$

$$h = 0.023\,\mathrm{Re}^{0.5}\,\mathrm{Pr}^{0.4}\left\{\begin{aligned}&(1-x)^{0.01}((1-x)^{1.5}+1.9x^{0.6}(\frac{\rho_{liquid}}{\rho_{gas}})^{0.35})^{-2.2}\\&+x^{0.01}\left[\left(\frac{\alpha_{gas}}{\alpha_{liquid}}\right)(1+8(1-x)^{0.7}(\frac{\rho_{liquid}}{\rho_{gas}})^{0.67})\right]^{-2}\end{aligned}\right\}^{-0.5} \quad (4)$$

The condenser of the HTC is also modeled by the same plate heat exchanger model. The correlation of refrigerant side is the same as Eq. (4), while the correlation of the waterside follows

$$Nu = 0.122\,\mathrm{Pr}*\frac{1}{3}(\frac{\eta_{fluid}}{\eta_{wall}})^{\frac{1}{6}}(\xi*\mathrm{Re}^2\sin(2\varphi))^{0.374} \quad (5)$$

The orifice valve module *TIL.VLEFluidComponents. Valves.OrificeValve* is used to model the expansion valve, which can calculates the mass flow rate in dependency of the pressure drop using the equation of Bernoulli as follow:

$$m_{flow} = A_{eff} * \sqrt{(p_{input} - p_{output}) * 2\rho_{input}} \quad (6)$$

The evaporator fan is modeled with a simple fan module *TIL.GasComponents.Fans.SimpleFan*, whose operation can be defined with pressure differential or mass flow rate. The water pump is modeled with the *TIL.Liquid Components.Pumps.SimplePump* module, which is an affinity-law based pump model. In this study, the mass flow rate for the air side of the evaporator and the water side of the condenser are both set as constant.

In this study, the two scroll compressor are both controlled by the PI controller, the evaporator fan speed is fixed, so the fan power is constant. The total power is:

$$P_{total} = P_{ucomp} + P_{lcomp} + P_{fan} \quad (7)$$

Then the COP of the cascade water heat pump is the ratio of the heat capacity to the total power:

$$COP = \frac{Q}{P_{total}} \quad (8)$$

The system model is developed based on the ZX-DKFXRS-10II heat pump water heater manufactured by Zhengxu New Energy Equipment Technology Co., Ltd in China. The heat capacity is 58.5 kW under the nominal condition which is defined as: ambient temperature of −12°C dry bulb and −14°C wet bulb, and the water outlet temperature of 85°C. The nominal power consumption is 32.5 kW, the rated volumetric flow rate for the evaporator fan is 30000 m^3/h, and the refrigerant charges for the LTC and HTC are 17 kg and 19 kg respectively. The steady-state characteristics of the simulation model have been validated with the lab testing by the manufacturer, however, the validation data cannot be disclosed by the time of paper preparation.

In this model, the minimal total power corresponds to the maximum COP because of the constant heat capacity maintained throughout the simulated operation. The total power is adopted as the only ESC feedback rather than the COP, because power measurement is relatively simple and cost-effective while determination of COP requires several measurements of thermal and fluidic variables.

3 Extremum seeking control design

ESC deals with the online optimization problem of finding an optimizing input $u_{opt}(t)$ for the generally unknown and/or time-varying cost function $l(t, u)$ (Krstic & Wang 2000; Rotea 2000). The standard gradient based ESC is illustrated in Fig. 3, in which a pair of dither-demodulation signals are used along with high-pass and low-pass filters to extract the gradient information. Closing the loop with integral controller can drive the input to optimality provided that the closed-loop stability is achieved. A typical design guideline for ESC follows Li et al. (2010):

(1) Perform open-loop step test to estimate the input dynamic for the input channel.

(2) The dither frequency should be chosen well within the bandwidth of the input dynamics.

(3) The dither amplitude should be chosen such that the dithered output has sufficient signal-to-noise ratio at the dither frequency.

(4) The dither frequency is generally located in the

pass band of the high-pass filter and in the stopband of the low-pass filter.

(5) The dither-demodulation phase difference integral gain should be chosen to guarantee the asymptotic stability based on some estimate of the input/output dynamics and the Hessian of the static map near the equilibrium.

(6) The integral gain needs to be adjusted to achieve the desirable transient performance under the selected dither signal.

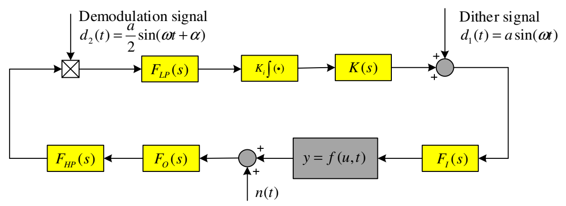


Figure 3. Block diagram of extremum seeking control.

In this study, the ESC controller is used to find the optimal intermediate temperature that can minimize the total power consumption, which is equivalent to maximizing the system COP under fixed heating capacity operation. For the HTC, the water outlet temperature is regulated by the HTC compressor capacity with a proportional-integral (PI) controller, while the superheat of the HTC side of cascade heat exchanger is regulated by the electronic expansion valve (EEV) opening with another PI controller. For the LTC, the evaporator superheat is also regulated by a dedicated EEV, and the intermediate temperature as defined above is regulated by the LTC compressor.

For a cascade heat pump operating in a fixed condition, that means the fixed water inlet and outlet temperature of the condenser and fixed ambient air condition (temperature and relative humidity). A higher low temperature cycle condensing temperature will result in the higher compressor ratio of low temperature cycle, meanwhile it will lead to a lower high temperature cycle compressor ratio because of the fixed heat capacity. The higher compressor ratio means higher power consumption. In the contrary, the lower R410A condenser temperature will results in lower compressor pressure ratio of low temperature cycle and higher-pressure ratio of high temperature cycle. So there exists an optimal R410A condenser temperature which can make a proper pressure ratio for each cycle, and therefore the optimal isentropic efficiency of compressor can be obtained for the each cycle. So in this condition, the total power consumption will be the minimal value. So in a realistic variable ambient air condition, the optimal intermediate temperature changes in real time.

For applying the proposed control strategy in the cascade heat pump to obtain the optimal intermediate temperature therefore the maximal COP and minimal total power, we need to estimate the system input dynamics first in a fixed condition. The hot water inlet temperature is set to be 55°C, and the outlet temperature is regulated to 60°C by the high temperature cycle compressor via a PI controller. The hot water mass flow rater is set to be 0.72 kg/s so that the total heat capacity is the system is fixed at around 15 kW. The ambient air temperature and relative humidity are set to be -7°C and 60% respectively. The air mass flow rate is set to be a constant at 1.85 kg/s. The main loop superheat is fixed at 5°C via a PI controller. For estimating the system dynamics, the simulation is testing under the above condition and the intermediate temperature is regulating from 10°C to 20°C, 20°C to 30°C, 30°C to 40°C, respectively. Based on the simulation results, the normalized response is shown in Fig. 6 and the input dynamics is estimated as

$$\hat{F}_I(s)=\frac{0.021^2}{s^2+2\times0.76\times0.021s+0.021^2} \tag{9}$$

A dither signal with amplitude of 1°C and frequency of 0.005 Hz is then selected. The high-pass and low-pass filters are chosen as

$$F_{HP}(s)=\frac{s^2}{s^2+2\times0.8\times0.0037s+0.0037^2} \tag{10}$$

$$F_{LP}(s)=\frac{0.0032^2}{s^2+2\times0.9\times0.0032s+0.0032^2} \tag{11}$$

4 Simulation Study

In this section, the ESC controller designed in the previous section is evaluated with simulation study using the cascade heat pump system model described earlier.

4.1 Simulation under Fixed Condition

The operation scenario is the same as that described in Section 3. The static map of the total power and COP to the intermediate temperature is shown in Fig. 4, where the intermediate temperature ranges from 10°C to 45°C. The total power achieves the minimal value of 5933.8 Watt at the intermediate temperature of 27.5°C. The COP shows an opposite tendency over the total power profile, achieving the maximum of 2.54 at the same intermediate temperature.

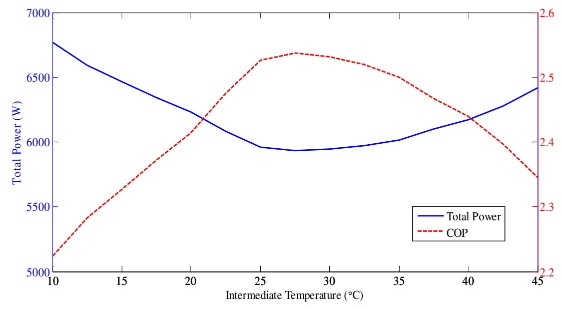


Figure 4. Static map of total power and COP in terms of intermediate temperature for the cascade heat pump.

The simulation results are shown in Fig. 5 and Fig. 6. The initial HTC compressor speed is fixed at 75 Hz, the LTC compressor speed is fixed at 79 Hz, and the initial intermediate temperature is set to be 20°C. The ESC is turned on at 1 hour and converges after about 3000 seconds. Then the optimal intermediate temperature found by ESC is 29.4°C. Compared to the optimum from the static map in Fig. 3, the steady-state error is about 1.6%. The LTC compressor speed is settled at 90 Hz with the extremum seeking process, and the HTC compressor speed is adjusted to 59 Hz. The total power decreases from 6213.05 W to 5933.77 W, and the COP increases from 2.42 to 2.54, which implies a 4.9% power saving or COP enhancement. It is noteworthy that the water outlet temperature is well stabilized at 60°C with variation less than 0.1°C. The superheats for HTC and LTC are both stabilized at 5°C. This means that the proposed ESC strategy can indeed optimize the system performance by searching for the optimal intermediate temperature, which will make the HTC and LTC operating at their optimal compression ratio. This is achieved by changing the two compressor speeds. The requirements for load demand satisfaction and superheat regulation are all met.

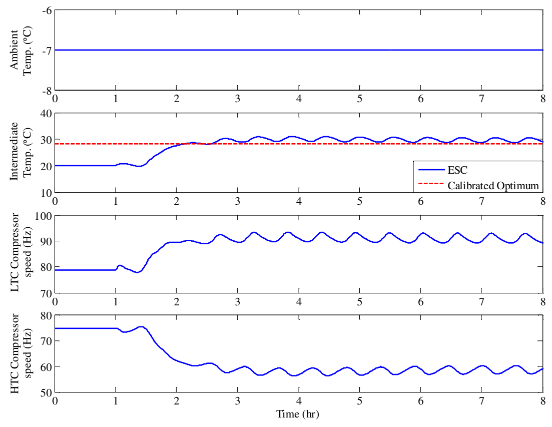


Figure 5. ESC simulation results for the cascade heat pump under fixed ambient temperature.

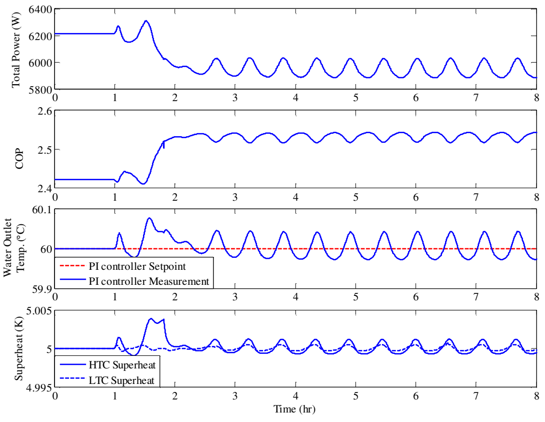


Figure 6. Performance and inner loop regulation for ESC simulation of the cascade heat pump under fixed ambient temperature.

As a further evaluation, the two compressor speeds are set to start at a different combination of initial values. As shown in Fig. 7, the LHC compressor starts at 105 Hz, and for maintaining the heat capacity requirement, the HTC compressor is decreased to 42 Hz. The intermediate temperature is then 42°C. The ESC is also turned on at 1 hour, and converges after about 2000 second. The intermediate temperature found by ESC is 29.6°C, which is consistent with the result from the above case. The HTC and LTC compressor speeds are 90 Hz and 58 Hz, respectively. Similarly, the total power and COP are all as same as the previous result. The performance and inner loop regulation are plotted in Fig. 8.

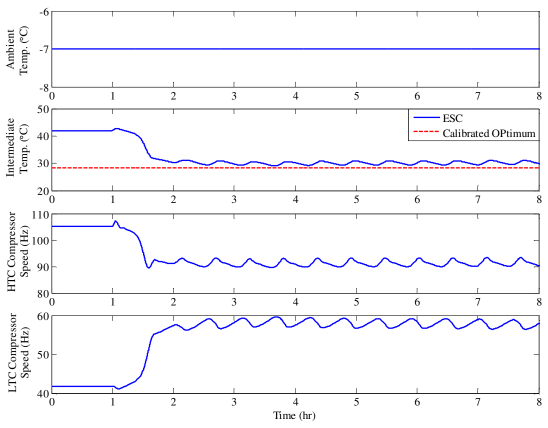


Figure 7. ESC simulation results for the cascade heat pump for second case of fixed ambient temperature.

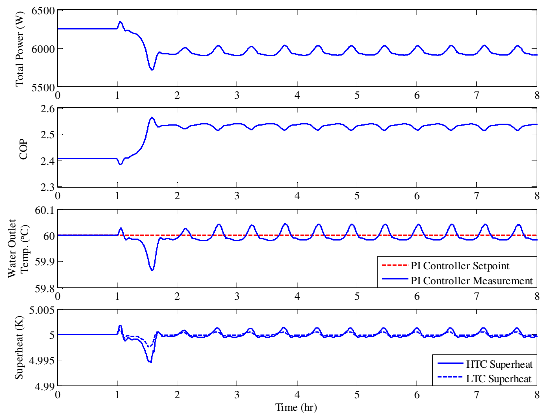


Figure 8. Performance and inner loop regulation for ESC simulation of the cascade heat pump for second case of fixed ambient temperature.

4.2. ESC under variable ambient temperature

The proposed ESC strategy is then evaluated for a staircase ambient temperature profile, as shown in Figs. 9 and 10. The ambient temperature changes from -7°C to -12°C and then -17°C, each change following a 3600-second linear ramp. Fig. 9 shows that the intermediate temperature converges to the respective optimum found the SQP (sequential quadratic programming) procedure offered by the Dymola Optimization Library. The transient time associated with each change of ambient temperature is also reasonable. Fig. 10 shows the energy performance as well as the inner loop regulation, and the results indicate the validity of the ESC search.

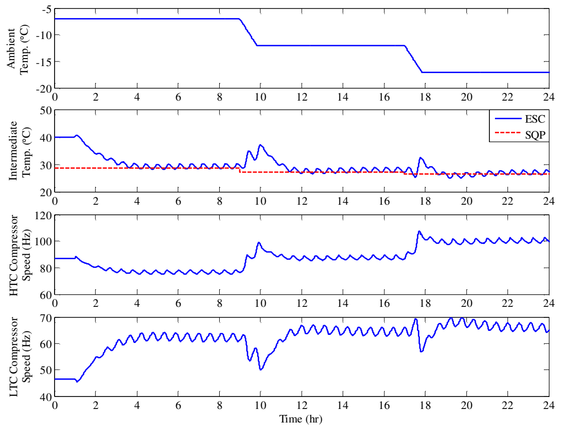


Figure 9. ESC simulation results for the cascade heat pump under staircase profile of ambient temperature.

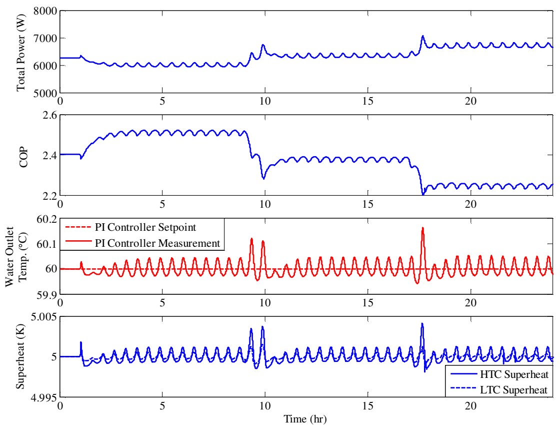


Figure 10. Performance and inner loop regulation for ESC simulation of the cascade heat pump under staircase profile of ambient temperature.

4.3 Simulation under the realistic condition

The ESC strategy is then evaluated with a realistic ambient temperature profile, for which a two-input ESC is designed. Instead of using the intermediate temperature, the compressor speeds for the LTC and HTC are used as the manipulated inputs. From the TMY3 weather data (Buildings.BoundaryConditions. WeatherData.ReaderTMY3), a one-week ambient temperature profile of the O'Hare Airport in Chicago is adopted, which spans January 13 to 20. As shown in the first subplot of Fig. 11, the ambient temperature ranges from −14.4°C to 11.8°C.

As benchmark, the simulation is conducted with the compressor speed fixed at 50 Hz, which is the manufacturer's recommended setting. Then the ESC strategy is simulated. The simulation results for two compressor speeds, intermediate temperature and COP are compared in Fig. 11 for the ESC and benchmark operations. With the ESC strategy, the COP profile is clearly above that under the benchmark operation through the whole-week period being simulated. The largest COP improvement is 15.1% at the maximal ambient temperature of 11.8°C. The smallest COP improvement of 1.1% occurs at the ambient temperature of −12°C, which is exactly the nominal ambient temperature for the equipment design. This is not surprising in that the system operation parameters are optimized for this very condition. The results of heat capacity, total power and water mass flow rate are shown in Fig. 12. The water outlet temperature and superheat are well regulated to their respective setpoints, which justifies the effectiveness of the system operation being simulated.

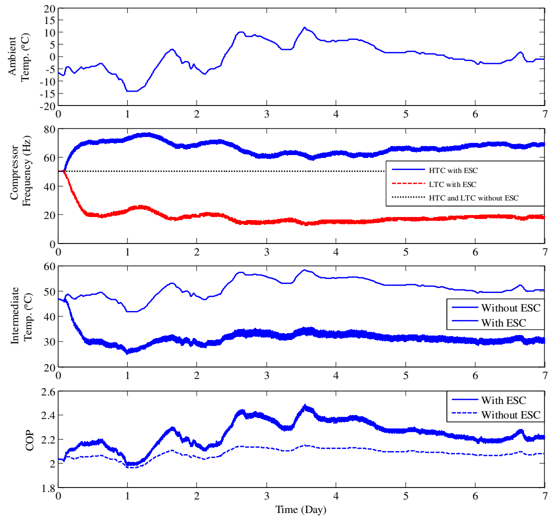


Figure 11 – Trajectories of ambient temperature, compressor speed, intermediate temperature and COP under the realistic condition.

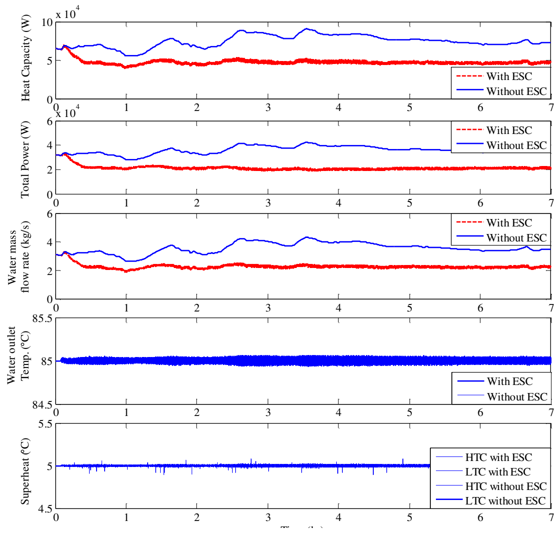


Figure 12. Trajectories of heat capacity, total power, water mass flow rate, water outlet temperature and superheat under the realistic condition.

More simulations are performed under way for different ambient and load conditions.

5 Conclusion

In this paper, we propose an ESC based model free real time optimization method for optimizing the intermediate temperature of cascade heat pump system. A Modelica dynamic simulation model is developed for a cascade heat pump water heating system, using Dymola and TIL Library. Simulations have been performed under fixed, staircase and realistic ambient temperature profiles. Simulation results show that the proposed strategy can converge the operation to pre-calibrated optimum, which promises great benefit for practical operation of the cascade heat pumps systems. Such control strategy requires only power measurement as feedback, which minimizes the sensor requirement. Further work is under way to evaluate the proposed strategy under other operating scenarios.

Acknowledgements

The authors are grateful of TLK-Thermo GmbH for their permission for access to TIL-Suite products, as well as their technical help.

References

C. Arpagaus, F. Bless, J. Schiffmann, and S.S. Bertsch, Multi-temperature heat pumps: A literature review, Int. J. Refrig. 69 (2016) 437-465.

P.K. Bansal, S. Jain, Cascade systems: past, present, and future, ASHRAE Trans. 113 (2007) 245-252.

S. Bhattacharyya, S. Mukhopadhyay, J. Sarkar, CO2-C3H8 cascade refrigeration-heat pump system: heat exchanger inventory optimization and its numerical verification, Int. J. Refrig. 31 (2008) 1207-1213.

S.S. Bertsch, E.A. Groll, Two-stage air-source heat pump for residential heating and cooling applications in northern US climates, Int. J. Refrig. 31 (2008) 1282–1292.

J.H. Chae, J.M. Choi, Evaluation of the impacts of high stage refrigerant charge on cascade heat pump performance, Renewable Energy 79 (2015) 66-71.

Dassault Systems, Dymola: Multi-Engineering Modeling and Simulation based on Modelica and FMI, 2017. < https://www.3ds.com/products-services/catia/products/dymola >.

J.A. Dopazo, J. Fernández-Seara, J. Sieres and F. J. Uhía, Theoretical analysis of a CO2–NH3 cascade refrigeration system for cooling applications at low temperatures, Appl. Therm. Eng. 29 (2009) 1577–1583.

B. Hu, Y. Li, F. Cao, Z. Xing, Extremum seeking control of COP optimization for air-source transcritical CO2 heat pump water heater system, Applied Energy 147 (2015) 361-372.

H.W. Jung, H. Kang, W.J. Yoon, Y. Kim, Performance comparison between a single-stage and a cascade multi-functional heat pump for both air heating and hot water supply, Int. J. Refrig. 36 (2013) 1431–1441.

H.W. Jung, H. Kang, et al., Performance optimization of a cascade multifunctional heat pump in various operation modes, Int. J. Refrig. 42 (2014) 57-68.

D.H. Kim, H.S. Park, M.S. Kim, Optimal temperature between high and low stage cycles for R134a/R410A cascade heat pump based water heater system, Exp. Therm. Fluid Sci. 47 (2013) 172-179.

D.H. Kim, H.S. Park, M.S. Kim, The effect of the refrigerant charge amount on single and cascade cycle heat pump systems, Int. J. Refrig. 40 (2014) 254-268.

D.H. Kim, M.S. Kim, The effect of water temperature lift on the performance of cascade heat pump system, Appl. Therm. Eng. 67 (2014) 273-282.

Krstić, M., Wang, H. H. 2000. "Stability of extremum seeking feedback for general nonlinear dynamic systems." Automatica 36 (4): 595-601.

T. Lee, C. Liu, T. Chen, Thermodynamic analysis of optimal condensing temperature of cascade-condenser in CO2/NH3 cascade refrigeration systems, Int. J. Refrig. 29 (2006) 1100–1108.

P. Li, Y. Li, J.E Seem, Efficient operation of air-side economizer using extremum seeking control, Journal of Dynamic Systems, Measurement, and Control 132 (2010) 031009.

X. Ma, Y. Zhang, X. Li, et al., Experimental study for a high efficiency cascade heat pump water heater system using a new near-zeotropic refrigerant mixture, Appl. Therm. Eng. 138 (2018) 783-794.

B. Mu, Y. Li, J.E Seem, B. Hu, A multivariable Newton-based extremum seeking control for condenser water loop optimization of chilled-water plant, Journal of Dynamic Systems,Measurement, and Control 137 (2015) 111011.

H. Park, D.H. Kim, M.S. Kim, Thermodynamic analysis of optimal intermediate temperatures in R134aeR410A cascade refrigeration systems and its experimental verification, Appl. Therm. Eng. 54 (2013) 319–327.

H. Park, D.H. Kim, M. S. Kim, Performance investigation of a cascade heat pump water heating system with a quasi-steady state analysis, Energy 63 (2013) 283-294.

C. Park, H. Lee, Y. Hwang and R. Radermacher, Recent advances in vapor compression cycle technologies, Int. J. Refrig. 60 (2015) 118-134.

Rotea, M.A., 2000. Analysis of multivariable extremum seeking algorithms. In American Control Conference. Proceedings of the 2000, 433-437, IEEE.

TLK-Thermo GmbH. TIL3.4.2, 2017. <http://www.tlk-thermo.com/index.php>.

B. Wang, H. Wu, J. Li, Z. Xing, Experimental investigation on the performance of NH3/CO2 cascade refrigeration system with twin-screw compressor, Int. J. Refrig. 32 (2009) 1358–1365.

Tube-fin Heat Exchanger Circuitry Optimization For Improved Performance Under Frosting Conditions

Zhenning Li[1] Hongtao Qiao[2] Vikrant Aute[3]*

[1,3] Center for Environmental Energy Engineering, University of Maryland
College Park, MD 20742 USA

[2] Mitsubishi Electric Research Labs, 201 Broadway Cambridge, MA 02139 USA

Abstract

Frost accumulation on tube-fin heat exchanger leads to reduction in evaporator capacity and deteriorates cycle efficiency. The conventional counter-flow heat exchanger circuitry has the disadvantage that more frost tends to accumulate in the first few banks exposed to the incoming air. This frost concentration makes the air side flow resistance increase rapidly, thus reduces the air flow rate and evaporator capacity under constant fan power. In this paper, a novel integer permutation based Genetic Algorithm is used to obtain optimal circuitry design with improved HX performance under frosting conditions. A dynamic HX model with the capability to account for non-uniform frost growth on a fan-supplied coil is used to assess the performance of optimal circuitry. The case study shows that the proposed circuitry design approach yields better circuitry with larger HX capacity, more uniform frost distribution, less air flow path blockage, and therefore longer evaporator operation time between defrost operations.

Keywords: Heat Exchanger, Frost Growth, Circuitry Optimization, Genetic Algorithm

1 Introduction

Tube-fin heat exchangers have wide applications in the refrigeration and air conditioning industry. They are used to transfer heat between air and the working fluid (e.g. refrigerants, water, glycols etc.). One of the major concerns for the refrigeration and heat pump engineers is frost formation on outdoor unit since it can lead to significant reduction on heat exchanger capacity and cycle efficiency.

Frost will accumulate on the surfaces of evaporator coil when the coil surface temperature is below the dew point temperature of incoming air and meanwhile the air dry bulb temperature is below 0 °C. The process of frost formation on the surface of an evaporator coil is a result of two mechanisms: the buildup of small ice particles that exist in the free air stream and accumulate by impaction or interception when they contact the evaporator coil surfaces (Malhammar, 1988) and the diffusion of water vapor onto cold surfaces due to the water vapor concentration difference between the air stream and the frost layer surface (Sanders, 1974).

Formation of frost on a heat exchanger surface results in reduction on heat transfer rate due to fouling characteristics of frost development and blockage of air flow passages through the heat exchanger. Several techniques have been proposed to reduce the frost accumulation rate thereby increasing the evaporator operation time between defrost operations. For example, (Ogawa *et al*, 1993) suggested to use variable geometry tube-fin heat exchangers with different fin geometries on different tube banks to reduce the heat and mass transfer rates at the first few banks exposed to incoming air. However, this geometry modification may be difficult to realize without adding substantial complexity in manufacturing process. (Aljuwayhel *et al*, 2007) developed a heat exchanger frost accumulation model to simulate the performance of counter-flow and parallel-flow circuitry evaporators under frosting conditions. They validated the model by testing the counter-flow circuitry evaporator. They found that heat exchanger circuitry can influence the frost distribution across the evaporator as well as its transient capacity under frosting conditions. Their study shows good circuitry design is a convenient and economic way to reduce the effect of frost accumulation and can provide longer evaporator operation time before defrosting.

As discussed in (Li *et al*, 2018), various performance metrics have been used as objectives for the TFHX circuitry optimization studies, however, in literature there is no study which optimizes circuitry with the goal of improving HX performance under frosting conditions. This paper presents a tube-fin heat exchanger circuitry design approach to tackle this problem.

The remainder of the paper is organized as follows: section 2 details the circuitry optimization approach and analyzes the optimization results from a case study. Section 3 introduces the dynamic HX model with integrated frosting growth model and then demonstrates the efficacy of the proposed circuitry design approach by evaluating the dynamic performance of different circuitry designs under frosting conditions. Conclusions are drawn in Section 4.

2 TFHX Circuitry Optimization

2.1 Integer Permutation Based GA

An integer permutation based genetic algorithm (IPGA) developed by (Li *et al*, 2018) is used to obtain the optimal circuitry designs. (Li and Aute, 2018) has shown that IPGA demonstrates superior capability to obtain better refrigerant circuitries with lower computational cost than the other methods in literature. Meanwhile IPGA can guarantee good manufacturability of resulting circuitries.

Usually, the dynamic simulation of HX performance under frosting conditions is computationally expensive, which means the computational time for a single simulation can take from a few minutes to several hours. Assuming one HX evaluation takes a few minutes to complete, as there will be at least thousands of HX evaluations in one optimization run, using dynamic HX model to evaluate HXs generated by the optimizer is not feasible in the interests of time. This study strives to tackle this problem by exploring an effective problem formulation used in steady state HX optimization in order to generate circuitry designs with desirable dynamic performance under frosting conditions.

At the fitness assignment stage of IPGA, a mass flow based steady state tube-fin heat exchanger model, CoilDesigner® (Jiang *et al*, 2006), is used to evaluate HX performance. This model can account for the refrigerant maldistribution among different circuits by iterating on the pressure residual at the outlet of each circuit.

2.2 Problem Formulation

As explained in previous session, the goal of this study is to explore a HX performance index which can foresee its dynamic performance under frosting conditions. (Qiao *et al*, 2017) observed that the tubes which the frost is the most likely to deposit on are the ones close to the refrigerant inlets with low HX surface temperature. In contrast, the tubes which the frost is the least likely to deposit on are the refrigerant outlet tubes where the in-tube refrigerant flow is superheated. Based on this finding, Equation (1) shows the problem formulation, in which to maximize the total length of superheat tube length is used as the objective. Five constraints are enforced on this problem. The 1st constraint guarantees that the optimal design has equal or larger capacity than the baseline. The 2nd constraint limits that the optimal circuitry has less refrigerant pressure drop than the baseline with 1.1 as the relaxation factor. The 3rd constraint confines the outlet superheat of the entire coil to be similar with that of the baseline within ±1 K variation. The last two constraints are manufacturability constraints. The 4th constraint makes the inlet and the outlet tubes on the same side of HX. The 5th constraint avoids long U-bends stretching across more than 2 tube rows. (Li and Aute, 2018) presents the details of various constraints and constraint handling techniques in IPGA.

$$
\begin{aligned}
&\text{Objective: Maximize(total superheat tube length)}\\
&\mathit{Subject\ to}:\\
&\qquad Q \geq Q_{baseline}\\
&\qquad \Delta P_{refrigerant} \leq 1.1 \times \Delta P_{refrigerant,baseline}\\
&\qquad \Delta Tsat - 1\,K \leq \Delta Tsat \leq \Delta Tsat + 1\,K\\
&\qquad \text{Inlets and outlets on the same side of HX}\\
&\qquad \text{No long U-bend across more than 2 tube rows}
\end{aligned}
\tag{1}
$$

2.3 Baseline Outdoor Heat Exchanger

An outdoor heat exchanger (Figure 1) from a flash tank vapor injection cycle (FTVI) is used as the baseline for circuitry optimization. The steady state heat exchanger model was validated with measured data for this coil in previous research project under different operating conditions (Xu *et al*, 2013). Figure 2 shows that the heat exchanger capacity deviations between CoilDesigner® simulations and experiments are within 6%.

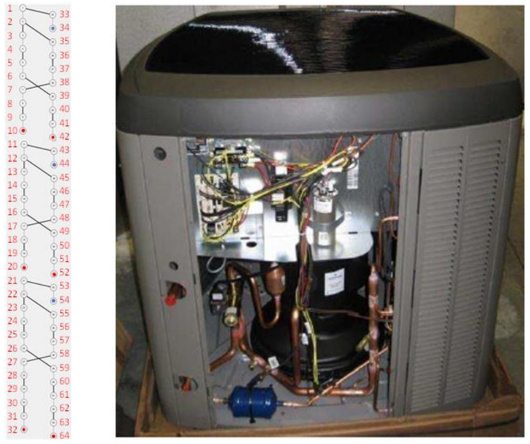


Figure 1. Outdoor unit from FTVI

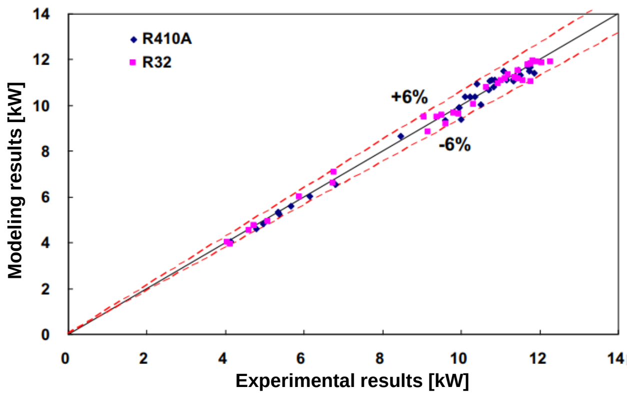


Figure 2. Experiment tests vs CoilDesigner® simulations

Table 1 lists the structural parameters of the baseline HX. Table 2 shows the operating conditions used in the steady state HX simulation. The air side condition is adopted from (ASHRAE, 2010) frost accumulation test. Table 3 lists the empirical correlations adopted for the local heat transfer and pressure drop calculations.

Table 1. Structural Parameters of Baseline Evaporator

Structural Parameters	*Value*
Tube Outer Diameter	7.9 mm
Fins per inch	22 FPI
Fin Type	Wavy Herringbone
Tube Length	2.565 m
Vertical Spacing	24.1 mm
Horizontal Spacing	20.9 mm
Number of Tube Banks	2
Number of Tubes Per Bank	32

Table 2. Operating Conditions of Baseline Evaporator

Operating Conditions	*Value*
Refrigerant	R410A
Refrigerant Inlet Pressure	636.3 kPa
Refrigerant Inlet Quality	0.19
Refrigerant Mass Flow Rate	0.035 kg/s
Air Volume Flow Rate (Uniformly Distributed)	2267 ft^3/min
Air Pressure	101.325 kPa
Air Temperature	1.7 °C
Air Relative Humidity	82 %

Table 3. Correlations Adopted in HX Simulation

Operating Mode	*Heat Transfer*	*Pressure Drop*
Refrigerant Liquid Phase	Gnielinski, 1976	Blasius, 1907
Refrigerant Two Phase	Shah, 2017	Müller-Steinhagen & Heck, 1986
Refrigerant Vapor Phase	Gnielinski, 1976	Blasius, 1907
Air	Kim *et al*, 1997	Kim *et al*, 1997

2.4 Circuitry Optimization Results

For the optimization practice conducted in this case study, the GA population size is set as 200 and the number of generations is set as 500. The GA progress plot (Figure 3) indicates that after 500 generations, the optimal circuitry yields an increase of total superheat tube length from 59.69% to 70.63% of the entire HX tube length.

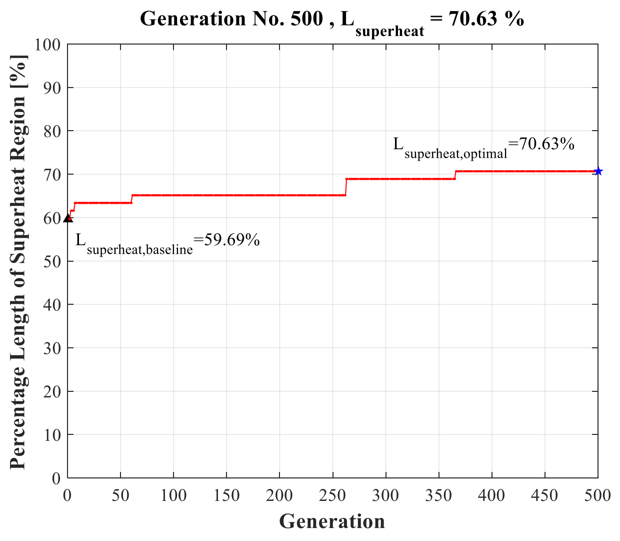


Figure 3. IPGA optimization progress

Figure 4 shows the optimal circuitry design after 500 generations. A solid line represents a U-bend on the front end of the heat exchanger, while a dotted line represents a U-bend on the farther end. Different color represents different circuits. The red tubes are the inlets, while the blue ones are outlets.

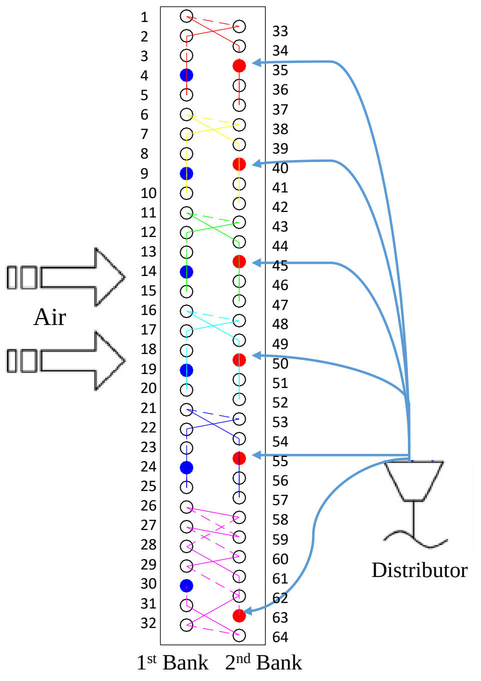


Figure 4. Optimal heat exchanger circuitry

Table 4 compares the steady state performance of the baseline and the optimal design. The optimal design has almost the same capacity as the baseline, while the refrigerant pressure drop yields a decrease from 59.6 kPa to 55.8 kPa by 6.3%. This is because the optimal circuitry in Figure 4 has 6 circuits without merging U-bends. However, the baseline (Figure 1) has six inlets and three outlets, so in each circuit two streams are merged into one. The high refrigerant pressure drop in baseline is induced by the large refrigerant mass flux at the downstream of each circuit. Despite the significant increase of superheat region length, the total outlet superheat of the baseline and the optimal design only varies within 1 K, which is good to avoid refrigerant flooding in compressor. In Table 4, 'U-bends L1' and 'U-bends L2' indicate the number of U-bends which stretches across 1 tube row and 2 tube rows respectively. 'U-bends ≥ L3' is the number of long U-bends which stretches more than 2 tube rows. It can be seen that the optimal circuitry has acceptable manufacturability without long U-bends.

Table 4. Comparison of Baseline VS Optimal Design

Case	*Baseline*	*Optimal*
Capacity [W]	7502	7514
Refrigerant DP [kPa]	59.6	55.8
Outlet Superheat [K]	10.4	11.3
Superheat length [%]	59.7	70.6
U-bends L1	55	40
U-bends L2	6	18
U-bends ≥ L3	0	0

3 Heat Exchanger Simulation under Frosting Conditions

3.1 Dynamic HX Model with Non-Uniform Frost Growth Prediction

In order to evaluate the HX performance under frosting conditions, a distributed-parameter heat exchanger model (Qiao *et al*, 2015) implemented using Modelica language on Dymola 4.7 (Dassault Systemes, 2014) is used. This dynamic heat exchanger model is integrated with a detailed frost growth prediction model (Qiao *et al*, 2017) to account for non-uniform frost accumulation on a fan-supplied coil. The air flow redistribution is solved by linearizing a system of non-linear air pressure drop equalization equations. Readers are referred to original papers for details of the dynamic HX model and the frost growth model.

It is worth mentioning that this HX model as well as the frosting growth model were validated with measured data for the flash tank vapor injection heat pump cycle as described in section 2.3. Figure 5 shows the comparison between the dynamic simulation results and experimental data for this FTVI system in terms of HX capacity, power consumption and cycle COP. It can be seen that the HX model with frost growth model integrated can predict the transients of this system with reasonable accuracy under frosting conditions.

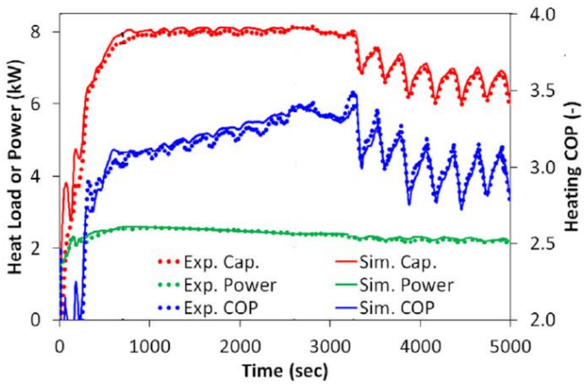


Figure 5. Frosting experiment tests vs simulations (Qiao *et al*, 2017)

3.2 Transient Simulation Results and Discussion

In this section, the baseline and optimal circuitry designs are simulated for 6000 seconds frosting test. The adopted operating conditions is the same as those in Table 2 (ASHRAE, 2010).

Figure 6 compares the HX capacity between the baseline and optimal design. Both capacities decrease as frost accumulates on HX surface. However, the capacity of the baseline decreases faster than the optimal design, which ultimately yields a 11.6% capacity difference at the end of 6000s frosting test.

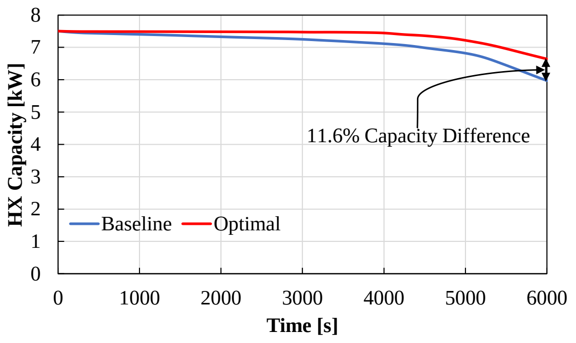


Figure 6. Evaporator capacity transients

Figure 7 shows the transients of the total frost mass accumulation on HX surface. At the early stage of the frosting test, the frost mass on the two HXs are very close. The frost mass starts to show noticeable difference after 3300s and the difference ends up to be 5.1% at 6000s (4.12 kg frost on the baseline and 3.91 kg frost on the optimal design).

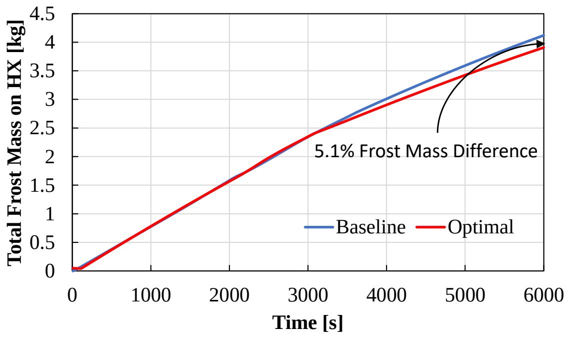


Figure 7. Frost mass transients

Figure 8 presents the frost mass distribution on each tube bank of the two HXs. It is evident that the 1st tube bank of the baseline has much faster frost accumulation rate. Although the frost growth rate for the 2nd bank of the baseline is less than the frost growth rate of the 2nd bank of the optimal design, it doesn't provide much benefit for the baseline, because the large amount of frost on the 1st bank tends to dominate the air flow resistance of the entire coil. In fact, the slow frost growth on the 2nd bank attributes to the insufficient air which can flow through the 1st bank to reach the 2nd bank.

In other words, despite the total frost growth rate on the two HXs are similar (Figure 7), the bank-wise frost distribution on the optimal circuitry is more uniform than that of the baseline (Figure 8).

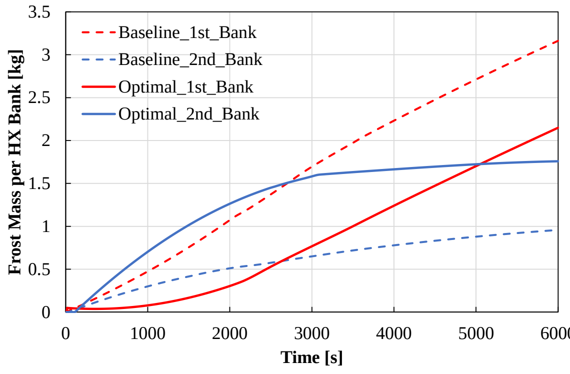
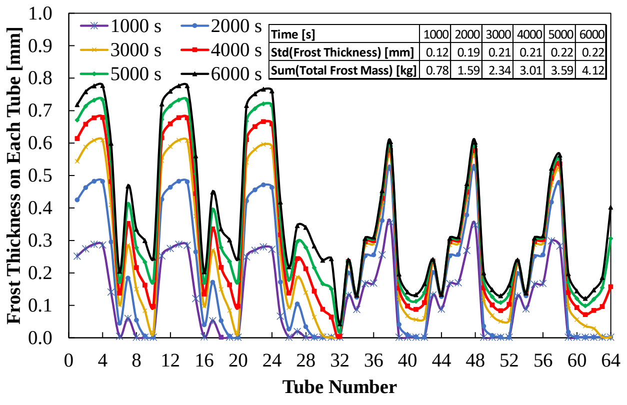


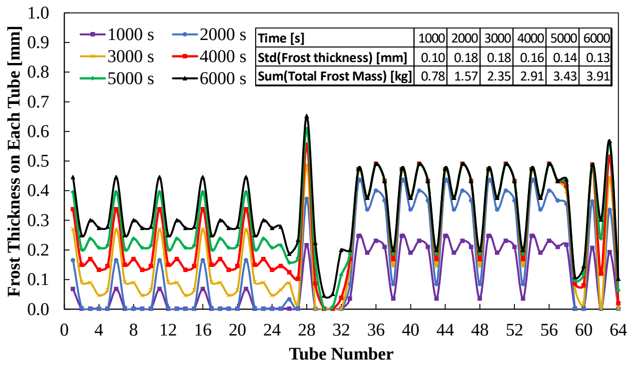
Figure 8. Frost distribution on different banks

Figure 9 and Figure 10 perform an in-depth observation on the frost distribution by presenting the time evolution of the frost thickness on each tube for the two HXs. It is prominent that the frost thickness on the tubes at the 1st bank of optimal design (Figure 10, tube #1 to #32) are much less than that of the baseline (Figure 9, tube #1 to #32). This indicates that in addition to the uniform bank-wise frost distribution, the optimal design also yields more uniform frost distribution on each tube. As listed in Figure 9 and Figure 10, at the end of the test (6000s), the standard deviation of frost thickness among all tubes is 0.22 mm for the baseline, while that of the optimal is only 0.13 mm. This result implies that the proposed circuitry optimization approach can generate circuitry design with more uniform frost distribution.



Time [s]	1000	2000	3000	4000	5000	6000
Std(Frost Thickness) [mm]	0.12	0.19	0.21	0.21	0.22	0.22
Sum(Total Frost Mass) [kg]	0.78	1.59	2.34	3.01	3.59	4.12

Figure 9. Evolution of per tube frost thickness (baseline)



Time [s]	1000	2000	3000	4000	5000	6000
Std(Frost thickness) [mm]	0.10	0.18	0.18	0.16	0.14	0.13
Sum(Total Frost Mass) [kg]	0.78	1.57	2.35	2.91	3.43	3.91

Figure 10. Evolution of per tube frost thickness (optimal)

Figure 11 shows the transients of the air volume flow rate during the frost buildup process. It can be seen that the air flow rate reduction of the baseline coil is substantially higher than that of the optimal design. The non-uniform frost distribution results in 16.7% air flow rate difference at the end of the frosting test. This reinforces the previous finding that the capacity merit of the optimal design attributes to the superior characteristics of the frost accumulation and the associated effect on the air side pressure drop.

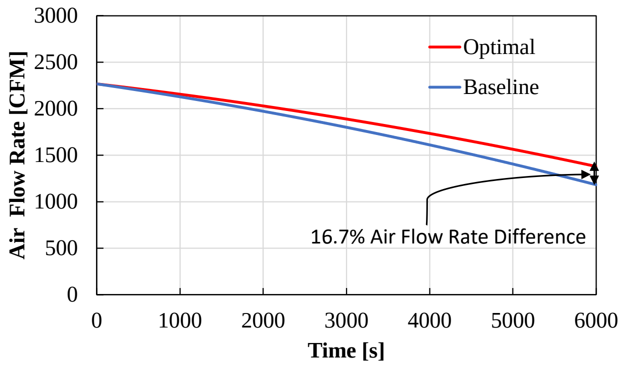


Figure 11. Transients of calculated air volume flow rate

4 Conclusion

This study proposes a novel TFHX circuitry optimization approach to obtain circuitry design with improved HX performance under frosting conditions. In order to verify the efficacy of the proposed approach, a dynamic HX model integrated with frost growth model is used to evaluate the performance of baseline and optimal circuitry designs. The results show that the optimal circuitry leads to a predicted 11.6% HX capacity improvement and 5.1% frost mass reduction at the end of the 6000 seconds frosting test. Although the actual improvement from the optimal circuitry design needs to be further validated by manufacturing and testing the HX, these results demonstrate that the proposed approach has potential to generate circuitry designs with larger HX capacity, more uniform frost distribution and therefore longer evaporator operation time between defrost operations.

Acknowledgements

This work was supported by the Modeling and Optimization Consortium at the University of Maryland.

References

Aljuwayhel, N., Reindl, D., Klein, S., and Nellis, G. (2007). Comparison of parallel-and counter-flow circuiting in an industrial evaporator under frosting conditions. *International Journal of Refrigeration, 30*(8), 1347-1357.

ASHRAE. (2010). Methods of testing for rating seasonal efficiency of unitary air conditioners and heat pumps. *ANSI/ASHRAE Standard*, Atlanta, GA, USA.

Blasius, H. (1907). *Grenzschichten in Flüssigkeiten mit kleiner Reibung*: Druck von BG Teubner.

Dassault Systemes (2014). Dymola 7.4.

Gnielinski, V. (1976). New equations for heat and mass transfer in turbulent pipe and channel flow. *Int. Chem. Eng., 16*(2), 359-368.

Jiang, H., Aute, V., and Radermacher, R. (2006). CoilDesigner: a general-purpose simulation and design tool for air-to-refrigerant heat exchangers. *International Journal of Refrigeration, 29*(4), 601-610.

Kim, N.-H., Yun, J.-H., and Webb, R. (1997). Heat transfer and friction correlations for wavy plate fin-and-tube heat exchangers. *Journal of Heat Transfer, 119*(3), 560-567.

Li, Z. and Aute, V. (2018). Optimization of Heat Exchanger Flow Paths Using a Novel Integer Permutation Based Genetic Algorithm. *EngOpt 2018 Proceedings of the 6th International Conference on Engineering Optimization, Lisboa, Portugal*.

Li, Z., Ling, J., and Aute, V. (2018). *Tube-fin heat exchanger circuitry optimization using integer permutation based genetic algorithm*. Paper presented at the 17th International Refrigeration and Air Conditioning Conference Purdue.

Malhammar, A. (1988). Monitoring frost growth in evaporators is a complex process. *Australian Refrigeration, Air conditioning and Heat*.

Müller-Steinhagen, H. and Heck, K. (1986). A simple friction pressure drop correlation for two-phase flow in pipes. *Chemical Engineering and Processing: Process Intensification, 20*(6), 297-308.

Ogawa, K., Tanaka, N., and Takeshita, M. (1993). *Performance improvement of plate fin-and-tube heat exchangers under frosting conditions.* Paper presented at the the 1993 Winter Meeting of ASHRAE Transactions. Part 1, Chicago, IL, USA, 01/23-27/93.

Qiao, H., Aute, V., and Radermacher, R. (2015). Transient modeling of a flash tank vapor injection heat pump system– Part I: Model development. *International Journal of Refrigeration, 49*, 169-182.

Qiao, H., Aute, V., and Radermacher, R. (2017). Dynamic modeling and characteristic analysis of a two-stage vapor injection heat pump system under frosting conditions. *International Journal of Refrigeration, 84*, 181-197.

Sanders, C. T. (1974). The influence of frost formation and defrosting on the performance of air coolers.

Shah, M. M. (2017). Unified correlation for heat transfer during boiling in plain mini/micro and conventional channels. *International Journal of Refrigeration, 74*, 606-626. doi:10.1016/j.ijrefrig.2016.11.023

Xu, X., Hwang, Y., and Radermacher, R. (2013). Performance comparison of R410A and R32 in vapor injection cycles. *International Journal of Refrigeration, 36*(3), 892-903.

Coupled Simulation of a Room Air-conditioner with CFD Models for Indoor Environment

Hongtao Qiao[1*] Xu Han[2] Saleh Nabi[1] Christopher R. Laughman[1]

[1]Mitsubishi Electric Research Laboratories, USA, {qiao,nabi,laughman}@merl.com
[2]University of Colorado Boulder, USA, Xu.Han-2@colorado.edu

Abstract

Coupled simulation of building energy systems (BES) and computation fluid dynamics (CFD) often focuses on the integration of air handlers with indoor environment, and does not incorporate vapor compression systems into the analysis, yielding inaccurate prediction of building energy consumption. This paper presents a coupled simulation to explore the pull-down performance of a room air conditioning system. The dynamic models of the air-conditioner are constructed in Modelica, whereas the indoor environment is simulated in OpenFOAM. Dynamic characteristics will be compared with different vane angles and airflow modes. Numerical simulations demonstrate that both vane angle and airflow mode exhibit pronounced impact on the pull-down time. Meanwhile, the well-mixed assumption that most of building energy simulation programs are built upon exhibits drastically different dynamic characteristics compared to the detailed CFD model, suggesting that neglecting non-uniform air flow and temperature distributions in buildings might lead to significant errors in control design.

Keywords: Modelica, OpenFOAM, co-simulation, building energy simulation, vapor compression system

1 Introduction

Buildings account for 40% of the primary energy in the U.S. and are important sources of CO_2 emission. Currently, roughly half of energy consumption in buildings is used by spacing heating and cooling. These end-uses represent significant opportunities to reduce energy consumption, improve energy security and reduce CO_2 emissions. Therefore, efficiency improvements in HVAC systems become more enticing and will play a crucial role in energy revolution.

Due to the multi-scale, highly nonlinear and complex nature of their dynamic characteristics, design of high energy-efficient HVAC systems nowadays heavily relies on computer simulations because they can not only provide verifiable and robust predictions of system performance, but also radically accelerate the design, testing, and specification of these systems by drastically reducing the number of experimental iterations required. Meanwhile, model-based design approach is often used to develop and evaluate advanced control algorithms of HVAC systems.

The assessment of closed-loop control performance of HVAC systems often requires an integrated approach that couples the dynamic models of HAVC system with building energy simulation programs. Most of these building simulation programs assume that indoor air is well mixed in order to simplify the computation. However, this prevailing assumption fails to simulate the distribution of temperature, pressure, concentration in buildings with large space and high heat gain. As a result, these programs cannot accurately predict building energy consumption and the closed-loop performance of HVAC systems. In addition, they cannot satisfy advanced design requirements, such as personal cooling/heating in occupied zone and optimal sensor placement, due to lack of local thermal comfort information (Zhai et al., 2002).

In contrast with the well-mixed assumption, computational fluid dynamics (CFD) divides fluid domain into a large number of small volumes such that a detailed prediction of airflow and temperature distributions, thermal comfort and indoor air quality can be obtained. Consequently, coupling building energy simulation programs with CFD can be effective to overcome the deficiencies of stand-alone programs and achieve better results. In such a way, building simulations can provide dynamic boundary conditions to CFD, whereas CFD simulates the airflow dynamics based on these boundary conditions and then can send local airflow and temperature information back to building simulations such that the closed-loop performance of HVAC systems can be evaluated.

Recently, there is an emerging interest in using fast fluid dynamics (FFD) for indoor environment simulation instead of full-scale CFD because of superior computational speed of FFD (Zuo et al., 2016). Compared with CFD, however, FFD possesses many limitations. First of all, FFD does not support unstructured mesh and thus cannot deal with complex geometry. Meanwhile, FFD does not have turbulence flow models, which may yield inaccurate predictions in airflow dynamics. Moreover, FFD assumes that heat flow is uniformly injected into space and therefore cannot handle non-uniform heat source/sink. All these limitations impose great challenges when applying FFD in indoor environment simulations.

Comprehensive review of the literature regarding the coupled simulation of building energy systems and the indoor environment (Tian et al., 2018) indicates that the previous studies only focused on the integration of air handler units with indoor environment, whereas none of these studies ever took into account the dynamics of vapor compression systems, which are the essential part of building HVAC systems through which heat is removed from or added to buildings. Also, they are the primary energy consumer compared to other auxiliary components, such as pumps and fans. Without incorporating vapor compression systems into the coupled analysis, it is impossible to accurately predict building energy consumption. Meanwhile, feedback control design of such systems requires to account for the complex dynamic characteristics of phase-changing refrigerant flow, which are affected by air flows inside buildings. Therefore, it is worthwhile to couple vapor compression system with indoor environment to explore the closed-loop dynamic performance of these systems.

Another interesting finding through literature review is that the reported coupled simulations are all based on the commercial CFD programs, e.g., Fluent, CFX and STAR-CD, rather than open-source code. These CFD programs are extremely expensive and might not be readily available for everyone. Development and maintenance of the middleware that couples the commercial CFD programs with building energy simulations could be costly and time-consuming since the source code is not generally accessible. As an open-source and well recognized CFD engine, OpenFOAM possesses many advantages compared to the commercial programs because of unrestricted access to its source code. Full control on OpenFOAM simulator makes integration with other programs through middleware much easier.

To bridge the identified research gap, this paper attempts to couple the dynamic models of a wall-mounted split-type air-conditioner with the detailed CFD model for indoor environment to explore its closed-loop pull-down performance with different vane angles and airflow modes. The dynamic models of the air-conditioner are constructed in Modelica, whereas the indoor environment is simulated in OpenFOAM. Therefore, a co-simulation platform that couples Modelica with OpenFOAM needs to be developed. Dynamic characteristics will be compared against those obtained by the well-mixed air model to demonstrate the effect of non-uniform airflow distribution on the system performance.

The remainder of the paper is organized as follows. In Section 2, the coupling mechanisms between Modelica and OpenFOAM are presented. In Section 3, we describe the dynamic models for the air-conditioning system and the models for air flow dynamics. In Section 4, we discuss the influence of the vane angle and airflow mode on the pull-down performance of the air-conditioning system. Conclusions from this work are then summarized in Section 5.

2 Coupling Mechanisms

A middleware interface is required to facilitate data exchange between individual programs, i.e., Modelica and OpenFOAM in this study. A quasi-dynamic data synchronization scheme is used in the coupled simulation, which means that the received data of individual programs are discrete with time and remain unchanged between two successive synchronization points, and will be only updated when the next synchronization point is reached. The synchronization time step needs to be predefined before simulation, and cannot change during simulation. Please be noted that both Modelica and OpenFOAM have their respective integration time steps, which can be either fixed or adaptive and are usually much smaller than the synchronization time step.

Zuo et al. (2016) developed a coupled simulation between FFD and the Modelica Buildings library for the dynamic ventilation system with stratified air distribution. In order to make life easier, we decided to modify their framework so that OpenFOAM can communicate with Modelica. Therefore, the implementation on the Modelica side and between Modelica and middleware requires no modifications. Different from that Modelica uses external C functions to exchange data with the middleware, OpenFOAM uses its built-in `externalCoupled` function object which provides a file-based communication interface to transfer data to and from OpenFOAM (OpenFOAM Foundation, 2017), as shown in Fig. 1. The data exchange employs specialized boundary conditions to provide either uni- or bi-directional models. At start-up, the `externalCoupled` function creates a lock file, i.e., `OpenFOAM.lock`, to signal the external source, i.e., middleware in this case, to wait. During the boundary condition update, boundary values are written to file, e.g., `data.out`. The lock file is then removed, instructing the external source to take control of the program execution. When ready, the external program should create the return values to file, e.g., `data.in`, and then re-instate the lock file. The `externalCoupled` function will then read the return values, and pass program execution back to OpenFOAM. The logic of data exchange between Modelica and OpenFOAM is shown in Fig. 2.

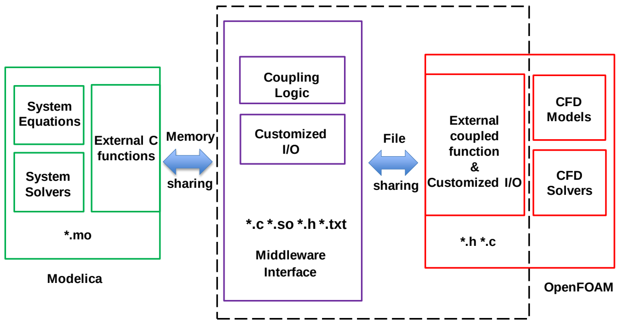


Figure 1. Coupling between Modelica and OpenFOAM.

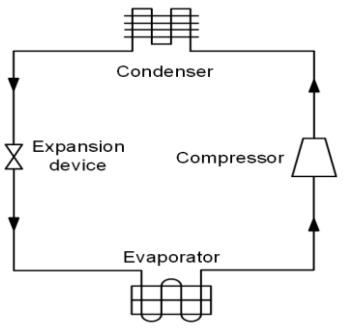


Figure 3. Schematic of an air-conditioning system.

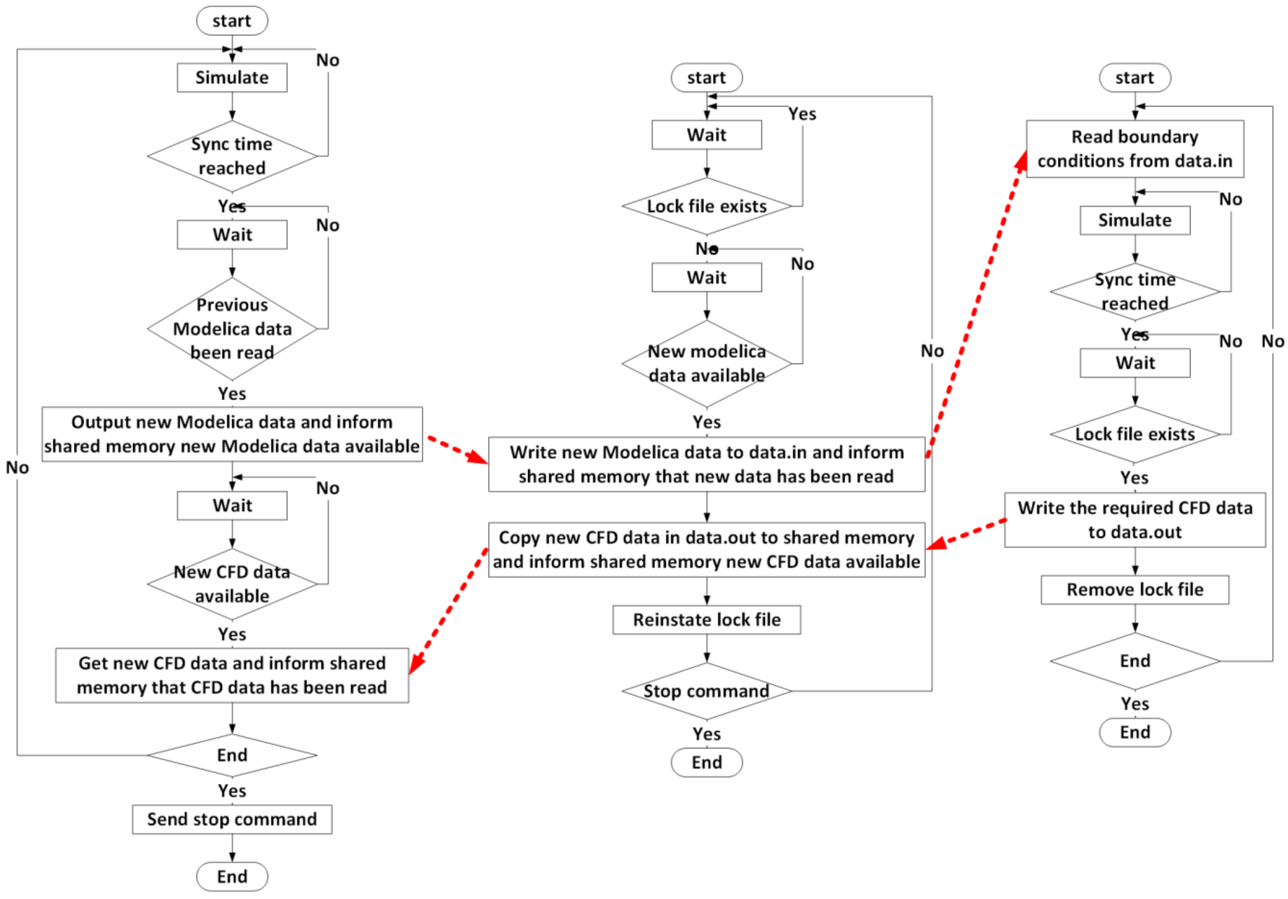

start
Simulate
Sync time reached
No
Yes
Wait
No
Previous Modelica data been read
Yes
Output new Modelica data and inform shared memory new Modelica data available
Wait
No
New CFD data available
Yes
Get new CFD data and inform shared memory that CFD data has been read
End
No
Yes
Send stop command
End
start
Wait
Yes
Lock file exists
No
Wait
No
New modelica data available
Yes
Write new Modelica data to data.in and inform shared memory that new data has been read
Copy new CFD data in data.out to shared memory and inform shared memory new CFD data available
Reinstate lock file
Stop command
No
Yes
End
start
Read boundary conditions from data.in
Simulate
No
Sync time reached
Yes
Wait
No
No
Lock file exists
Yes
Write the required CFD data to data.out
Remove lock file
End
Yes
End

Figure 2. Logic of data exchange in co-simulation.

3 Model Development

The studied air-conditioning system consists of two main parts: the outdoor unit and the indoor unit. The outdoor unit is installed on or near the wall outside of the room or space that you wish to cool. The outdoor unit houses the compressor, condenser, whereas the indoor unit contains the evaporator, expansion device, a blower fan and an air filter. Fig. 3 illustrates the schematic of a typical air-conditioning system.

3.1 Compressor Model

A variable-speed low-side scroll compressor, in which the motor is cooled by compressed low-pressure suction refrigerant, was used in this work. Because the performance map for the compressor has reduced accuracy when extrapolated beyond the specific ranges of saturated discharge and suction temperatures over which it is tabulated, we converted the performance map into a set of curve-fitted equations to avoid poor numerical behavior during the simulations.

The volumetric efficiency of this compressor model is a function of the suction pressure, discharge pressure and compressor frequency

$$\eta_v = \kappa_0 + \kappa_1\varphi + \kappa_2\varphi^2 + \kappa_3\left(p_{dis} - p_{suc}\right)\left(1+\kappa_4 p_{suc}\right) \quad (1)$$

where $\varphi = p_{dis}/p_{suc}$, $\kappa_i = a_{i,1} + a_{i,2}\varpi + a_{i,3}\varpi^2$, and $\varpi = f/f_{nom}$. The power consumed by the compressor is determined by

$$\dot{W} = \zeta_1 p_{suc}\dot{V}_{suc}\left(\varphi^{\zeta_2} - \zeta_3\right) + \zeta_4 \quad (2)$$

where $\zeta_i = b_{i,1} + b_{i,2}\varpi + b_{i,3}\varpi^2$. In addition, the fraction of the compressor power absorbed by the refrigeration is determined by

$$\lambda = \theta_0 + \theta_1\varphi + \theta_2\varphi^2 \quad (3)$$

where $\theta_i = c_{i,1} + c_{i,2}\varpi + c_{i,3}\varpi^2$. Lastly, the mass flow rate delivered by the compressor is determined by

$$\dot{m} = \eta_v f \rho_{suc} V_{disp} \quad (4)$$

and the enthalpy of the discharged refrigerant is

$$h_{dis} = \lambda\dot{W}/\dot{m} + h_{suc} \quad (5)$$

3.2 Expansion Valve Model

Linear electronic expansion valves (LEVs) were also used in the system. A standard orifice-type relationship between mass flow rate and pressure drop across the valve was used to describe the system behaviour, in which the mass flow rate through the valve is determined by the flow coefficient, the inlet density, and the pressure difference across the valve

$$\dot{m} = C_v\sqrt{\rho_{in}\Delta p} \quad (6)$$

where the flow coefficient C_v is a function of the valve opening determined by regression based on experimental data.

3.3 Heat Exchanger Models

The heat exchangers were described by a finite-volume model of the heat and fluid flow. Each heat exchanger segment is divided into three sections: the refrigerant stream, the finned walls, and the air stream. The refrigerant stream is described by a one-dimensional flow with fluid properties varying only in the direction of flow; consequently, these properties are uniform or averaged at every cross section along the axis of the channel. Additional assumptions used to simplify the dynamic models included: (1) the fluid is Newtonian, (2) axial heat conduction in the direction of refrigerant flow is ignored, (3) viscous dissipation is neglected, (4) liquid and vapor are in thermodynamic equilibrium in each volume in the two-phase region, (5) the potential energy and kinetic energy of the refrigerant are neglected, and (6) dynamic pressure waves are of minor importance and are thus neglected in the momentum equation (Brasz and Koenig, 1983). The conservation laws can thus be formulated as follows:

$$\frac{\partial}{\partial t}\left(\bar{\rho}A\right) + \frac{\partial}{\partial z}\left(GA\right) = 0 \quad (7)$$

$$\frac{\partial}{\partial t}\left(\bar{\rho}\bar{h}_\rho A\right) + \frac{\partial}{\partial z}\left(G\bar{h}A\right) = Pq_w'' + \frac{\partial}{\partial t}\left(pA\right) \quad (8)$$

$$A\frac{\partial p}{\partial z} + \bar{\tau}_w P = 0 \quad (9)$$

where average density, average mass flux, average density-weighted enthalpy, average flow-weighted enthalpy, and average wall shear stress are defined as follows, respectively.

$$\bar{\rho} = \frac{1}{A}\int_A \rho dA\,,\ G = \frac{1}{A}\int_A \rho u dA\,,\ \bar{h}_\rho = \frac{1}{\bar{\rho}}\left(\frac{1}{A}\int_A \rho h dA\right),$$

$$\bar{h} = \frac{1}{G}\left(\frac{1}{A}\int_A \rho u h dA\right),\ \bar{\tau}_w = \frac{1}{P}\int_P \tau_w dl\,.$$

For single-phase flow, the refrigerant density can be presumed to be uniform at each cross sectional area, and the following relations can be readily obtained

$$\bar{\rho} = \rho \quad (10)$$

$$\bar{h}_\rho = \bar{h} = h \quad (11)$$

This work relaxed the typical homogeneous flow assumption of equality between the velocities of vapor and liquid for control volumes in the two-phase region to achieve a more realistic refrigerant system mass inventory for the overall system. This necessitated the use of the average density and specific enthalpies for two-phase flow, which are determined by

$$\bar{\rho} = \rho_g\gamma + \rho_f\left(1-\gamma\right) \quad (12)$$

$$\bar{h}_\rho = \left[\rho_g h_g\gamma + \rho_f h_f\left(1-\gamma\right)\right]/\bar{\rho} \quad (13)$$

$$\bar{h} = h_g x + h_f\left(1-x\right) \quad (14)$$

where the void fraction and flow quality are defined as

$$\gamma = \frac{A_g}{A} \quad (15)$$

$$x = \frac{\rho_g u_g\gamma}{G} \quad (16)$$

The flow quality x is distinct from the static quality $\hat{x}$, which is defined as the ratio of mass of vapor to that of the total mixture,

$$\hat{x} = \frac{M_g}{M_g + M_f} = \frac{\rho_g\gamma}{\rho_g\gamma + \rho_f\left(1-\gamma\right)} \quad (17)$$

Hence, the density-weighted enthalpy can be described using the static quality

$$\bar{h}_\rho = \hat{x}h_g + \left(1-\hat{x}\right)h_f \quad (18)$$

The pressures and density-weighted enthalpies of each control volume are used as the state variables in the heat exchanger model. Density-weighted enthalpy can be related to flow-weighted enthalpy via

$$\bar{h} - \bar{h}_\rho = \left(x - \hat{x}\right)\left(h_g - h_f\right) \quad (19)$$

Note that x and $\hat{x}$ are both zero for the subcooled liquid and unity for the superheated vapor, implying that these two enthalpies should be equal to each other for single-phase flows. All of the variables in Eq. (19) are thermodynamic properties, and can be readily calculated except for the flow quality x.

While this refrigerant-side model was principally developed for the heat exchangers, it was also used to describe the behavior of the refrigerant pipes connecting the components. These pipe models were an important part of the overall system model because they contained a significant fraction of the refrigerant in the entire system. The main distinction between the pipe models and the heat exchanger models regards the air-side models used: the heat exchanger air-side models describe the effect of the enhanced heat transfer surface, while the pipe air-side models use natural convection as the boundary condition.

The model of the air-side of the heat exchanger is based on the following assumptions: (1) one-dimensional quasi-steady airflows, (2) negligible heat conduction in direction of the air flow, (3) the temperature profile within fins follows the steady-state profile, allowing the use of heat transfer and combined heat and mass transfer fin efficiencies, (4) simultaneous heat and mass transfer follows the Lewis analogy, and (5) equal temperature of tube wall and fins within each control volume. Under these assumptions, the governing equations for the air side describing the energy balances for the wetted coils are

$$\dot{m}_a c_{p,a} \frac{dT_a}{dy} \Delta y = \alpha_a \left(A_{o,t} + \eta_{fin} A_{o,fin} \right) \left(T_w - T_a \right) \tag{20}$$

$$\dot{m}_a \frac{d\omega_a}{dy} \Delta y = \alpha_m \left(A_{o,t} + \eta_{fin} A_{o,fin} \right) \min\left(0, \omega_{w,sat} - \omega_a \right) \tag{21}$$

where $\omega_{\mathrm{w,sat}}$ is the humidity ratio of saturated air evaluated at T_{w}. The mass transfer coefficient α_{m} is determined by applying the Lewis analogy.

Finally, without considering the axial conduction along the tube, the energy equation of the tube walls and associated fins can be written as

$$\left(M_t c_{p,t} + M_{fin} c_{p,fin} \right) \frac{dT_w}{dt} = q_r + q_a \tag{22}$$

$$q_r = \alpha_r A \left(T_r - T_w \right) \tag{23}$$

$$q_a = \dot{m}_a \left[c_{p,a} \left(T_{a,in} - T_{a,out} \right) + \left(\omega_{a,in} - \omega_{a,out} \right) \Delta h_{fg} \right] \tag{24}$$

3.4 CFD Model

CFD uses numerical simulations to predict the fluid flow phenomena based on the conservation laws (conservation of mass, momentum and energy) governing fluid motion. The general conservation equation of any quantity ϕ is given by

$$\frac{\partial}{\partial t}(\rho\phi) = -\nabla \bullet (\rho \mathrm{U} \phi) + \nabla \bullet (D \nabla \phi) + S_\phi \tag{25}$$

In the equation above, D stands for the diffusion coefficient, that can be a scalar or a vector and S_ϕ stands for any kind of sources or sinks that influence the quantity ϕ. Now we are able to simply derive the mass, momentum and other conservative equations out of this by replacing the quantity ϕ by the quantity of interest.

The partial differential conservation equations are more complex than they appear and they are non-linear, coupled, and difficult to solve. Therefore, discretization methods, e.g., finite volume method, are used to approximate the differential equations by a system of algebraic equations, which can be solved on a computer. The approximations are applied to small domains in space and/or time so the numerical solution provides results in discrete locations in space and time.

4 Case Study

The objective of this numerical study is to explore the dynamic performance of a wall-mounted split-type air-conditioner with different vane angles and airflow modes during pull-down operation. The air conditioning system used R410A as the working fluid. The compressor was a low-side scroll compressor with displacement of 6.8 cm^3 and nominal rotational speed of 3500 rpm, and both heat exchangers were louvered fin-and-tube heat exchangers. The heat exchanger models were augmented by a set of empirical closure relations describing the single- and two-phase heat transfer coefficients and frictional pressure drops for the fluid on both the refrigerant side and the air-side (Qiao et al., 2015). The Levy void fraction model (Levy, 1967) was used to compute the two-phase refrigerant mass inventory. A tube-by-tube approach was employed for the heat exchanger analysis, i.e., the performance of each tube was analyzed separately and each tube was associated with different refrigerant and air parameters.

Two PI controllers in the air-conditioning system adjust the EEV opening and compressor speed to control the suction superheat at 3K and the return air temperature at 26°C, respectively. The values of PI gains are given in Table 1.

The indoor unit is installed at the center of the wall in a room with dimensions of 5 × 5 × 2.6 m (length × width × height), as shown in Fig. 4. The solver settings and boundary conditions for the CFD room model are summarized in Table 2. To reduce the calculation load, the indoor unit is simplified as a cube with dimensions of 0.70 × 0.25 × 0.30 m (length × width × height). The inlet is at the bottom of the indoor unit, while the outlet is on the top of the indoor unit.

The closed-loop performance of the air-conditioner is compared with three different vane angles and two airflow modes. The vane angle is defined as the angle between the horizontal plane and the up-down flap (Fig. 4). Three vane angles are small (15°), medium (45°) and high vane angle (75°), respectively. Airflow rate of the indoor unit is regulated by the speed of the fan in the unit. The supply air flow rate is 0.08 m^3/s and 0.15 m^3/s for low and high fan speeds, respectively. According to Lee et al. (2017), the air flow rate only changes slightly when the vane angle varies since the flow resistance of the up-down flap is very small. In the presented study, therefore, the same air flow rate is used for the same airflow mode regardless of vane angles.

Modelica air-conditioner models define the inlet boundary conditions for the CFD model, i.e., the inlet air temperature and velocity, while CFD calculates the outlet air temperature for the Modelica models. Meanwhile, Modelica models provide the time-average surface temperatures of the side walls and the ceiling, and CFD calculates the surface heat flux for the Modelica models. The synchronization time step is 10 sec. The Modelica models are implemented using the Dymola 2019 simulation environment. The co-simulation runs on a desktop with an Intel i7-2600 processor with 8 cores and 8 Gb of RAM, and the ratio of the physical time to the CPU time is approximately 1:3.

Table 1. Gains of PI controllers.

Parameter	*Value*
K_p (Hz/°C) of compressor rpm controller	2.0
T_i (sec) of compressor rpm controller	250
K_p (Count/K) of suction SH controller	1.25
T_i (sec) of suction SH controller	66

Table 2. Solver settings and boundary conditions of CFD room model.

Item	*Content*
Computational domain	5 × 5 × 2.6 m (length × width × height)
Turbulence model	k-epsilon turbulence model
Time dependent	Transient simulation (Courant number < 1)
Inlet boundary condition	externalCoupledTemperature & externalCoupledVelocity
Outlet boundary condition	zeroGradient for temperature and velocity
Side walls & ceiling	Isothermal condition (26.85°C)
Floor	fixed temperature gradient (fixed at 356 K/m)
Solver	buoyantPimpleFoam
Scheme	MUSCL

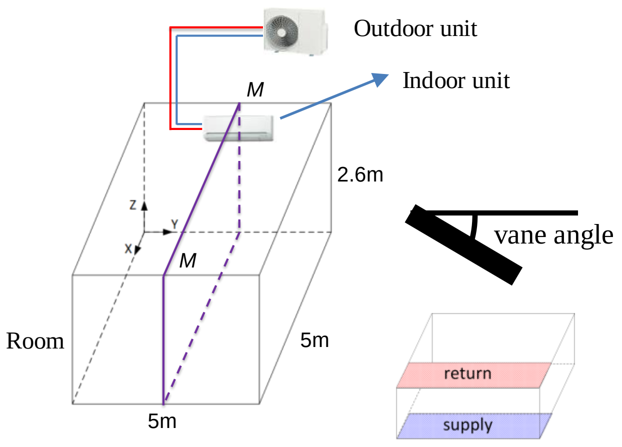
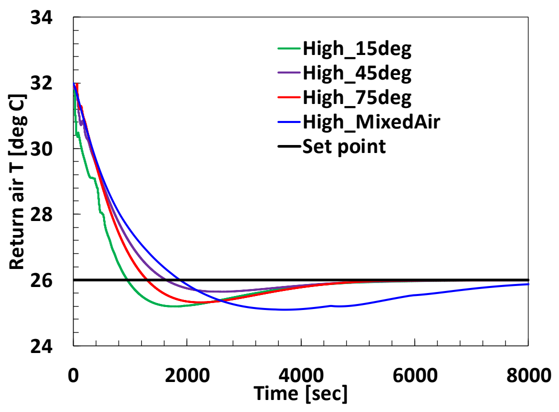
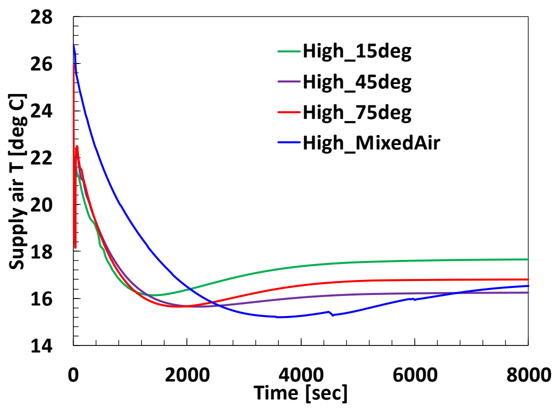


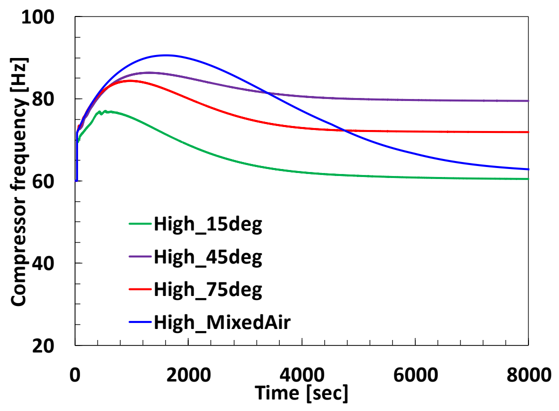
Figure 4. Wall-mounted air-conditioner in a room.



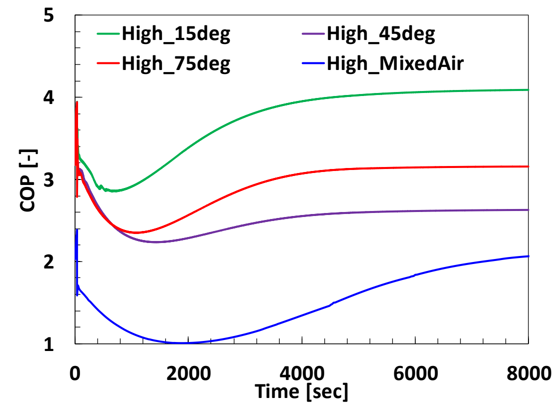
(a)



(b)



(c)



(d)

Figure 5. Dynamics of air-conditioning system in high airflow mode.

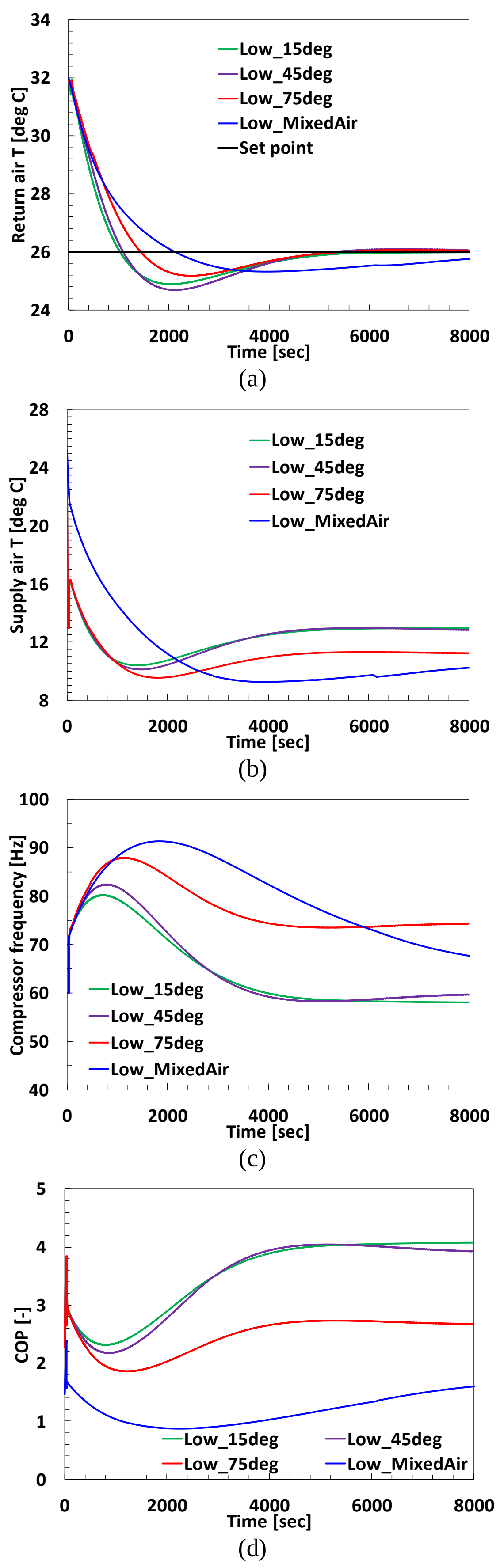


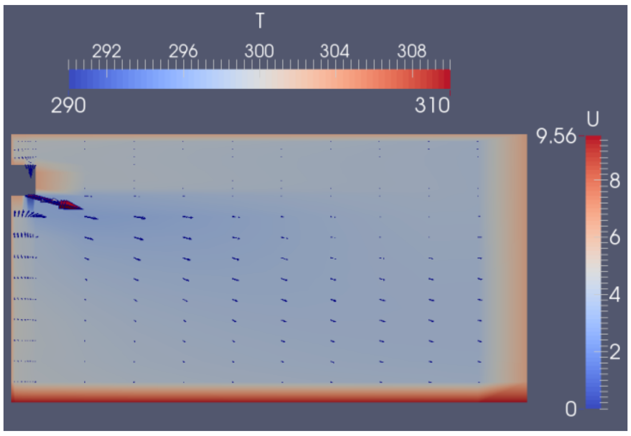
Figure 6. Dynamics of air-conditioning system in low airflow mode.

Fig. 5 illustrates the dynamic characteristics of return air temperature (a), supply air temperature (b), compressor frequency (c) and coefficient of performance (COP) (d) at different vane angles in the high airflow mode, respectively. It is evident from Fig. 5a that the vane angle affects the pull-down time, which is defined as the time required to bring down the temperature of room air from the initial temperature to the final desired temperature. The pull-down time for vane angles of 15°, 45° and 75° is 950 sec, 1630 sec and 1300 sec, respectively. When the vane angle is 15°, the jet flow takes the shortest distance to reach the outlet and entrains the least hot air from the heated floor during the period, resulting in the fastest pull-down. When the vane angle is 75°, the jet flow takes much longer distance to reach the outlet and exhibits a prolonged pull-down time. When the vane angle is 45°, the jet flow reaches the vicinity of the floor center, showing the strongest mixing effect with the hot air on top of the heated floor, resulting in the slowest pull-down. Fig. 5b reveals the transients of supply air temperature and it can be observed that the supply air temperatures vary quite differently for different vane angles under the closed-loop control. Fig. 5c shows the variations of compressor speed during pull-down operation. Given that 15° vane angle has the best pull-down performance, it is straightforward to expect that this vane angle results in the lowest compressor speed and the highest COP, as shown in Fig. 5d.

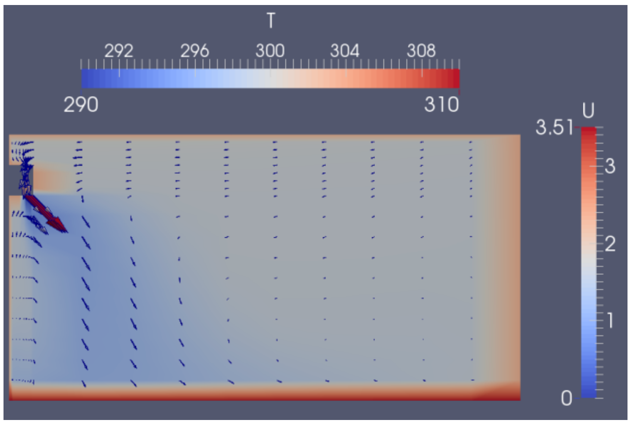
The pull-down transients in the low airflow mode are given in Fig. 6. The pull-down time of the low airflow mode is 1030 sec, 1090 sec and 1440 sec for vane angles of 15°, 45° and 75°, respectively. In the low airflow mode, the jet flow from the indoor unit has a low momentum and is not affected by the buoyancy easily. Therefore, mixing effect is not as strong as in the high airflow case. Low and medium angles yield shorter pull-down time and higher COP than the high vane angle case because the travelling distance of the jet flow is shorter.

The pull-down transients using the mixed air room model (Wetter et al., 2014) are also given in Figs. 5 and 6. Apparently, the mixed air room model leads to much longer pull-down time and exhibits more damping than the CFD room model. This is because the mixed air model ignores the non-uniformity in the temperature field and assumes the all the heat is picked up by the jet flow, resulting in the larger load on the compressor and higher return air temperature as well as longer pull-down time. This suggests that the well-mixed air model might not be suited for control design and therefore a detailed CFD model is more favorable.

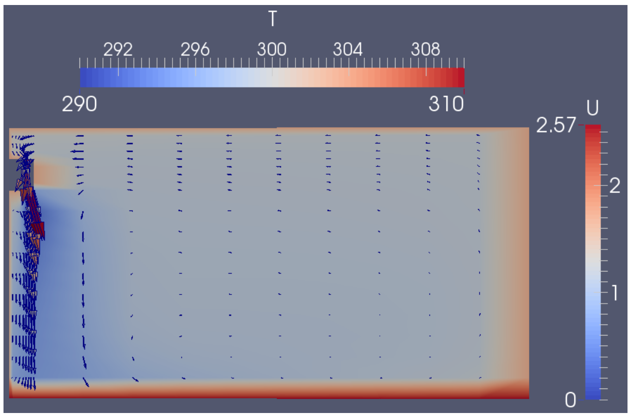
Figs. 7 and 8 show the temperature and airflow distributions of the middle plane M-M of the room when the return air temperature reaches the set point in the high and low airflow modes, respectively. The figures illustrate that air near the inlet has the lowest temperatures, whereas air near the walls and floor and has the highest temperatures. The air temperatures clearly are not uniformly distributed as assumed by the well-mixed air model.



(a)

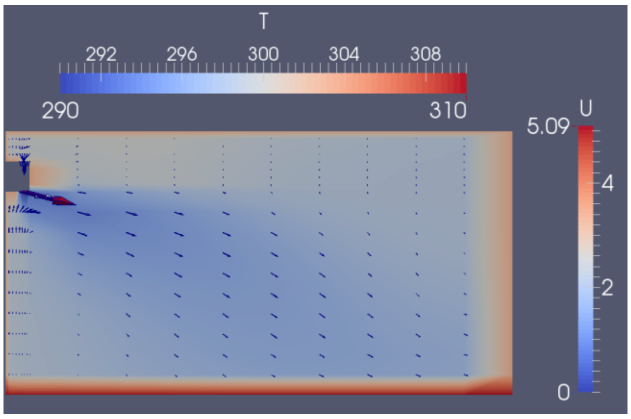


(b)

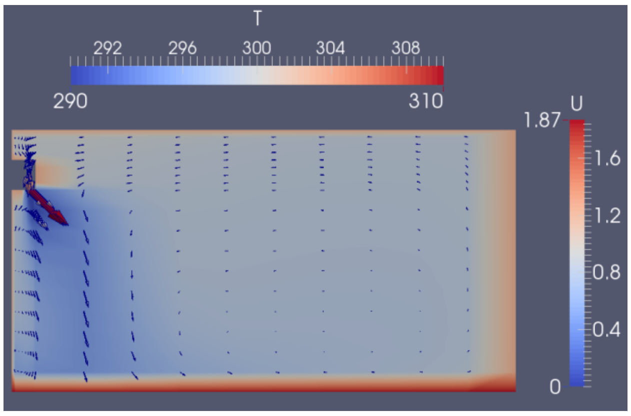


(c)

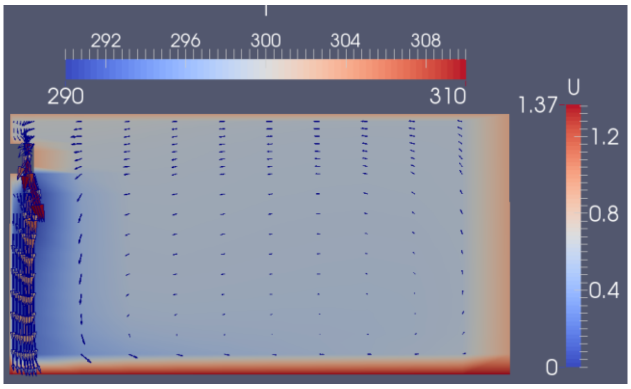
Figure 7. Temperature and airflow distributions in high airflow mode: (a) - 15° vane angle; (b) - 45° vane angle; (c) - 75° vane angle.



(a)



(b)



(c)

Figure 8. Temperature and airflow distributions in low airflow mode: (a) - 15° vane angle; (b) - 45° vane angle; (c) - 75° vane angle.

5 Conclusions

This paper demonstrated a coupled simulation of Modelica and OpenFOAM for the transient modeling of a wall-mounted split-type room air conditioner during pull-down operation. The use of coupled simulation with detailed CFD model for indoor environment facilitates more accurate exploration of system dynamics than using the well-mixed air model due to the inherent non-uniform air flow and temperature distributions in buildings. Numerical simulations indicated that the vane angle and airflow mode showed pronounced impact on the pull-down performance of air-conditioning system. Future work will include explore the effect of location of sensor and heat source as well as experimental validation.

References

Zhiqiang Zhai, Qingyan Chen, Philip Havesb and Joseph H. Klems. On Approaches to Couple Energy Simulation and Computational Fluid Dynamics Programs. *Buildings and Environment*, No 37, pp. 857-864, 2002. doi: /10.1016/S0360-1323(02)00054-9.

Wei Tian, Xu Han, Wangda Zuo and Michael D. Sohn. Building Energy Simulation Coupled with CFD for Indoor Environment: A Critical Review and Recent Applications. *Energy and Buildings*, No 165, pp. 184-199, 2018. doi: 10.1016/j.enbuild.2018.01.046.

Wangda Zuo, Michael Wetter, Wei Tian, Dan Li, Mingang Jin and Qingyan Chen. Coupling Indoor Airflow, HVAC, Control and Building Envelope Heat Transfer in the Modelica Buildings Library. *Building Performance Simulation*, No 9, pp. 366-381. 2016. doi: 10.1080/19401493.2015.1062557.

OpenFOAM Foundation. https://github.com/OpenFOAM/ /OpenFOAM-dev.

Joost J. Brasz and Kenneth Koenig. Numerical methods for the transient behavior of two-phase flow heat transfer in evaporators and condensers. *Numerical Properties and Methodologies in Heat Transfer*, pp: 461-476, 1983.

Hongtao Qiao, Vikrant Aute and Reinhard Radermacher. Transient Modeling of a Flash Tank Vapor Injection Heat Pump System - Part I: Model Development. *International Journal of Refrigeration*, No 49, pp.169–182, 2015. doi: 10.1016/j.ijrefrig.2014.06.019.

Salomon Levy. Forced Convection Subcooled Boiling Prediction of Vapor Volumetric Fraction. *International Journal of Heat and Mass Transfer*, No 10, pp: 951-965, 1967. doi: 10.1016/0017-9310(67)90071-3.

Sihwan Lee, Juyoun Lee and Shinsuke Kato. Influence of Vane Angle on the Effectiveness of Air Conditioning of Wall-mounted Split-type Air Conditioners in Residential Buildings. *Science and Technology in the Built Environment*, No 23, pp. 761-775. 2017. doi: 10.1080/23744731.2016.1260410.

Michael Wetter, Wangda Zuo, Thierry S. Nouidui and Xiufeng Pang. Modelica Buildings Library. *Building Performance Simulation*, No. 7, pp: 253-270, 2014. doi: 10.1080/19401493.2013.765506.

Session 3B: Language

Modelica language extensions for practical non-monotonic modelling: on the need for *selective* model extension

Christoff Bürger[1]

[1] Dassault Systèmes AB, Sweden, Christoff.BUeRGER@3ds.com

Abstract

A Modelica language extension for structural non-monotonic model variation is presented. It enables *selective* model extension: the well-defined refinement of models by deselecting components and connections not of interest or inappropriate for a new design. The need for such variations is explained by the example of Modelica Synchronous, whose adaptation is suffering from crosscutting synchronous decompositions that cannot be *anticipated* when continuous models are designed; instead, contradicting model structure has to be removed when an *actual* sampling is desired. Besides synchronous, further applications for selective model extension are investigated using our prototype implementation in Dymola.

Keywords: Modelica, model variation, synchronous

1 Introduction

Of key importance for Modelica is model variation support, enabling simulation of design alternatives and their step-wise refinement from idealistic prototypes to physically-detailed solutions. To that end, Modelica provides many different abstraction and variation techniques, like model extension, replaceable components, parameters and component modifications.

Having a strong heritage from object-oriented programming however, Modelica's model variation constructs are monotonic with respect to model structure because components, connections or equations can only be added but not removed when extending models. An unfortunately overlooked consequence of flatting is however, that such a structural-monotonic type-strictness, as known from class inheritance in traditional strongly typed object-oriented programming languages like Java or C++, is not required in Modelica. In Modelica, models are flattened before simulation. Flattening essentially reduces the design space of a set of models to a fixed number of instances according to a given parameterization and replaces the resulting instances with their corresponding fixed equation system (Modelica Association, 2017). The difference to traditional strongly typed object-oriented programming is striking: all instances are known before runtime, such that they can be statically constructed. There exists no *runtime* control-flow in Modelica that may cause different instantiations of entities; dynamic dispatch is not required, ultimately neglecting object-oriented polymorphism and the type-system restrictions that typically come with it (Wegner, 1987; Knudsen 1993)[1]. As a consequence, Modelica's current restriction that sub-models must inherit all components and connections of their base-models when extending – that model extension must be monotonic with respect to model structure – can be dropped.

Leveraging on this observation, the paper presents a new Modelica-language extension for non-monotonic modelling: *selective model extension*. Selective model extension can be used to exclude components and connections in a *well-defined* way from inheritance when extending models. Its semantic can be fully understood in terms of model-diagram edits, such that tools can support a convenient graphical user interface for structure-wise non-preserving model variation. The main contribution of selective model extension therefore is to enable unforeseen structural variability without requiring deliberately prepared base-models.

The paper starts with an evaluation on the need for non-monotonic model variation in Modelica (Section 2). To that end, the application of Modelica Synchronous (Elmqvist *et al*, 2012; Otter *et al*, 2012) to refine continuous models for discrete use-cases is chosen which requires non-monotonic modeling to handle the crosscutting clock-partitions of different synchronous designs. Based on the non-monotonic modeling requirements elaborated throughout that discussion, an exact syntax and semantic for selective model extension is presented (Section 3). A demonstration of general practical modelling-benefits, not only for Modelica Synchronous, follows (Section 4). A prototype implementation in Dymola is used on a sophisticated example taken from the Modelica Standard Library to show how selective model extension enables model-development along the lines of real engineering processes – i.e., in terms of step-wise model variation and adaptation – avoiding model variation inconsistencies and artificial intermediate models without physical meaning.

[1]Object-oriented languages typically require monotony of inheritance to ensure the functionality of entities is well-defined for all usage-contexts, independent of control-flows determining instantiation. If sub-classes could drop base-class functionality – i.e., inheritance could be non-monotonic – runtime errors are possible whenever base-class functionality is called on sub-class objects. Static type-systems enforce monotonic inheritance to avoid such errors in the first place.

2 Motivation: Modelica Synchronous adaptation challenges

This section motivates the need for non-monotonic model variation. As a practical problem the potential of Modelica Synchronous (Elmqvist *et al*, 2012; Otter *et al*, 2012) for *existing* examples of the Modelica Standard Library is investigated. The challenge is to enable use-case driven partial sampling of continuous systems without having to change them and with reasonable adaptation workload. As will be shown, the crosscutting of synchronous decompositions cannot be handled by monotonic model variation however; a refinement-based non-monotonic adaptation is required, giving rational for the *selective* model extension proposed in Section 3.

The modelling problems presented in the following are not Modelica Synchronous specific; they can be generalized as will be shown in Section 4.

2.1 Synchronous potential of the MSL

The Modelica Standard Library 3.2.2 has about 60 existing continuous *example* test-models *with controllers* whose discrete modelling might be of interest (cf. Appendix A for the used selection criteria). A lot of these test-models share the same controller, maybe differently parameterized. For example, the 35 models of `Electrical.PowerConverters.Examples` interesting for synchronous modelling share just five controllers defined in `PowerConverters.ACDC.Control` and `PowerConverters.DCDC.Control`. The remaining 25 test-models are much more heterogeneous however, making each a potentially worthwhile candidate for synchronous adaptation.

2.2 Objective: synchronous adaptation of continuous models via refinement

To adapt 60 test-models for synchronous is a major effort, in particular coordinating so many authors from different engineering domains. Involvement of the original authors therefore should be minimized and mostly only be required to ensure that the controllers of the existing continuous test-models are relevant for sampling from a domain perspective. After all, the existing test-models as such – their purely continuous modelling – are mature and useful.

To that end, sampling of their controllers should be an independent task, not requiring changing the original models. Instead, samplings should be introduced in terms of derived test-models that only add discrete partitions, i.e., by refinements adding samples, holds and clocks with respect to the components of an existing model. Such derived tests-models would be partially-discrete instances of their continuous originals, ultimately enabling validation and investigation of *different* samplings.

The original test-models would stay unchanged and cannot be corrupted by synchronous adaptation errors; their correctness is assured from previous model reviews and testing. Code duplication and inconsistencies are avoided and upcoming library changes eased. Ideally, future changes of a continuous model are either automatically incorporated in its derived partially-discrete models (in case the structural interface between continuous and discrete parts is not influenced, i.e., there are no new controller inputs or outputs), or result in translation errors of its derived partially-discrete models (denoting that the controller interface changed and samplings must be adapted).

To support such an iterative development process with seamless and incremental design from a continuous whole system model to different partially discrete variations via model-refinement is of uttermost importance for the success of Modelica Synchronous; it enables the incorporation and automatic change propagation of late continuous *and* discrete design changes and would be a distinctive Modelica feature compared to common block diagram based languages for *causal*-modelling of controllers.

2.3 Problem: monotony of model extension

To derive a partially-discrete model by sampling parts of an existing continuous model requires the introduction of samples, holds, clocks and their respective connections such that the derived model has a *consistent* clock partitioning. Modelica's existing model extension via **`extends`** is sufficient to add all required synchronization components. The derived model can also add the connections combining the sample and hold operators of the intended discrete model partitions with the model parts remaining continuous. The resulting derived model is *inconsistent* however, because it comprises all components of the original continuous model, particularly the old connections bypassing the sample- and hold-interface just introduced; clock-partitioning of the derived model fails due to the structural singularities resulting from having contradicting sampled and non-sampled connections. Since model extension via **`extends`** can only add components, modify the value of inherited components or exchange components deliberately prepared for variability via **`replaceable`**, it is impossible to fix clock-partitioning errors due to inherited connections and therefore *consistently* incorporate samplings.

2.4 Problem: prescient modelling

To enable sampling via model extension, parametrization or modification requires *deliberate preparations* of model parts that might become subject to sampling. For example, models could be prepared for sampling by pushing the parts constituting controllers into separate models and referencing them as replaceable components well-suited for modification or by using conditional declarations instantiating a continuous or discrete design depending on parameterization. Such workarounds are in conflict with our objectives however, as they anticipate *specific* samplings before their actual need, implying changes of the original model when the need for a new specific sampling *actually* arises. Deliberate preparations of models to enable future samplings naturally only enable the prepared samplings, i.e., specific discrete use-cases. To anticipate all possible samplings of continuous model parts that might be of interest in future discrete use-cases is impractical however.

2.5 Problem: crosscutting synchronous decompositions

The design of controllers significantly varies depending on available sensor information (varying control input signals) and control-task splitting (varying model parts constituting controllers, for example due to independent asynchronous control vs. synchronous cascade-control with different sub- or super-samplings). Typically, many reasonable synchronous designs exist, each resulting in a specific clock partitioning. The clock partitions of *different* synchronous designs are likely in conflict however.

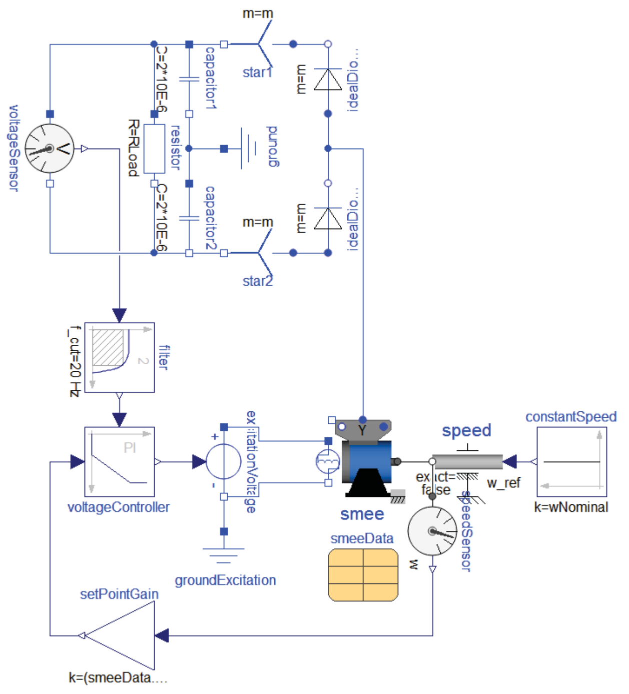


Figure 1. Induction machine with voltage controller.

Consider for example the electrical excited synchronous induction machine of the Modelica Standard Library presented in Figure 1 (`Electrical.Machines.Examples.SynchronousInductionMachines.SMEE_Rectifier`). Five different synchronous designs immediately come to mind for its voltage controller: (1) a fat controller, comprising not only the gain and PI controller but also filter, (2) a design with the filter being independent, either as (2.1) a separate asynchronous sampled system or (2.2) not sampled at all and (3) a cascade control, with the filter being (3.1) sub-sampled, in case set point changes are more critical than filtering the current voltage, or (3.2) super-sampled, in case the filter implementation requires higher sample rates than the rest of the controller. The clock partitions of all five variants are in mutual conflict although each, in itself, is sound.

Important for our investigation is that Modelica models already have a dominant decomposition with respect to their component hierarchy (network of interconnected hierarchical components); and it is that very hierarchy in whose terms model variation is defined using parametrizations, modifications and re-declarations, whereas model extension always preserves it. Clock partitioning however is about decomposing a model according to its differently clocked parts. Thus, even if a model's structure is aligned with *some* future synchronous design, it will be in conflict with *other* designs. Clock partitioning crosscuts the natural composition of physical systems as hierarchical component networks[2].

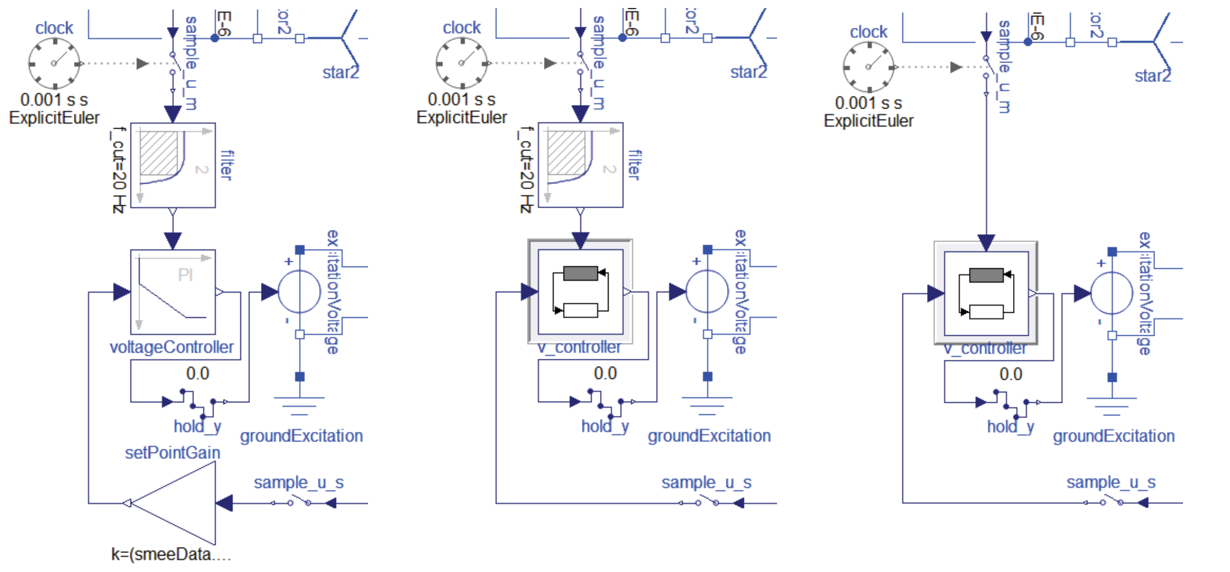
2.6 Solution: non-monotonic extension

To incorporate a specific sampling into an existing model means to modify its component network according to the sampling's crosscut, i.e., to change the model's *structure* at the intersection points of clock partitions and further control-design adaptations. Intersection points of clock partitions correspond to connections that must be removed; instead respective samples and holds are added, connecting the clock partitions. Control-design adaptations usually correspond to components that have to be removed because the new control-design is structural different duo to changed sensor and actuator usage (for example the filter and gain of Figure 1 may not be required by a third party library controller). All such changes are well-defined by removing connections and components that are superfluous and replaced by the intended synchronous design. The required refinement can be defined as ordinary model extension with parts of the original model excluded from inheritance, ultimately enabling structural non-monotonic changes.

3 Selective model extension proposal

This section presents a concrete proposal to enable non-monotonic modelling in Modelica. The proposed selective model extension enables the deselection of

[2]Implementation techniques for system parts cross-cutting a dominant component hierarchy are subject of aspect-oriented programming (Kiczales *et al*, 1997; Tarr 1999). Particularly object-oriented programming language extensions enabling crosscutting implementation are well-investigated. The proposed selective model extension can be seen in that tradition; it is Modelica-specific however, since it depends on static instantiation via model-flattening as explained in Section 1.



(a) simple sampling **(b)** off-the-shelf controller **(c)** off-the-shelf dampening-controller

Figure 2. Three control scenarios for the induction machine of Figure 1 (controller only excerpts).

connections and components when extending models. To start with, a simple sampling example sketches the new language concepts. The definition of actual syntax and semantic follows.

3.1 Selective model extension example

Consider again the controlled induction machine presented in Figure 1. Figure 2 presents three different synchronous control designs for it, implemented in the following using selective model extension to handle their structural non-monotonic variation.

(a) Simple sampling scenario: A straightforward synchronous control design is to just sample the control components of the original example. To that end, the connections between `voltageSensor` and `filter`, `speedSensor` and `setPointGain` and `voltageController` and `excitationVoltage` have to be replaced with likewise-connected samples and holds. Using selective model extension, the implementation of Figure 2 (a) is:

```
model SMEE_Rectifier_Sampled
  extends SMEE_Rectifier;
/* PART 1: Drop "outdated" continuous parts. */
  // Exclude connections from inheritance:
  for each extends
    break connect(voltageSensor.v, filter.u);
    break connect(speedSensor.w,
                  setPointGain.u);
    break connect(voltageController.y,
                  excitationVoltage.v);
  end for each extends;
/* PART 2: Introduce sampling. */
  // Introduce clock, samples and hold and…
  PeriodicRealClock clock(
    period = 0.001,
    useSolver = true);
  SampleClocked sample_u_m;
  Sample sample_u_s;
  Hold hold_y;
equation
  // …connect them:
  connect(clock.y, sample_u_m.clock);
  connect(voltageSensor.v, sample_u_m.u);
  connect(sample_u_m.y, filter.u);
  connect(speedSensor.w, sample_u_s.u);
  connect(sample_u_s.y, setPointGain.u);
  connect(voltageController.y, hold_y.u);
  connect(hold_y.y, excitationVoltage.v);
end SMEE_Rectifier_Sampled;
```

The new sampling-related components – the clock, sample and hold and their connections with inherited components – are introduced as used to and are subject to normal Modelica 3.4 semantic (Part 2). Also syntax and semantic of the `extends` clause are as used to, except that the `for each extends` block modifies the set of features the `extends` clause defines to be inherited (Part 1). Each `break connect` clause within a `for each extends` block removes the respective connection from the set of features the model inherits. Note the plural form *clauses*, implying that *`for each extends`* modifies *all* `extends` clauses of a model. In our case, just the extension from `SMEE_Rectifier` is modified, excluding the ingoing connections of `filter` and `setPointGain` and the outgoing connection of `voltageController` from inheritance; the deselections are `break connect(voltageSensor.v,filter.u)`, `break connect(speedSensor.w,setPointGain.u)` and `break connect(voltageController.y,excitationVoltage.v)`.

(b) Off-the-shelf controller scenario: Another reasonable design is to use an off-the-shelf controller provided by a specialized library, as shown in Figure 2 (b). To that end, the original control *components* have to be replaced. Note the plural; not a single `replaceable` component is changed, but the complete component network constituting the controller. Assuming the new controller still requires the filtering of voltage, only `voltageController` and `setPointGain` have to be removed. Using a selective model extension of scenario (a), the implementation of Figure 2 (b) is:

```
model SMEE_Rectifier_ExternalController
  extends SMEE_Rectifier_Sampled;
// Remove original controller with connections:
  for each extends
    break voltageController;
    break setPointGain;
  end for each extends;
// Introduce the new controller…
  replaceable ExternalController v_controller;
equation
// …and connect it:
  connect(filter.y, v_controller.u_m);
  connect(v_controller.y, hold_y.u);
  connect(sample_u_s.y, v_controller.u_s);
end SMEE_Rectifier_ExternalController;
```

The original controller and gain are excluded from inheritance via `break voltageController` and `break setPointGain`. Deselecting a component automatically deselects all its connections. Thus, the only thing to do besides deselecting the original control components is to integrate the new controller reusing the sampling inherited from scenario (a).

(c) Off-the-shelf dampening-controller scenario: Finally, an off-the-shelf controller with specialized dampening of its input voltage can be used as shown in Figure 2 (c). In that case, the original filter is not required. The implementation based on scenario (b) is:

```
model SMEE_Rectifier_DampeningExternalController
  extends SMEE_Rectifier_ExternalController(
    redeclare DampeningController v_controller);
  for each extends
    break filter;
  end for each extends;
equation
  connect(sample_u_m.y, v_controller.u_m);
end SMEE_Rectifier_DampeningExternalController;
```

Conclusions: Scenarios (a) to (c) demonstrated the consecutive synchronous adaptation of a continuous model. Each refinement step required structural non-monotonic changes in terms of removing superfluous connections and components inappropriate for a more sophisticated control design. Using selective model extension, the respective synchronous adaptations are possible without changing the original continuous model, ensuring configuration consistency of the controlled system when comparing the synchronous designs one another. Also diagrammatic consistency is improved; after all, the diagrams of Figure 2 are derived from Figure 1 by normal `extends` semantic and our Dymola implementation of deselections.

3.2 Syntax: selective extension clauses and inheritance modifications

Selective model extension as presented in Section 3.1 requires rule-additions to Modelica's context-free grammar. The changes required are very limited however. Only an additional alternative for `element` (cf. Appendix "B.2 Grammar" of the Modelica 3.4 specification) has to be added:

```
element :
  import-clause |
  extends-clause |
  selective-extension-clause | // new
  [ redeclare ]
  [ final ]
  [ inner ] [ outer ]
  ( (class-definition | component-clause) |
    replaceable (
      class-definition | component-clause)
    [constraining-clause comment])
```

whereas `selective-extension-clause` is:

```
selective-extension-clause :
  for each extends
  { inheritance-modification ";" }
  end for each extends
```

and `inheritance-modification` is:

```
inheritance-modification :
  break connect-clause | // Connection and…
  break IDENT // …component deselection.
```

with `connect-clause` and `IDENT` already well-defined in the specification. No new keywords are introduced. The new context-free derivations `for each extends`, `break connect` and `break IDENT` are syntax errors in current Modelica. As a consequence, the proposed selective model extension never changes the semantic of existing valid Modelica 3.4 models. Models that are syntactically invalid could theoretically become valid however, but chances are extremely low[3].

3.3 Semantic: terminology, well-formedness and interpretation

An important criterion of selective model extension is to ensure applications are meaningful. A selective extension is meaningful when *all* its modifications of the set of inherited elements are unambiguous and applicable, in which case it is called well-formed. Selective extensions that are not well-formed are modelling errors; they are meaningless, i.e., without unique interpretation defining the result of their application. The rest of this section defines well-formedness and interpretation for the proposed syntax.

3.3.1 Terminology

To ease further discussion, we define the following terms (words embraced by parenthesis are optional, only improving readability; the term "if X is evident" denotes "if X is already well-defined by context (i.e., specific) or not of particular interest (i.e., generic)"; the term "derivation" denotes a context-free derivation according to the syntax specified in Section 3.2):

(a) Context of selective extensions: A `selective-extension-clause` derivation S within a model A with arbitrary many `extends` clauses E_1, ... E_n, that extend

[3]It is not likely that a syntax error happens to satisfy the proposed syntax. It is even less likely the respective model will further satisfy the semantic constraints explained in Section 3.3.2; and, due to deselections, it is close to impossible that it is valid considering existing Modelica well-formedness constraints like *"referenced components must be declared"* and *"the system of equations must be well-defined"*.

models $M_1, \dots M_n$ respectively, is called a "selective (model) extension of $M_1, \dots M_n$"; if $M_1, \dots M_n$ are evident, just "selective (model) extension". $E_1, \dots E_n$ are called "(local) extends-clauses of A". S is called "extends-modification of A". If A is evident, we just speak of "local extends-clauses" and "extends-modification". We say "S is local to A", "$E_1, \dots E_n$ are local to A", "$E_1, \dots E_n$ are local to S" and vice-versa; and we call an "α local to β" a "(local) α of β" and vice-versa. We further say "S modifies $E_1, \dots E_n$" and "A selectively extends $M_1, \dots M_n$ with respect to S". If $M_1, \dots M_n$ or S are evident, we just say "A selectively extends". This terms are called "context of S"; if S is evident, just "selective extension context".

(b) Context of deselections: A selective extension S is a block; its body consists of the `inheritance-modification` derivations $m_1, \dots m_n$ applied throughout the derivation of S. Each `inheritance-modification` derivation is called a "deselection of S"; if S is evident, just "deselection". The set $m_1, \dots m_n$ are the "deselections of S"; if S is evident, just "deselections".

We distinguish two types of deselection:

(b.1) `break` `connect-clause` derivations are called "connection deselection"

(b.2) `break` `IDENT` derivations are called "component deselection"

If a deselection type is evident, we just say "element" instead of connection or component.

The subset of connection deselections of the deselections of S are called "connection deselections of S"; if S is evident, just "connection deselections". The deselections of S that are not connection deselections are called "component deselections of S"; if S is evident, just "component deselections".

The relations defined for S in (a) – like extends-clauses, extends-modification, modifies etc. – also hold for the deselections of S. We therefore can speak of the context of a deselection, defined by the model it is local to and the local extends clauses it modifies; and it is true by definition that *"deselections are extends-modifications of their local model and models selectively extend with respect to their deselections"*.

(c) Extent of selective extensions: Let $I_{extends}$ be the set of elements the local extends clauses of a model A define to be inherited; let connection elements be represented by their respective `connect-clause` derivations in $I_{extends}$ and components by `IDENT` derivations, i.e., their name. We call $I_{extends}$ the "preselective-extent of A". Let $D_{connection}$ be the set of connection deselections of A and $D_{component}$ the set of component deselections. We call two connections `connect(a1, b1)` and `connect(a2, b2)` "matching" if either `a1 = a2` $\wedge$ `b1 = b2` or `a1 = b2` $\wedge$ `b1 = a2`.

We call an inherited connection $c_i \in I_{extends}$ "extent of a (connection) deselection" $d =$ `break` $c_d \in D_{connection}$ if c_d and c_i are matching and say "c_i is deselected due to d" and "d deselects c_i"; if d is evident we just say "c_i is deselected", and if c_i is evident we just speak of a "deselected T (connection)" whereas T is the connector type of c_i. If also T is evident, we just speak of a "deselected connection".

We call an inherited component $c_i \in I_{extends}$ "extent of a (component) deselection" $d =$ `break` $c_d \in D_{component}$ if $c_d = c_i$ and say "c_i is deselected due to d" and "d deselects c_i"; if d is evident we just say "c_i is deselected", and if c_i is evident we just speak of a "deselected T (component)" whereas T is the component type of c_i. If also T is evident, we just speak of a "deselected component".

Let c_i be a component deselected due to a deselection d. We call the set $D = c_i \cup \{c \in I_{extends} \mid c \textit{ is connection of } c_i\}$ the "transitive-extent of d"; if d is evident, we just speak of a "transitive-extent". For each $c \in D \setminus c_i$ we say "c is indirectly-deselected due to d"; if d is evident, we just say "c is indirectly-deselected" and, if c is also evident, we speak of an "indirectly-deselected connection". Indirectly-deselected connections are deselected connections. The transitive-extent of a connection deselection is just its extent.

We call the union of the transitive-extents of the deselections of A the "deselective-extent of A". Let $I_{deselected}$ be the deselective-extent of A; we call the set $I_{selected} = I_{extends} \setminus I_{deselected}$ "selective-extent of A"; if A is evident, we just speak of "preselective-", "deselective-" and "selective-extent".

Colloquial usage: Whenever we emphasize the act of *modelling* via introducing deselections for an element or element type E, we use the term "deselection of E" or "deselecting E"; if E is evident, we just speak of "deselecting". Likewise, we speak of "selection of E" and "selecting E" for removing, or deliberately not introducing, deselections for E.

3.3.2 Well-formedness

Five well-formedness constraints are proposed for selective model extension. The following list also gives a short rational for each constraint:

Constraint (1) Selective model extensions must be element of a model or block (i.e., the enclosing scope of a `selective-extension-clause` must be a `class-definition` whose `class-prefixes` are derived to `model`, `block`, `partial model`, or `partial block`; cf. Appendix B.2.2 of the Modelica 3.4 specification).

Rational: Connector classes are prohibited to use selective extension because their whole purpose is to define common interfaces; deselection of connector-components would essentially make the derived connectors incompatible. Types are excluded for similar reasons. Records are excluded to avoid runtime errors due to instances queried for deselected fields. Packages are excluded because they are used to define a modelling environment with well-defined features; to reduce availability of provided features contradicts their purpose in the first place. Deselection of function in- and out-puts is prohibited, because call-sites depend on the applicability of a function's interface.

Constraint (2) The extent of deselections is not empty (i.e., for each deselection exists a local extends clause that inherits the deselected element).

Rational: Selective extensions should be meaningful, i.e., each of their deselections should be applicable. The constraint also makes it impossible to deselect beforehand, eliminating the risk that sub-models accidentally miss future base-model improvements.

Constraint (3) Deselected components are not modified by local extends-clauses.

Rational: Modifying a component and deselecting it within the same model hints at a modelling error.

Constraint (4) Deselected elements are not `final`.[4]

Rational: `final` is deliberately introduced by developers to prevent common model-configuration errors due to further modifications; this naturally encompasses modifications changing the *existence* of `final` modified components. The constraint also prevents the reintroduction of deselected `final` components as non-finals.

Constraint (5) Models have at most a single selective extension clause.

Rational: Since selective extension modifies all local extends-clauses, it makes sense to collect all deselections of a model within a single `for each extends` block. Doing so avoids scattering of inheritance modifications, ultimately increasing readability of models.

These constraints can be checked just considering the set of connections and components inherited due to *local* extends-clauses; there is no need to mutually compare the individual sets. Details, how inherited elements are defined, particularly if base-models apply other selective extensions, are not required.

Note that the proposed well-formedness constrains do not prohibit extending models from reintroducing components deselected. This is useful to solve currently non-manageable multiple-inheritance conflicts due to structural differences of base-models.

No further restrictions regarding the well-formedness of equations are proposed. Deselection of connections can result in structural non-singular equation systems however; likewise deselected components may result in base-model equations with unresolved references. The fallback on default equation well-formedness is important. It ensures the context of selective extensions is sound; in practice, this means that the "structural-holes" due to deselections must be properly fixed and invalidating base-model changes are caught. Note that speaking of "structural-holes" is reasonable, considering deselections are defined with respect to diagram-wise clearly distinguishable model parts; selective model extension is about the removal of interconnected component networks. To accidentally change model semantic not visible within the diagram layer, like non-connection equations, is impossible. As a consequence, deselection can be realized as graphical edit-operations in the diagram layer of Modelica tools.

[4]Constraint (4) does *not* prohibit the deselection of components containing `final` elements, as long as the deselected component itself is not `final`.

3.3.3 Interpretation

Given the terminology of Section 3.3.1 and the well-formedness constraints of Section 3.3.2, defining an interpretation for well-formed selective model extensions is straightforward.

The objective of a selective extension is to exclude elements from inheritance. To that end, one has to take care of – colloquial speaking – three kinds of inherited elements: (1) the elements inherited by a model's local `extends` clauses (i.e., inheritance as used to from Modelica 3.4), (2) the elements excluded from this set and (3) the resulting actually inherited elements. With respect to Section 3.3.1, these sets are the preselective-, deselective- and selective-extent.

The extent definitions of Section 3.3.1 are constructive; first, the preselective-extent is derived, based on it the deselective-extent, finally followed by the selective-extent. Note that, the extents can be empty for the definitions to hold. The deselective-extent of a model without a selective extension is the empty set; such a model's selective-extent just is its preselective. This characteristic significantly limits the changes required in the existing Modelica specification to incorporate selective model extension.

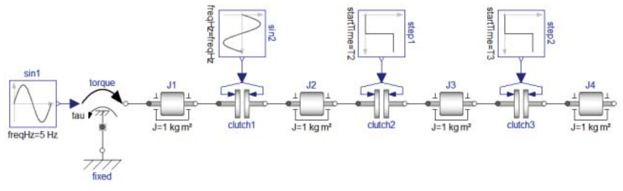
In the end, the interpretation of selective model extension just boils down to the addition of the terminology introduced in Section 3.3.1 and a single modification of the Modelica 3.4 specification regarding the definition of inherited elements; the new definition is: *"the inherited elements of a model are its selective-extent"*.

4 Advanced application scenarios

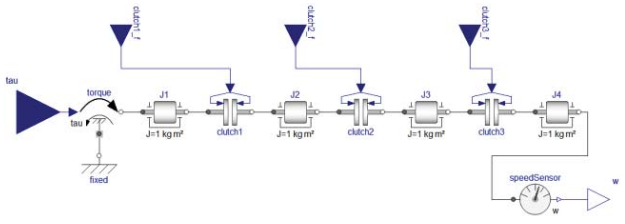
The applications of selective model extension presented so far are all in the domain of Modelica Synchronous. In the following, further applications, with the focus on general advantages for modelling from an engineering perspective, are investigated. To that end, selective extension is used to redesign an existing example scenario of the Modelica Standard Library; doing so will reveal implementation-shortcomings of the example and how non-monotonic modeling can be used to avoid them. First however, another important use case for selective model extension is presented: to adapt whole system models for further external or component usage.

4.1 Component extraction example

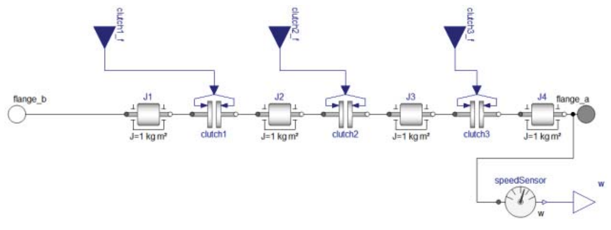
Figure 3 presents the well-known coupled clutches example of the Modelica Standard Library (`Mechanics.Rotational.Examples.CoupledClutches`) and two variations of it. The first variant prepares its export as Functional Mockup Unit (FMU) that can be distributed to third parties for simulation in non-Modelica contexts (Modelica Association, 2014); the second variant prepares the coupled clutches for usage as component within further Modelica models. In both cases, the fixed input stimuli of the original example must be removed and replaced by respective input connectors. The normal forces of the clutches just become real inputs; the in- and outputs of the inertias depend on usage however.



(a) original coupled clutches



(b) extracted FMU model based on (a)



(c) extracted component model based on (b)

Figure 3. Coupled clutches FMU/component extraction.

For FMU simulation, the input `tau` for the first inertia is a torque and the output of the simulation is the absolute angular velocity `w` of the fourth inertia `J4`; thus, both are just real values. For component usage however, one would like to stay with the flange interface of the Modelica Standard Library for the in- and output of the first and last inertias. Doing so ensures proper flow-derivation of the cut-torques[5].

[5]Because flow-variables – and therefore Modelica-like automatic flow-value derivation – are not supported in the FMI 2.0 standard, users of the FMU-component of Figure 3 (b) have to be careful that torques are correctly modeled in application contexts of the FMU; in that sense the FMU-component is less flexible compared to the Modelica-component of Figure 3 (c). On the other hand, the flow-variables of the flange-interface are the reason why the Modelica-component is unsuitable for FMU-export and usage in external simulations.

Both adaptations are straightforward using selective extensions. For FMU extraction the implementation is

```
extends …Rotational.Examples.CoupledClutches;
for each extends
  break sin1;break sin2;break step1;break step2;
end for each extends;
```

accompanied by introducing and connecting the in- and output normal forces, torque and velocity as shown in Figure 3 (b). When extracting a component however, the torque adapter becomes superfluous since a proper flange input will be provided. In terms of the FMU model, the respective selective extension is

```
extends CoupledClutches_FMU;
for each extends
  break torque; break fixed; break tau;
end for each extends;
```

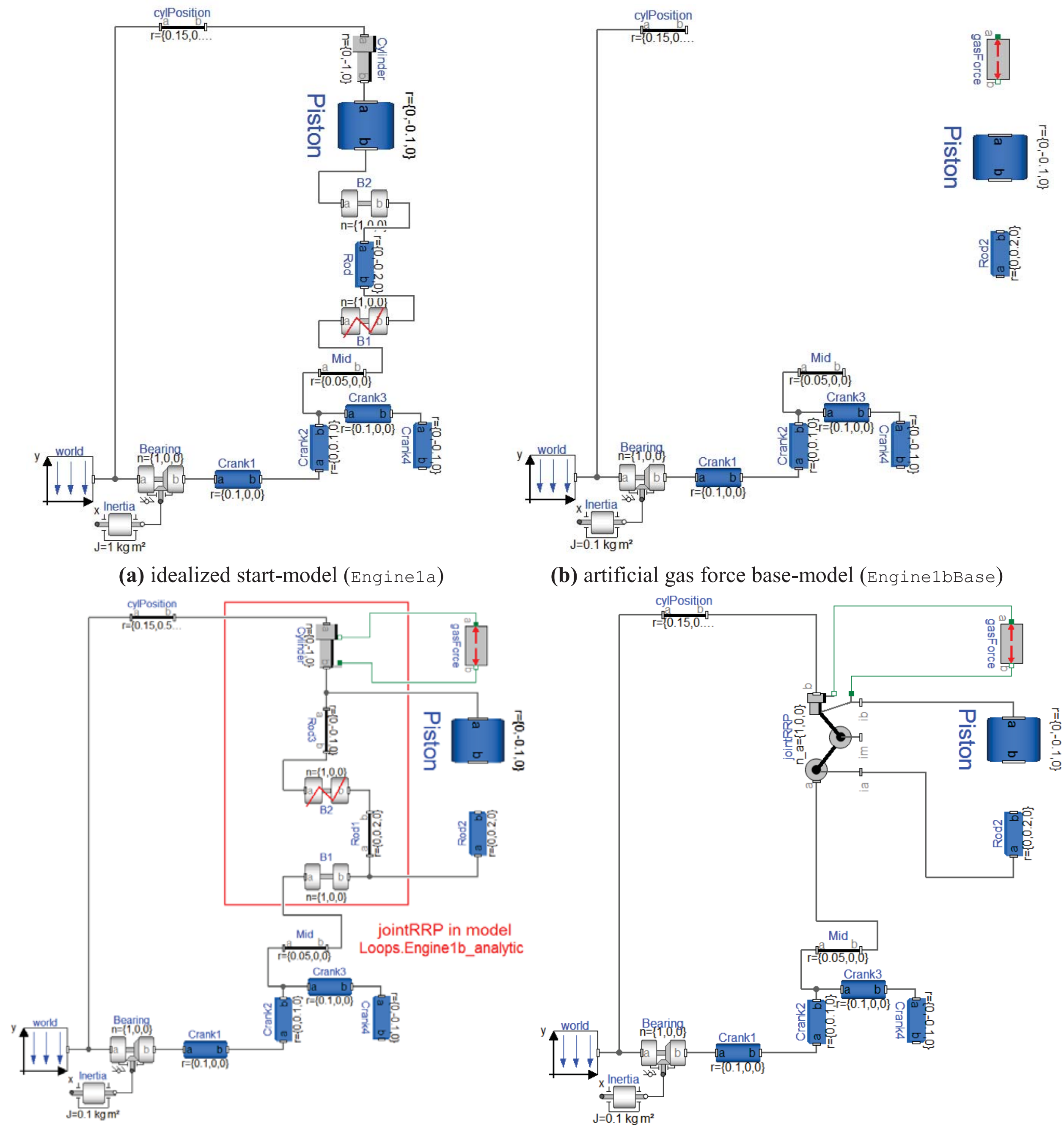
this time accompanied by introducing and connecting the in- and output flanges as shown in Figure 3 (c).

In conclusion, selective model extension enables to extract a component model from a whole system model and incorporate the usage-interfaces of future application contexts. Doing so we *know* the extracted component is working; after all it comes from a well-tested, whole system model with proper simulation that has just been lifted to a component on demand.

4.2 Domain-driven refinement example

Our final selective extension scenario is the step-wise design of a one cylinder engine, as exemplified by the `Engine1a`, `Engine1b` and `Engine1b_analytic` models in package `Mechanics.MultiBody.Examples.Loops` of the Modelica Standard Library. The basic idea is to design a final analytic engine model starting from an idealized model via one considering the gas force in the cylinder. Figure 4 summarizes the current solution of the standard library. There are several problems with it, all due to the lack of non-monotonic modeling means.

The most obvious inconsistency is that the models incorporating the cylinder's gas force (`Engine1b` and `Engine1b_analytic`) do not inherit from the idealized base model (`Engine1a`), but from a completely independent new **`partial`** model (`Engine1bBase`). The reason can be only understood by an investigation *starting* from the *final* analytic model: it introduces the `jointRPP` component, which encapsulates an analytic solution for original engine components. Thus, the final solution cannot extend the idealized start-design because it has to replace parts of the start-design's component network with something whose incremental design *is* the actual task. `Engine1bBase` was introduced to consistently configure *at least* the common components of models considering the gas force of the cylinder. But `Engine1bBase` is completely artificial: it cannot be simulated, its components are hanging in the air and it has nothing in common with the idealized model that was the original starting point for designing the engine. Quiet contrary it is the result of an inversed engineering process, taunting the natural design order.



(a) idealized start-model (`Engine1a`)

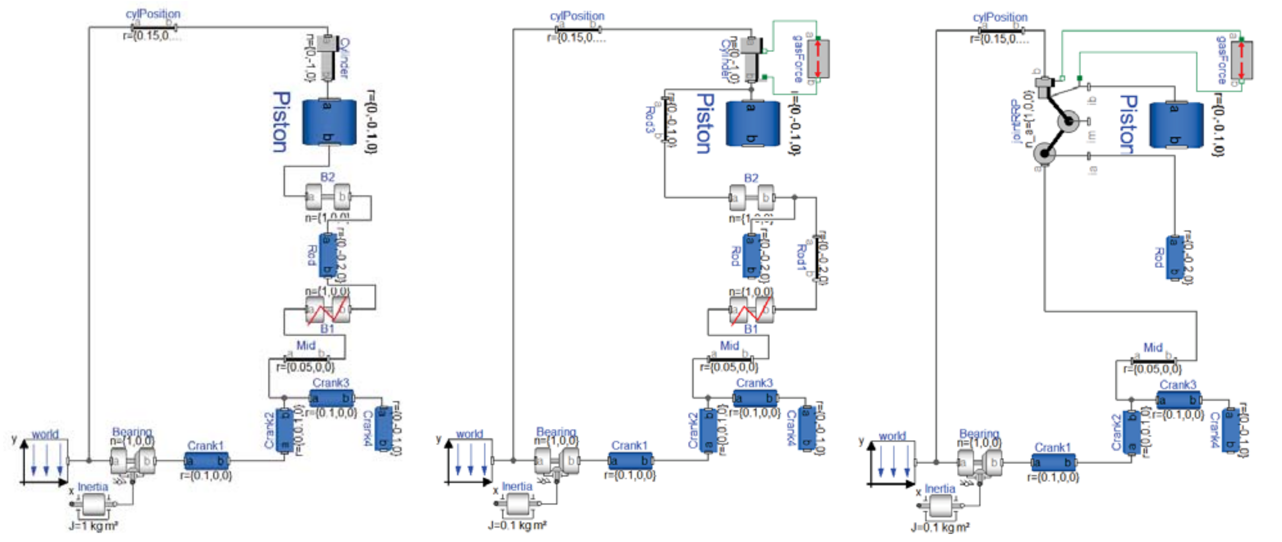
(b) artificial gas force base-model (`Engine1bBase`)

(c) gas force intermediate-model (`Engine1b`)

(d) analytic final-model (`Engine1b_analytic`)

Figure 4. One cylinder engine scenario (current standard library solution).

Since `Engine1bBase` does not extend `Engine1a` – in fact cannot – the obvious question is if the idealized and gas force incorporating models are *at least* consistently configured. This is an important issue because `Engine1bBase` is a partial model-copy of `Engine1a`; it is not obvious if differences are intentional or just copy-and-paste errors that slipped in throughout revisions. As it turns out there is a plethora of configuration differences however. First, the inertias are configured differently; likewise `r` of `cylPosition` is inconsistent. Second, the `a`-connector of the piston is connected with `b` of the cylinder in `Engine1a` but with `Rod3.a` in `Engine1b`. But most confusingly, the bearings `B1` and `B2` are switched in `Engine1b` compared to `Engine1a`. This is a tricky change to comprehend, since one of the bearings must break the kinematic-loop of the multi-body system; and the question is if turning them was required due to integration or numerical issues. As far as we can say that is not the case; `Engine1b` can be simulated with the bearings turned back without problems in Dymola. To make confusion complete, `Rod.r` and `Rod2/Rod1.r` are inversed between `Engine1a` and `Engine1bBase/Engine1b` (`r = {0, -0.2, 0}` vs. `{0, 0.2, 0}`) and must be turned to be consistent with the bearings switch. Although the sum of changes is correct, they are hard to comprehend. The incremental design of the engine is obscured behind a wall of model copying and modifications.



(a) `Engine1a` (existing MSL solution) **(b)** new `Engine1b` solution **(c)** new `Engine1b_analytic` solution

Figure 5. One cylinder engine scenario (proposed selective extension solution).

The alternative implementation of `Engine1b` and `Engine1b_analytic` based on selective extensions is much clearer. It is shown in Figure 5. Note the increase of diagram consistency; one can clearly see how starting from `Engine1a` the design is step-wise refined. The reason is that each design after the idealized start-model now inherits the diagram of the previous. Another advantage is that intentional modifications are evident. Consider for example the selective extension to implement `Engine1b`:

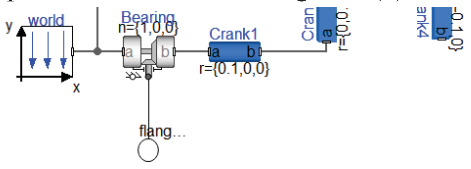
```
extends …MultiBody.Examples.Loops.Engine1a(
    Cylinder(useAxisFlange = true),
    Inertia(
      J = 0.1,
      phi(fixed = true, start = 0.001),
      w(fixed = true, start = 0)),
    cylPosition(r = {0.15,0.55,0}));
for each extends
  break connect(B2.frame_a, Piston.frame_b);
  break connect(B1.frame_b, Rod.frame_b);
  break connect(Rod.frame_a, B2.frame_b);
end for each extends;
// Add Rod3, Rod1 and gasForce and connect them…
```

The configuration differences to `Engine1a` can now be encapsulated in modifications as used to. Also the implementation of `Engine1b_analytic` is straight:

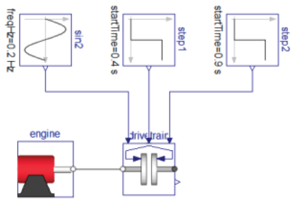
```
extends Engine1b;
for each extends
  break Cylinder;
  break B2; break B1; break Rod1; break Rod3;
end for each extends;
// Add the analytic solution and connect it…
```

The components comprised by the analytic solution are just replaced by it. Altogether, the new solution is much more consistent; changes like the unintended bearings switch and rod turning cannot just slip in.

As final challenge one could lift the engine to a component as shown in Section 4.1. Its fixed inertia, used for "startup", is problematic however. If used as component, an external inertia driven by the engine will be given instead. To that end, the inertia must be replaced by a flange-connector as shown in Figure 6 (a); the resulting engine component can be combined with the coupled clutch component of Section 4.1 to a simple powertrain as shown in Figure 6 (b).



(a) `Engine1b_analytic` component (excerpt)



(b) composition of engine component and clutches

Figure 6. Simple powertrain of engine and clutches.

5 Alternative designs

The proposed selective model extension is just a first step towards non-monotonic modelling in Modelica. Its final definition is open for discussion.

First of all, constraint (5) of Section 3.3.2 might be controversial; instead of a single **for each extends** block, several could be permitted. Deselections could be aligned with the **extends** clauses they deselect elements from. For example, the **extends** clause of Figure 2 (b) could look like (assuming the cut-off frequency of the filter has to be modified as well)

```
extends SMEE_Rectifier_Sampled(
  break voltageController,
  break setPointGain,
  filter(f_cut = 15));
```

Thus, all modifications and deselections regarding a base-model could be grouped with the respective model extension. An advantage of aligning deselections with `extends` clauses is, that only elements of a specific base-model are excluded from inheritance. Of course, this gives rise to the question of consistency in case similarly named elements exist in several base-models; should the deselection of all be enforced or is it fine to deselect only a subset? The proposed semantic of `for each extends` always deselects all elements sharing a name, such that common base-model elements are consistently deselected; selection of a specific namesake requires its deliberate reintroduction, avoiding otherwise easy to miss indirect selections (indirect because the actually selected element is implicitly given by deselecting namesakes).

Another open issue is how fine-grained connections can be deselected. The definition of "matching" in Section 3.3.1 (c) is a very simple equivalence test just comparing the syntactic structure of the component references selecting the connected elements; the proposed semantic always deselects the complete matching connection. Since connectors can be hierarchical structured, including array elements, one could imagine more fine-grained deselections to rearrange parts of a structured base-class connection. Partial deselections of a structured connection could for example unlink only certain of its nested array and component elements. The graphical representation and editing of such deselections would be problematic however, since the structure of connections is not visible in Modelica's current diagram layer design.

Another limitation of the proposed solution is that only base-model elements can be deselected, but not their nested elements. Considering the crosscutting nature of Modelica Synchronous, qualified deselection might be very useful for synchronous adaptation. Like for structured connections however, again diagrammatic presentation and editing of nested component deselections would be problematic.

It is worthwhile to note that a relaxation of `replaceable`, by assuming all components are implicitly declared `replaceable` without type constraints, is insufficient for many cases handled by selective model extension. The problem is that `redeclare` cannot be used to consistently replace a *network* of components, as for example required to integrate the off-the-shelf induction machine controller of Figure 2 (b), where several original components must be removed, including their connections. To remove components in terms of re-declarations also is very cumbersome, not to speak of the consequences for the diagram layer which becomes cluttered with components representing actually removed and therefore not existing model parts that – quiet contrary – should not be shown at all.

Also the idea that all declarations and connections are implicitly conditional looks unsuitable; the parametric referencing for enabling and disabling would be tedious. The proposed selective model extension comprises this approach, just the other way around: instead of declaring everything conditional, it deselects by extension when actually required.

6 Conclusions

Engineering processes are typically not monotonic in terms that everything of an old design is taken when developing a new; some parts may be deliberately excluded and not present in the derived design. In terms of physics modeling in Modelica, such non-monotonic model variations are model-extensions with some original base-model features excluded from inheritance. Unfortunately, Modelica 3.4 is missing convenient means for structural non-monotonic modelling, which is a serious deficit the proposed selective model extension solves. Using selective extensions, no copying, changes or deliberate preparations on models are required to derive well-defined variants not preserving all of the original model structure. The presented concepts suffice to conveniently adapt models – including the examples of the Modelica Standard Library – for different synchronous application scenarios. And as shown in Section 4, selective model extension is also beneficial for a more natural engineering process with refinement-based model variation and adaptation. Artificial intermediate models, without physics simulation meaning, and system variation inconsistencies can be avoided; and non-monotonic interface adaptations required for cross-library integration incorporated. Particularly the latter will likely become an important future challenge, considering the likelihood of interface incompatibilities between libraries tends to increase with the success of the Modelica community and respective growing number of library suppliers. Another promising application area for selective model extension is model testing, particularly systematic fault introduction to simulate non-nominal behavior. The idea is to weave error sources into existing models, like noise-components intercepting a connection. Using selective extensions, the tested models do not have to be specifically prepared for fault injection; system parts can just be removed from inheritance and replaced by faulty – or even mock-up versions using external table-data – to inject misbehavior and configure the environment of error scenarios.

Acknowledgements

I am grateful for the help of Hans Olsson with the implementation of selective model extension in Dymola; his advices regarding the example scenario of Section 4 and proof reading have been very valuable. I also like to thank Hilding Elmqvist for his idea to add FMU extraction as valuable application case.

References

Modelica Association. *Modelica® - a unified object-oriented language for systems modeling: language specification version 3.4*, 2017.

Modelica Association. *Functional mock-up interface for model exchange and co-Simulation*, 2014.

Hilding Elmqvist, Martin Otter and Sven Erik Mattson. Fundamentals of synchronous control in Modelica. *Proceedings of the 9th International Modelica Conference*, 2012. doi:10.3384/ecp1207615.

Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier and John Irwin. Aspect-oriented programming. *Lecture Notes in Computer Science*, 1241:220–242, 1997. doi:10.1007/BFb0053371.

Jørgen Lindskov Knudsen, Mats Löfgren, Ole Lehrmann Madsen and Boris Magnusson. *Object-oriented environments: the Mjølner approach*, Prentice Hall, 1993.

Martin Otter, Bernhard Thiele and Hilding Elmqvist. A Library for Synchronous control systems in Modelica. *Proceedings of the 9th International Modelica Conference*, 2012. doi:10.3384/ecp1207627.

Antero Taivalsaari. Classes vs. prototypes: some philosophical and historical observations. *Journal of Object-Oriented Programming*, 10(7):44–50, 1997.

Peri Tarr, Harold Ossher, William Harrison and Stanley M. Sutton. N degrees of separation: multi-dimensional separation of concerns. *Proceedings of the 21st International Conference on Software Engineering*, 1999. doi:10.1145/302405.302457.

Peter Wegner. Dimensions of object-based language design. *Proceedings of the 2nd conference on Object-Oriented Programming, Systems, Languages, and Applications*, 1987. doi: 10.1145/38765.38823.

7 Appendix A

The following list is a rough estimation of examples of the Modelica Standard Library 3.2.2 that might be of interest for synchronous adaptation; about 60 existing continuous models have been identified. As criteria to consider a model relevant for synchronous adaptation, the containment of components modelling a controller has been chosen; further only *examples* – i.e., models extending `Modelica.Icons.Example` – have been considered. The potential examples of interest are (the actual example models are highlighted *green*):

```
Blocks
  Examples
    PID_Controller
    NoiseExamples
      ActuatorWithNoise
Electrical
  Machines
    Examples
      AsynchronousInductionMachines
        SwitchYD
        AIMC_Inverter
        AIMC_Conveyor
        AIMC_withLosses
      SynchronousInductionMachines
        SMR_Inverter
        SMPM_Inverter
        SMPM_CurrentSource
        SMPM_VoltageSource
        SMPM_Braking
        SMEE_Generator
        SMEE_LoadDump
        SMEE_Rectifier
  PowerConverters
    Examples
      ACDC
        Rectifier1Pulse
          Thyristor1Pulse_R
          Thyristor1Pulse_R_Characteristic
        RectifierBridge2Pulse
          HalfControlledBridge2Pulse
          ThyristorBridge2Pulse_R
          ThyristorBridge2Pulse_RL
          ThyristorBridge2Pulse_RLV
          ThyristorBridge2Pulse_RLV_Characteristic
          ThyristorBridge2Pulse_DC_Drive
        RectifierCenterTap2Pulse
          ThyristorCenterTap2Pulse_R
          ThyristorCenterTap2Pulse_RL
          ThyristorCenterTap2Pulse_RLV
          ThyristorCenterTap2Pulse_RLV_Characteristic
        RectifierCenterTapmPulse
          ThyristorCenterTapmPulse_R
          ThyristorCenterTapmPulse_RL
          ThyristorCenterTapmPulse_RLV
          ThyristorCenterTapmPulse_RLV_Characteristic
        RectifierBridge2mPulse
          HalfControlledBridge2mPulse
          ThyristorBridge2mPulse_R
          ThyristorBridge2mPulse_RL
          ThyristorBridge2mPulse_RLV
          ThyristorBridge2mPulse_RLV_Characteristic
          ThyristorBridge2mPulse_DC_Drive
        RectifierCenterTap2mPulse
          ThyristorCenterTap2mPulse_R
          ThyristorCenterTap2mPulse_RL
          ThyristorCenterTap2mPulse_RLV
          ThyristorCenterTap2mPulse_RLV_Characteristic
      DCAC
        SinglePhaseTwoLevel
          SinglePhaseTwoLevel_R
          SinglePhaseTwoLevel_RL
        MultiPhaseTwoLevel
          MultiPhaseTwoLevel_R
          MultiPhaseTwoLevel_RL
      DCDC
        ChopperStepDown
          ChopperStepDown_R
          ChopperStepDown_RL
        HBridge
          HBridge_R
          HBridge_RL
          HBridge_DC_Drive
Magnetic
  QuasiStatic
    FundamentalWave
      Examples
        BasicMachines
          InductionMachines
            IMC_Inverter
          SynchronousMachines
            SMPM_CurrentSource
            SMR_CurrentSource
Mechanics
  MultiBody
    Examples
      Systems
        RobotR3
          oneAxis
          fullRobot
Fluid
  Examples
    PumpingSystem
    DrumBoiler
      DrumBoiler
    ControlledTankSystem
      ControlledTanks
    AST_BatchPlant
      BatchPlant_StandardWater
    TraceSubstances
      RoomCO2WithControls
Thermal
  HeatTransfer
    Examples
      ControlledTemperature
```

MetaModelica – A Symbolic-Numeric Modelica Language and Comparison to Julia

Peter Fritzson Adrian Pop Martin Sjölund Adeel Asghar
PELAB – Programming Environment Lab, Dept. of Computer and Information Science
Linköping University, SE-581 83 Linköping, Sweden
{peter.fritzson,adrian.pop,martin.sjolund,adeel.asghar}@liu.se

Abstract

The need for integrating system modeling with advanced tool capabilities is becoming increasingly pronounced. For example, a set of simulation experiments may give rise to new data that are used to systematically construct a series of new models, e.g. for further simulation and design optimization. Such combined symbolic-numeric capabilities have been pioneered by dynamically typed interpreted languages such as Lisp and Mathematica. Such capabilities are also relevant for advanced modeling and simulation applications but lacking in the standard Modelica language. Therefore, this is a topic of long-running design discussions in the Modelica Design group. One contribution in this direction is MetaModelica, that has been developed to extend Modelica with symbolic operations and advanced data structures, while preserving safe engineering practices through static type checking and a compilation-based efficient implementation. Another recent effort is Modia, implemented using the Julia macro mechanism, making it dynamically typed but also adding new capabilities. The Julia language has appeared rather recently and has expanded into a large and fast-growing ecosystem. It is dynamically typed, provides both symbolic and numeric operations, advanced data structures, and has a just-in-time compilation-based efficient implementation. Despite independent developments there are surprisingly many similarities between Julia and MetaModelica. This paper presents MetaModelica and its environment as a large case study, together with a short comparison to Julia. Since Julia may be important for the future Modelica, some integration options between Modelica tools and Julia are also discussed, including a possible approach for implementing MetaModelica (and OpenModelica) in Julia.

Keywords: Modelica, MetaModelica, symbolic, Julia, meta-programming, language, compilation

1 Introduction

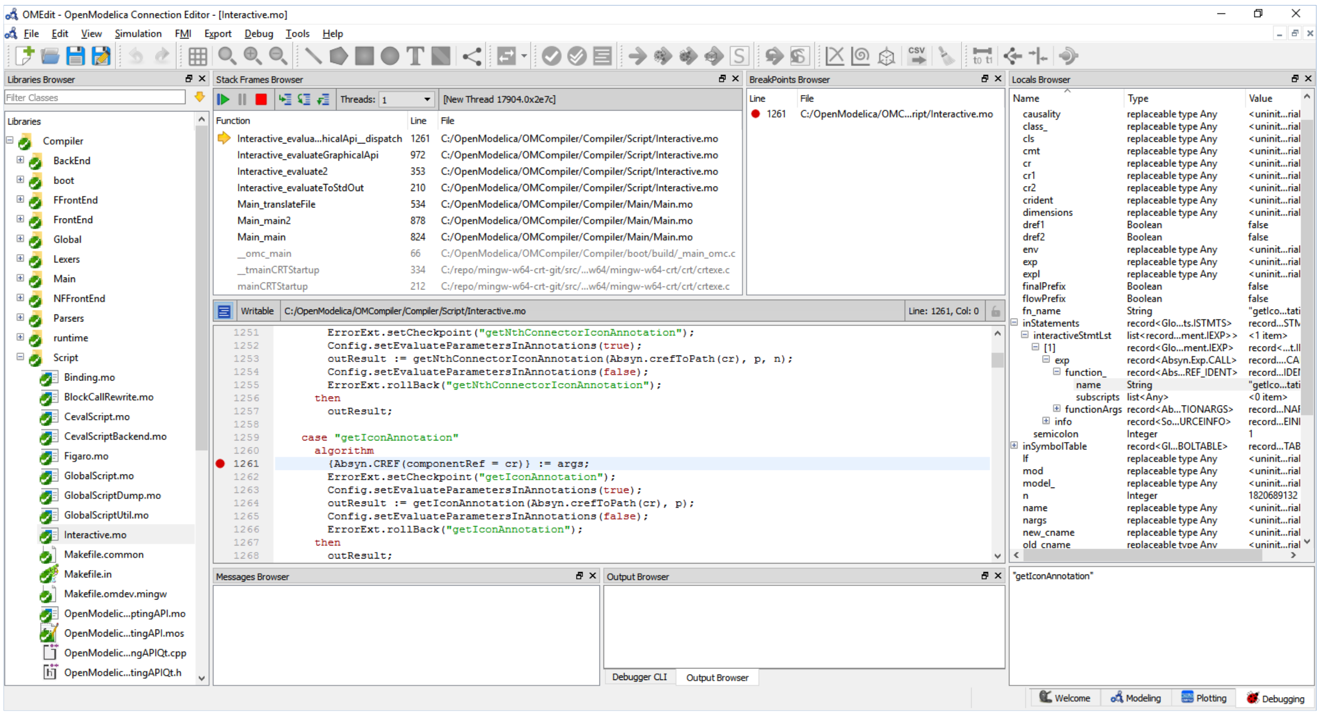
Advanced development of today's complex products requires integrated environments and equation-based object-oriented declarative languages such as Modelica (Fritzson, 2014; Modelica Association, 2017) for modeling and simulation. Such combined symbolic-numeric capabilities and advanced data structures have been pioneered by dynamically typed interpreted languages such as Lisp (Steel, 1993) and Mathematica (Wolfram, 2003), but are also relevant for modeling and simulation applications. Therefore, this is a topic of design discussions in the Modelica Design group regarding the future Modelica, and has also motivated the development of MetaModelica (Fritzson et al 2005; Pop et al, 2006, Fritzson et al, 2011) and Modia (Elmqvist et al, 2016; Elmqvist et al 2017);

1.1 Motivation and Design Goals

At the time when the MetaModelica effort was started, MetaModelica 1.0 (Fritzson et al 2005), there was no existing efficiently compiled language that combined strong numeric and symbolic capabilities. Our vision was to extend Modelica in that direction via MetaModelica, in a backwards compatible way, supporting the Modelica design goals of safe engineering practices through static type checking, and explicitly declared types for increased model readability and efficient compilation. In the longer term the goal was an efficient interactive environment based on incremental compilation or just-in-time compilation (Section 8.4).

However, in the meantime the rather young language Julia (Bezanson et al 2017; Julialang 2018) has matured, (Julia 1.0 was released in August 2018), with similar design goals of an efficiently compiled interactive symbolic-numeric language. However, also with the goals of dynamic typing and automatic interfacing with libraries in other languages, and no special requirement of integrating with the Modelica modeling language.

The design of MetaModelica has been mostly influenced by Modelica, Standard ML (Milner et al, 1997) and RML (Pettersson 1989), whereas Julia has been more influenced by dynamic languages such as Lisp and Mathematica.



[New Thread 17904.0x2e7c]

Function	Line	File
Interactive_evalua...hicalApi__dispatch	1261	C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Interactive_evaluateGraphicalApi	972	C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Interactive_evaluate2	353	C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Interactive_evaluateToStdOut	210	C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Main_translateFile	534	C:/OpenModelica/OMCompiler/Compiler/Main/Main.mo
Main_main2	878	C:/OpenModelica/OMCompiler/Compiler/Main/Main.mo
Main_main	824	C:/OpenModelica/OMCompiler/Compiler/Main/Main.mo
__omc_main	66	C:/OpenModelica/OMCompiler/Compiler/boot/build/_main_omc.c
__tmainCRTStartup	334	C:/repo/mingw-w64-crt-git/src/...w64/mingw-w64-crt/crt/crtexe.c
mainCRTStartup	212	C:/repo/mingw-w64-crt-git/src/...w64/mingw-w64-crt/crt/crtexe.c

Line	File
● 1261	C:/OpenModelica/OMC...ript/Interactive.mo

Writable C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo Line: 1261, Col: 0

```
        ErrorExt.setCheckpoint("getNthConnectorIconAnnotation");
        Config.setEvaluateParametersInAnnotations(true);
        outResult := getNthConnectorIconAnnotation(Absyn.crefToPath(cr), p, n);
        Config.setEvaluateParametersInAnnotations(false);
        ErrorExt.rollBack("getNthConnectorIconAnnotation");
      then
        outResult;

    case "getIconAnnotation"
      algorithm
        {Absyn.CREF(componentRef = cr)} := args;
        ErrorExt.setCheckpoint("getIconAnnotation");
        Config.setEvaluateParametersInAnnotations(true);
        outResult := getIconAnnotation(Absyn.crefToPath(cr), p);
        Config.setEvaluateParametersInAnnotations(false);
        ErrorExt.rollBack("getIconAnnotation");
      then
        outResult;
```

Name	Type	Value
causality	replaceable type Any	<uninit...rial
class_	replaceable type Any	<uninit...rial
cls	replaceable type Any	<uninit...rial
cmt	replaceable type Any	<uninit...rial
cr	replaceable type Any	<uninit...rial
cr1	replaceable type Any	<uninit...rial
cr2	replaceable type Any	<uninit...rial
crident	replaceable type Any	<uninit...rial
dimensions	replaceable type Any	<uninit...rial
dref1	Boolean	false
dref2	Boolean	false
env	replaceable type Any	<uninit...rial
exp	replaceable type Any	<uninit...rial
expl	replaceable type Any	<uninit...rial
finalPrefix	Boolean	false
flowPrefix	Boolean	false
fn_name	String	"getIco...tati
⊟ inStatements	record<Glo...ts.ISTMTS>	record...STM
⊟ interactiveStmtLst	list<record...ment.IEXP>>	<1 item>
⊟ [1]	record<Glo...ment.IEXP>	record<...t.II
⊟ exp	record<Absyn.Exp.CALL>	record....CA
⊟ function_	record<Abs...REF_IDENT>	record...IDEI
name	String	"getIco...tati
subscripts	list<Any>	<0 item>
⊞ functionArgs	record<Ab...TIONARGS>	record...NAI
⊞ info	record<So...URCEINFO>	record...EINI
semicolon	Integer	1
⊞ inSymbolTable	record<Gl...BOLTABLE>	record...TAB
lf	replaceable type Any	<uninit...rial
mod	replaceable type Any	<uninit...rial
model_	replaceable type Any	<uninit...rial
n	Integer	1820689132
name	replaceable type Any	<uninit...rial
nargs	replaceable type Any	<uninit...rial
new_cname	replaceable type Any	<uninit...rial
old_cname	replaceable type Any	<uninit...rial

"getIconAnnotation"

Figure 1. The integrated MetaModelica OMEdit-based development environment in debugging mode. *Left*: the package browser. *Top*: the active stack frames (including C routines) and breakpoints. *Middle*: text editing and breakpoint setting. *Right*: the local variables browser. The user can switch to modeling mode which has both textual and graphical editing.

The advent of Julia has changed the situation, as pointed out by (Elmqvist et al, 2016). Julia is a very capable and efficient symbolic-numeric language available with a rapidly growing ecosystem and set of libraries. Thus, it seems likely that Julia will influence the future of Modelica. The Modia prototype in Julia demonstrates several advantages but lacks support for safe engineering practices via static type checking. In Section 8 we briefly discuss ways of integrating Modelica tools with Julia without losing the Modelica static type checking.

A further discussion of related work is available in Section 10.

In the following, when we mention MetaModelica, we usually also include Modelica, since MetaModelica is an extension of Modelica.

1.2 Contributions

The contributions of this paper are not about inventing new language constructs. The introduced constructs have already been well proven in several other languages. Similar statements have been made regarding the Julia language. However, in the context of Modelica there are contributions on integrating such constructs into the Modelica language including the Modelica type system in a backwards compatible way.

Another contribution is the comparison of Julia and MetaModelica, showing many similarities and how Julia-like features have been integrated into the Modelica language via MetaModelica.

There are also contributions in the form of the very large case study of implementing the OpenModelica compiler in MetaModelica in an efficient way, and using the language and the associated developed environment (Figure 1, Section 8) for this large effort. Large case studies are valuable from a scientific point of view since it is often the case that results from investigations of small toy problems may not be true when problem sizes are scaled up.

1.3 Paper Organization

This paper is organized as follows.

Section 2 compares basic properties of MetaModelica and Julia. Sections 3 and 4 introduces uniontypes, tree and list data structures. Section 5 presents pattern matching including a symbolic example. Section 7 discusses compiler performance.

Section 8 presents the new OMEdit-based development environment for MetaModelica 3.0 and gives a comparison to the Eclipse-based MDT plug-in

Section 9 discuses integration of Modelica tools with Julia, whereas Section 10 presents related work and makes a short comparison to functional languages and languages such as Julia and Python. Finally, Section 11 gives conclusions and future work.

2 Some Properties of MetaModelica and Julia

We start by briefly summarizing and comparing some basic properties of MetaModelica and Julia.

2.1 Syntax

The syntax of the MetaModelica extension of Modelica is strongly influenced by Modelica, and to a lesser extent by Standard ML and C++. The Julia syntax is more influenced by languages like Python. Both are influenced by Matlab. The Julia syntax is more concise whereas the MetaModelica syntax is more verbose and descriptive, with more keywords.

2.2 Type System and Dynamic/Static Typing

MetaModelica/Modelica is structurally typed with some nominal typing parts, whereas Julia has a completely nominal type system. Thus, in Julia, concrete types may not be subtypes of each other. Both languages have concrete and abstract types, and parameterized types.

MetaModelica is a statically typed language; there are rules for determining the type of every expression in the program. Conversely, Julia is dynamically typed, types are properties of data values, and are dynamically created at runtime and implied by the way data flows through the program during execution. In static languages *expressions* have types, in dynamic languages *values* have types.

However, Julia has a rather sophisticated language for describing types, and it is possible to annotate expressions with types. For example, in Julia, `z::T` is an assertion that `z` is a value of type `T`; if that is true, `z::T` evaluates to the value of `z`, otherwise an error is raised. Type annotations in function signatures are slightly different: instead of asserting the type of an existing value, they indicate that the function only applies if the corresponding argument is of the indicated type.

To summarize, MetaModelica/Modelica is static, structural, and parametric, whereas Julia is dynamic, nominal, and parametric.

2.3 Multiple Dispatch and Overloading

Overloading of an operator or function means that in the presence of multiple implementations/definitions the definition with matching argument types is selected.

For some reason Julia has chosen to change from the well-established *overloading* terminology to instead use the term *multiple dispatch*. The new term might be more descriptive, but this change may cause some initial confusion for users. There is some arguing that dynamic selection is a reason for the new term, but one could instead talk about dynamic overloading.

MetaModelica provides user-defined overloading of both functions and operators, whereas standard Modelica only provides operator overloading. In both cases the selection is made at compile time based on statically available types of argument expressions. In Julia, the selection is done either at compile-time if the type can be inferred by the compiler, or at run-time based on runtime type tags of argument values.

3 Tree Data Structures

What are the needs for data structures and operations for symbolic (meta-programming) capabilities? One of the most common examples of programs that manipulate and produce other programs are compilers, which translate programs in some language into the same or another language. A small symbolic manipulation example is presented in Section 5.3.

The most common data type representation for programs in compilers are tree structures, and typical operations are transformations of such trees into trees during the translation process. Lists are a special case of tree data types but are typically given special support in many symbolic programming languages.

Tree data types have two interesting properties:

- Uniontype – a tree data type is typically the union of a number of node types, each representing a tree node.
- Recursive type – the children of a tree node may a type which is the tree data type itself.

Below we describe the MetaModelica uniontype language extension, give some examples of its usage, and briefly compare to Julia.

3.1 Uniontypes

The uniontype MetaModelica construct is a restricted class that can be viewed as the union of the record classes it contains. The keyword `uniontype` is followed by the name of the uniontype, in the example below called `Exp`

A record type belonging to a uniontype is called a union member record.

This example shows a small expression tree using uniontype `Exp` containing six different node types represented as Modelica record types, which must be declared within the scope of the union type. The `uniontype` restricted class construct has been extensively used in a Modelica context.

```
uniontype Exp
  record RCONST   Real rval;     end RCONST;
  record INTconst Integer exp1; end INTconst;
  record ADDop Exp exp1; Exp exp2; end ADDop;
  record SUBop Exp exp1; Exp exp2; end SUBop;
  record MULop Exp exp1; Exp exp2; end MULop;
  record DIVop Exp exp1; Exp exp2; end DIVop;
  record NEGop Exp exp1;           end NEGop;
end Exp;
```

The uniontype class grammar is as follows:

```
class_prefixes :
[ partial ]
```

```
( class | model | [ operator ] record |
block | [ expandable ] connector
| type | package | [ ( pure | impure ) ] [
operator] function | operator | uniontype)
```

The uniontype construct is used by functional languages such as OCAML, Standard ML, Haskell, etc. In several of these languages the uniontype construct is called *datatype*.

Uniontypes are also very common in Julia. However, in Julia the uniontypes are constructed dynamically at run-time since they are properties of values, not of expressions. Uniontypes in Julia can also be named and explicitly defined using the `Union` keyword:

```
IntOrString = Union{Int,AbstractString}
```

3.2 Main Properties of Uniontypes

The MetaModelica uniontype construct is a restricted class with the following main properties:

- Uniontype elements can be record declarations, replaceable type declarations declared using keywords replaceable type, *only* allowed to be used for type parameterization of the member records (and function(s)) and *not* to introduce uniontype member records. A record type declared within a uniontype is called a *uniontype member record*.
- Uniontypes can be *recursive*, i.e., reference themselves. That is the case in the above `Exp` example, where `Exp` is referenced inside its member record types.
- The typing rules for a uniontype are similar to operator records, i.e., nominal typing comparing type names. To check subtyping, (currently type identity) of two uniontypes, it is tested whether they belong to a subtype with the same name.
- Uniontypes can be parameterized by other types, using replaceable, similar to other restricted classes in Modelica.
- Inheritance, `extends`, between uniontypes is currently not allowed. The reason is that all issues for efficient implementation of such a feature are not yet resolved.
- Inheritance between member records is allowed e.g. `record ADDop2 = ADDop`; or using the long form: `record ADDop2 extends ADDop`; … `end ADDop2`
- Uniontypes provides a type-safe mechanism for variant records.

3.3 Calling Member Record Constructors

A *uniontype member record* constructor can be called using function syntax similar to standard record constructors, where the uniontype name is prefixed to the member record name to disambiguate:

```
UnionTName.MemberRecord()
```

If the union type is imported into a scope, the uniontype name prefix is not needed, for example:

```
import UnionTName.*;
MemberRecord()
```

For example, to construct the small expression tree of Figure 2 below using the above `Exp` uniontype without importing, the following would be needed:

```
Exp.ADDop(Exp.RCONST(12), Exp.MULop(
Exp.RCONST(5), Exp.RCONST(13)))
```

If importing of `Exp` into the current scope is used, the expression becomes more concise:

```
import Exp.*;
ADDop(RCONST(12), MULop( RCONST(5),
RCONST(13)))
```

3.4 A Small Expression Tree Example

A small expression tree, of the expression 12+5*13, is depicted in Figure 2.

Using the `Exp` record constructors `ADDop`, `MULop`, `RCONST`, this tree can be constructed by the expression ADDop(RCONST(12), MULop(RCONST(5), RCONST(13)))

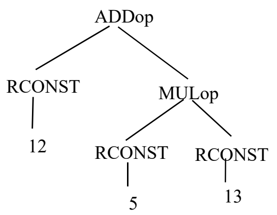


Figure 2. Abstract syntax tree of the expression 12+5*13.

3.5 Supertype Any

The predefined type `Any` is a supertype of any other MetaModelica type, i.e., all other MetaModelica types are subtypes of `Any`.

Since all other types are subtypes of `Any`, by using `Any` in a replaceable type declaration, it is possible to avoid any constraints and provide full flexibility in using any type as a type parameter in the following replaceable type declaration:

```
replaceable type TypeParam = Any
  constrainedby Any;
```

This is equivalent to the following, since the default type is used as constraining type if that is missing:

```
replaceable type TypeParam = Any;
```

The type `Any` is also present in Julia, with the same semantics that it is a supertype of all other types.

3.6 Predefined Uniontype Option for Optional Values

The predefined MetaModelica `Option` uniontype provides a type-safe way of representing the common situation where a data item is optionally present in a data structure.

The constructor `NONE()` is used to represent the case where the optional data item is not present, whereas the constructor `SOME()` is used when the data item is present in the data structure.

The following is a definition of the parameterized `Option` uniontype with a type parameter:

```
uniontype Option
 replaceable type TypeParam = Any
   constrainedby Any;
 record NONE
 end NONE;
 record SOME
   TypeParam elem;
 end SOME;
end Option;
```

For example, a `StringOption` type and a function using it are defined:

```
uniontype StringOption = Option(redeclare
  TypeParam=String);
function stringOrDefault
  input StringOption strOpt;
  input String default;
  output String str;
algorithm
  str := match strOpt
    case Option.SOME(str) then str;
    else default;
  end match;
end stringOrDefault;
```

Calling the function a few times:

```
stringOrDefault(Option.NONE(),"default")
 "default"
stringOrDefault(Option.SOME("string"),
   "default")
"string"
```

A similar predefined facility is available in Julia. Declaring a function argument or a record field as having the type `Union{T, Nothing}` allows setting it either to a value of type `T`, or to `nothing` to indicate that there is no value.

3.7 Parameterized Union Types

Parameterized union types with opaque type parameters are available. This means that only minimal information about the type parameter is needed.

There is also support for `redeclare` in cases where only information about sorting order needs to be available about the type used as type parameter. For example, this sorting order is provided by the type `Key` given by the function `keyCompare` in the `AvlSetString` package available in the OpenModelica utility library.

```
package AvlSetString
  import BaseAvlSet;
  extends BaseAvlSet;
  redeclare type Key = String;
  redeclare function extends keyStr
    algorithm
     outString := inKey;
    end keyStr;
  redeclare function extends keyCompare
    algorithm
     outResult :=
        stringCompare(inKey1, inKey2);
    end keyCompare;
end AvlSetString;
```

4 Lists and Tuples

List and tuple data types are common in many languages used for meta-programming and symbolic programming, and are available in both MetaModelica and Julia.

4.1 Lists

The following MetaModelica operations allows creation of lists and addition of new elements in front of lists in a declarative way, i.e., such lists are immutable. Extracting elements is done through pattern-matching in match-expressions.

- `list` – `list(el1,el2,el3, ...)` creates a list of elements of identical type. Examples: `list()` is the empty list, `list(2,3,4)` is a list of integers.
- `::` – the `::` operator in the expression `element::lst` adds an element in front of the list `lst` and returns the resulting list.

The types of lists and list variables can be specified as follows:

- `list` – `list<type-expr>` using angle-bracket notation is a `list` *type constructor*, e.g.:
  ```
  type RealList = list<Real>;
  ```
- Direct declaration of a variable `rlist` that denotes a list of real numbers:
  ```
  list<Real>   rlist;
  ```

A list type is a parametrized uniontype; the Option type is also such a type. The only addition is the `::` operator.

Lists are available in Julia with about the same semantics and similar but slightly different syntax.

4.2 Tuples

Tuples can be viewed as instances of anonymous records. The syntax is a parenthesized list. The same syntax is used in extended Modelica presented here and is in fact already present in standard Modelica as a receiver of values for functions returning multiple results.

- An example of a tuple literal: `(a,b,"cc")`
- A tuple with a single element can be created using the `tuple` constructor instead of the short-hand parentheses notation: `tuple(a)`
- A tuple can be seen as being returned from a function with multiple results in standard Modelica:
  ```
  (x,y,z) := foo(var, 2, 3, 5);
  ```
- Access of field values in tuples can be achieved via pattern-matching, e.g. the following will extract the three field values from a tuple value:
  ```
  (x,y,z) := tuplevalue
  ```

The main reason to introduce tuples is for convenience of notation. You can use them directly without explicit declaration. Tuples using this syntax are already present in the major functional programming languages.

A tuple will of course also have a type. When tuple variable types are needed, they can for example be declared using the following notation:

```
type VarBND  = tuple<Ident, Integer>;
```

or directly in a declaration of a variable `bnd`:

```
tuple<Ident, Integer>   bnd;
```

Tuples are also available in Julia, with the same syntax (parenthesized list) and semantics. Tuple types can also be defined explicitly in Julia using the `Tuple` keyword:

```
 Tuple{Ident,Int}
```

5 Match Expressions for Processing Complex Data

Matching on instances of structured data types such as trees is one of the central facilities in symbolic processing languages. The matching provided by the match-expression construct is very close to similar facilities in many functional languages but is also related to switch statements in C or Java. Match-expressions have two important advantages over traditional switch statements e.g. available in languages such as C or Java:

- A match-expression can appear in any of the three Modelica contexts: expressions, statements, or in equations.
- The selection in the case branches is based on pattern matching, which reduces to equality testing or switch in simple cases but is much more powerful in the general case.

Regarding allowed patterns used in match-expressions they are defined by the pattern language, see Section 5.2. For example, constants can be patterns, e.g., `"one"`, `384`, `RequirementStatus.violated`. Constructors with or without pattern variables can be patterns. The wildcard pattern _ (underscore) matches anything.

A very simple example of a match-expression is the following code fragment, which returns a number corresponding to a given input string. The pattern matching is very simple – just compare the string value of s with one of the constant pattern strings "one", "two" or "three", and if none of these matches return 0 since the wildcard pattern _ matches anything.

```
  String s;
  Real   x;
algorithm
  x := match s
    case "one"   then 1;
    case "two"   then 2;
    case "three" then 3;
    case   _     then 0;
  end match;
```

Alternatively, an else-branch, `else 0;`, can be used instead of the last wildcard pattern `case _ then` 0:

Another, more useful example, but still trivial since it only shows constants, is a match expression converting an enumeration value to a Boolean value:

```
type RequirementStatus =
  enumeration(violated, undecided, satisfied);
function RequirementStatusToBoolean
   input  RequirementStatus r;
   input  Boolean undecided = false;
   output Boolean b;
algorithm
  b = match r
   case RequirementStatus.violated  then false;
   case RequirementStatus.undecided then
                                    undecided;
   case RequirementStatus.satisfied then true;
  end match;
end RequirementStatusToBoolean;
```

The match expression in the above conversion function gives the same result as the following if-expression, but can be compiled more efficiently (Section 5) and is easier to follow:

```
  b = if r == RequirementStatus.violated
        then false
      elseif r == RequirementStatus.undecided
        then undecided
      else true;
```

The general syntactic structure of match-expressions starting with the `match` keyword is indicated by the syntax outline below. The else-branch is optional and is identical to a `case _` branch. Local equation sections contain equations, local algorithm sections contain statements. The syntax outline:

```
match <match-value-expr> <opt-local-decl>
    ...
  case <pat-expr>
    [equation | algorithm]
      <opt-equations-or-statements>
    then <expr>;
    ...
  case <pat-expr>
    [equation | algorithm]
      <opt-equations-or-statements>
    ...
  else
    [equation | algorithm]
      <opt-equations-or-statements>
end match;
```

A slightly more advanced usage of match-expressions compared to the above trivial cases is in a small expression evaluator, the function `eval`. Here we use as-binding of the result of a match to `x`, and standard Modelica dot-notation to access values, e.g. `x.rval` or `x.exp2`. The constructor pattern notation with empty parentheses, e.g., `ADDop()`, means matching with arbitrary arguments to that constructor.

```
function eval
  input  Exp  inExpression;
  output Real result;
  import Exp.*;
algorithm
 result := match x as inExpression
  case RCONST() then x.rval;
```

```
    case ADDop() then eval(x.exp1)+eval(x.exp2);
    case SUBop() then eval(x.exp1)-eval(x.exp2);
    case MULop() then eval(x.exp1)*eval(x.exp2);
    case DIVop() then eval(x.exp1)/eval(x.exp2);
    case NEGop() then -eval(x.exp);
  end match;
 end eval;
```

Without the `import Exp.*` clause, constructors would need the `Exp` prefix, e.g. `Exp.RCONST()`, `Exp.ADDop`.Regarding Julia, there are several third-party libraries available for pattern matching is available with a semantics very close to the abovementioned match expression construct. Match.jl (Squire 2013) and Rematch.jl (RelationalAI 2018) use the following syntax:

```
 @match item begin
     pattern1              => result1
     pattern2, if cond end => result2
    pattern3 || pattern4  => result3
    _                     => default_result
End
```

To make its semantics even closer to MetaModelica match we included the Rematch.jl Julia package in our prototype MetaModelica.jl compatibility layer and enhanced it to also include named pattern matching (Section 5.1) and `matchcontinue` semantics (pattern matching with exception handling, see Fritzson et. al 2011 for details). To implement match and `matchcontinue` semantics requires only 350 lines of Julia code. See also Section 9.

5.1 Named Pattern Matching with Pattern Variables vs Positional Matching

Named pattern matching uses named association to match/bind pattern variables to values of corresponding named arguments (e.g., record field names) of constructors.

This notation is more verbose than that for positional pattern matching but has the advantages that it is more robust against model changes such as constructor argument order, invention and maintenance of pattern variable names is avoided, and usually increased readability since the argument names are visible, especially if there are many arguments.

This example is of an `ADDop` named pattern mentioning field names `exp1` and `exp2` with pattern variables `e1` and `e2` which become bound to values during matching.

Named pattern matching is possible, i.e., the position of the pattern variable does not matter, only the field name (below `exp1` or `exp2`) which it is associated to:

```
 case ADDop(exp1=e1,exp2=e2)
    then eval(e1) + eval(e2);
```

In positional pattern matching this case would appear as follows. It is more concise but dependent on argument order:

```
 case ADDop(e1,e2) then eval(e1)+eval(e2);
```

5.2 Pattern Expressions

Pattern expressions are used in match expressions and can have the following forms:

- Patterns can contain literal constants of strings, integers, real numbers, Booleans, enumeration values, e.g. `"string"`, `555`, `3.14`, `true`, `false`, `Sizes.medium`.
- Patterns can contain the _ wildcard which matches one item of anything.
- A pattern can be a pattern variable, i.e., an identifier, which can appear as an argument to a constructor, and which matches one item of anything.
- A pattern variable need not be declared. Its type is inferred using simple type inferencing, e.g. from the corresponding formal parameter type when it appears as an argument to a record constructor.
- A pattern variable is automatically introduced into the local scope, e.g. a case-clause, where the variable is first mentioned. Therefore, it shadows variables with the same name in outer scopes.
- A pattern variable is bound to the value it matches during pattern matching.
- The same pattern variable may occur at most once in the main part of the pattern expression, i.e., excluding the optional guard part.
- Patterns can contain calls to *record constructor* functions, *not* to other kinds of functions except constructors such as the array constructor `array`(), the array function `cat`(), the `list`() constructor or the `tuple`() constructor.
- Positional and/or named argument constructor call syntax can be used in patterns containing constructors, e.g., the positional call `FOO(1,_,2)`, is allowed; a named argument call version, e.g., `FOO(field1=1,field3=2)`, or `FOO(field1=1,field3=myvar)`, where `myvar` is a pattern variable, is also allowed. Moreover, you can mix positional and named arguments in the call pattern, with positional arguments first: `FOO(1,field3=myvar)`.
- A constructor pattern `NAME`(…) can have an unspecified argument list denoted by an empty argument list as in `FOO()`. This matches the corresponding constructor, here `FOO`, with arbitrary (zero or more) arguments.
- A constructor pattern `NAME`(…) is interpreted as implicitly filling unspecified argument patterns _ at the end of the argument list until it matches the declared number of arguments of the constructor; in the case of `array`(…) matching arbitrary (zero or more) arguments after the specified arguments. For example, a constructor `R` with three members x, y, z, would fit all of the following patterns: `R()`, `R(v1)`,`R(v1,v2)`,`R(v1,v2,v3)`.

- Patterns can contain curly-brace array constructors, which match exactly those elements mentioned, e.g., `{}`, `{3,5}`, {3,5,_}, {3,5,6}, {a,5}. The array pattern {} matches an empty array value.
- Patterns can contain the `as` binding operator, [*e.g.* state1 `as FOO(env,…)`.
- Patterns may optionally have guards, i.e., conditional expressions that are evaluated at run-time and are part of the pattern condition, i.e., if the whole matching fails including the guard, the match may try another pattern if present. *Example*: `case REAL() guard x.value > 0 then x.value`.
- Currently the MetaModelica pattern language does *not* support explicit and/or combinations of patterns, e.g. `pattern1 and pattern2`, `pattern1 or pattern2` whereas the Rematch.jl code was influenced by Scala and does support this. The and-mechanism can be achieved by embedding two or more patterns in a list of patterns, e.g. `{pattern1, pattern2}`, whereas the or-mechanism can be achieved by having two case-rules, e.g., `case pattern1 …; case pattern2 … .`

Some pattern examples:

```
"a"          // constant literal string pattern
33           // constant literal Integer pattern
3.14         // constant literal Real pattern
false        // constant literal Boolean pattern
true         // constant literal Boolean pattern
p            // pattern variable pattern, name p
Sizes.medium // literal enumeration pattern
ADDop()      // constructor pattern with zero
             // or more arbitrary arguments
ADDop(3)     // constructor pattern, first is 3,
             // followed by arbitrary args
ADDop(_,_)   // constructor pattern with 2 or
             // more arbitrary arguments
ADDop(p,_)   // constructor pattern with 2 or
     // more arbitrary arguments, the first
     // argument bound to pattern variable p
(_,_)       // tuple pattern with 2 arguments
list(_,_)   // list pattern with >=2 arguments
x :: rest   // list pattern where x matches the
 // first element and rest the rest of the list
array(_)    // array pattern with one or more
            // arbitrary arguments
array(3,4)  // array pattern with the first
            // two elements being 3 and 4,
cat(1, {head}, rest) // Pattern which matches
 // both the head (first element) and
 // the rest (remaining elements) of an array
array()// array pattern, >= zero arguments
{_,_}  // array pattern, exactly two elements
{_,55} // array pattern; two elements, 2nd 55
{44}   // array pattern; one element being 44
{}     // array pattern, zero elements
{33,_} // array pattern,  two elements, 1st 33
{_,33,_,44} // array pattern with four
         // elements, the 2nd is 33, 4th is 44
```

Syntax rule:

```
pattern : expression
```

The pattern expression syntax is a subset of the general expression syntax. This is checked by semantics rules. The syntax looks slightly different in the Rematch.jl package, but all of the semantics are supported (and has some additional semantics as well).

5.3 Symbolic Differentiation Example

Symbolic differentiation of expressions is a symbolic operation that transforms expressions into differentiated expressions.

```
uniontype Exp
  record RCONST  Real e1;     end RCONST;
  record ADD Exp e1; Exp e2;  end ADD;
  record SUB Exp e1; Exp e2;  end SUB;
  record MUL Exp e1; Exp e2;  end MUL;
  record DIV Exp e1; Exp e2;  end DIV;
  record NEG Exp e1;          end NEG;
  record IDENT   String name; end IDENT;
  record CALL Exp id; Exp[:]args;end CALL;
  record AND Exp e1; Exp e2;  end AND;
  record OR  Exp e1; Exp e2;  end OR;
  record LESS Exp e1; Exp e2; end LESS;
 record GREATER Exp e1;Exp e2;end GREATER;
end Exp;
```

An example function `df` performs symbolic differentiation of the expression `expr` with respect to the variable `time`, returning a differentiated expression.

As previously mentioned, in the patterns _ is a reserved word that can be used as a placeholder instead of a pattern variable when the particular value in that place is not needed later as a variable value. The **as**-construct is used to bind the additional identifier to the matched value of the relevant expression.

In the following example the _ is used as a placeholder of any argument in one of the patterns, `CALL(IDENT("sin"),{_})`. This is a function call to sin with the argument list being an array of exactly one element `{_}`. The example also uses constructors with empty parentheses like `ADD()` to match for zero or more arguments with any contents.

The following well-known derivative rules are represented in the match-expression code:

- The time-derivative of a constant `RCONST()` is zero.
- The time-derivative of the time variable is one.
- The time-derivative of a time dependent variable `id` is `der(id)` but is zero if the variable is not time dependent, i.e., not in the list `tv/timevars`.
- The time-derivative of the sum `add(x.e1,x.e2)` of two expressions is the sum of the expression derivatives.
- The time-derivative of sin(x) is cos(x)*x' if x is a function of time, and x' its time derivative.

Some operators have been excluded in the `df` example below:

```
function df "Symbolic differentiation of
expression with respect to time"
  input  Exp expr;
  input  String[:] tv;
  output Exp diffexpr;
  import Exp.*;
```

```
algorithm
  diffexpr := match x as expr
   // der of constant
  case RCONST() then RCONST(0.0);
   // der of a variable
  case IDENT() then
      if x.id == "time" then RCONST(1.0)
      // der of time variable
      else if member(x.id,tv)
      // der of any variable id
        then CALL(IDENT("der"),{x.id})
      else RCONST(0.0);
  // (x.e1 + x.e2)′ => x.e1′ + x.e2′
  case ADD() then
    ADD(df(x.e1,tv), df(x.e2,tv));
  case SUB()then
    SUB(df(x.e1,tv), df(x.e2,tv));
 // (x.e1*x.e2)′ => x.e1′*x.e2+x.e1*x.e2′
  case MUL() then
    PLUS(MUL(df(x.e1,tv),x.e2),
         MUL(x.e1, df(x.e2,tv);));
  case DIV()then
    DIV(SUB(MUL(df(x.e1,tv),x.e2),
      MUL(x.e1,df(x.e2,tv))),
      MUL(x.e2,x.e2));
  case NEG() then NEG(df(x.e1,tv);
    // sin(x.e1)′ => cos(x.e1) * x.e1′
  case CALL(IDENT("sin"),{_}) then
   MUL(CALL(IDENT("cos"),{x.e2[1]}),
   df(x.e2[1],tv)); //first elem from e2
   case AND() then
     AND(df(x.e1,tv), df(x.e2,tv));
   case OR() then
     OR(df(x.e1,tv), df(x.e2,tv));
   case LESS() then
     LESS(df(x.e1,tv), df(x.e2,tv));
   case GREATER() then
     GREATER(df(x.e1,tv), df(x.e2,tv));
    // etc...
  end match;
end df;
```

6 Exception Handling

The available MetaModelica exception handling construct has the following structure:

```
  try
    // Perform something which might fail
  else
    // Perform something different
  end try;
```

This is used extensively in many of the existing MetaModelica applications. There is also a function `fail()`, which can be called to create a failure that can be caught by the next level exception handler, typically after emitting an error message.

Julia has a very similar exception handling mechanism, with a `try – catch` statement and a `throw` call to create exceptions, but additionally has named exceptions and the `finally` clause.

7 Compiler Size and Performance

The OpenModelica compiler is a very large application implemented in MetaModelica 3.0. The sizes of the main parts are shown in Table 1. It is also bootstrapped, i.e., it compiles itself (Sjölund et al, 2014).

Moreover, the new OpenModelica compiler frontend, (Pop, et al, 2019) using the new facilities of MetaModelica 3.0, has a flattening speed of between one and two orders of magnitude faster than the previous compiler frontend.

Table 1. Sizes of OpenModelica compiler phases, lines of code, including several code generators.

Compiler Phase	*Lines*
BackEnd (from flat Modelica to sorted equation systems)	106299
FrontEnd (up to flat Modelica)	152059
Intermediate representation for code generation	17368
Code generators (generated code)	356889
Code generators (template source code)	8957
Code generators template language compiler & runtime	14586
OpenModelica scripting environment	35460
Utility modules	31050
Total size (excl. generated code)	*412869*

The compilation speed for two example models is indicated in Table 2.

Table 2. Compilation speed of the OpenModelica compiler implemented in MetaModelica 3.0 for some models, using a standard desktop computer.

Example model and size	*Compile time* (s)
Hummod, 29145 equations	239 s
Engine V6 (analytic), 9016 eqs	26 s

8 New Development Environment

As previously mentioned, the new integrated OMEdit-based development environment supports algorithmic code development in MetaModelica 3.0 or Modelica 3.4, or equation-based Modelica 3.4 model development. There are four simulation arrow buttons visible in Figure 3, from the left: standard simulation, simulation with the transformational debugger for equation models, simulation with the algorithmic code debugger, and simulation with 3D graphic animation.

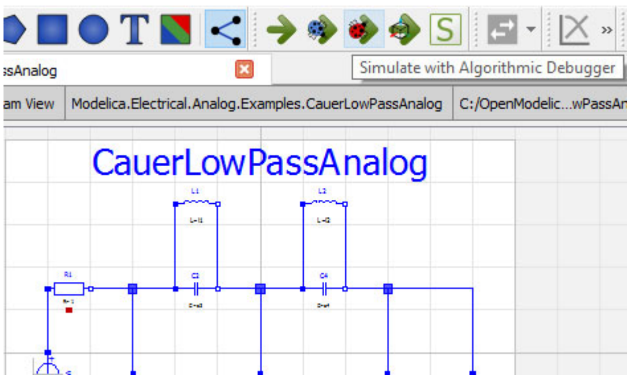

Simulate with Algorithmic Debugger
Modelica.Electrical.Analog.Examples.CauerLowPassAnalog
CauerLowPassAnalog

Figure 3. OMEdit with graphical view of an electrical model, as well as its four simulation+debug buttons.

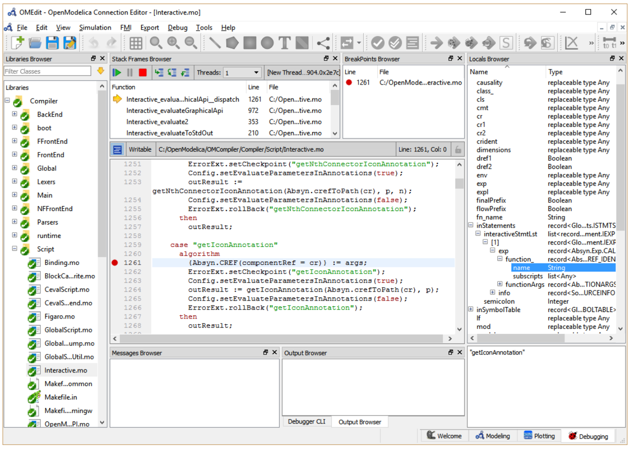
8.1 Why Develop a New Environment

What was the motivation for developing a new version of the integrated environment since the Eclipse-based MDT-plugin was already available? There are basically two reasons:

- *Ease of use*. Users and developers asked for a more integrated tool, instead of needing to install the rather complex Eclipse tool for textual model debugging and MetaModelica development.
- *Performance*. Several developers were dissatisfied with the Eclipse plugin since they felt it was too slow, even though it provided useful functionality. By contrast, the new OMEdit environment is very fast, also for large applications.

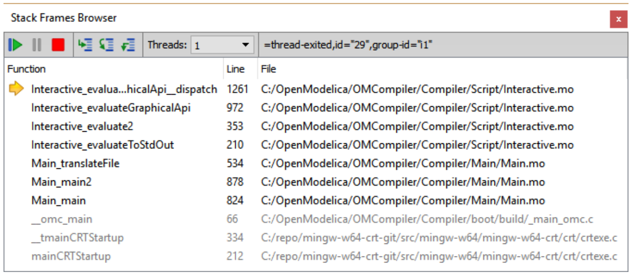
8.2 Browsing and Debugging

The OMEdit-based development environment supports browsing and searching of MetaModelica packages just as the MDT Eclipse plugin. Debugging, including setting breakpoints, stepping, conditional breakpoints, attaching the debugger to an already running process, etc., is supported. A new feature is the ability to also show C function calls in the stack trace. See Figure 4, Figure 5, and Figure 6.



```
          ErrorExt.setCheckpoint("getNthConnectorIconAnnotation");
          Config.setEvaluateParametersInAnnotations(true);
          outResult :=
getNthConnectorIconAnnotation(Absyn.crefToPath(cr), p, n);
          Config.setEvaluateParametersInAnnotations(false);
          ErrorExt.rollBack("getNthConnectorIconAnnotation");
        then
          outResult;

      case "getIconAnnotation"
        algorithm
          {Absyn.CREF(componentRef = cr)} := args;
          ErrorExt.setCheckpoint("getIconAnnotation");
          Config.setEvaluateParametersInAnnotations(true);
          outResult := getIconAnnotation(Absyn.crefToPath(cr), p);
          Config.setEvaluateParametersInAnnotations(false);
          ErrorExt.rollBack("getIconAnnotation");
        then
          outResult;
```

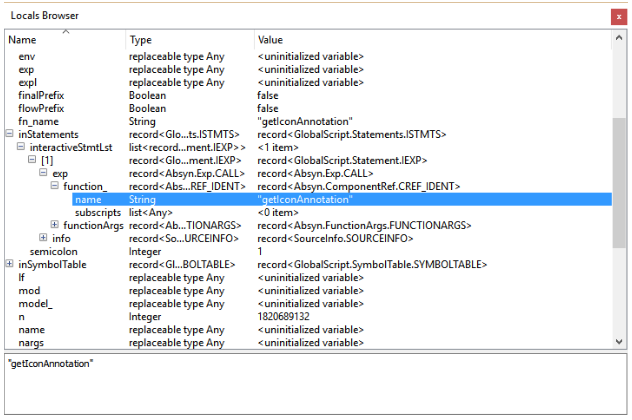
Figure 4. OMEdit during MetaModelica development. See also Figure 1 for more details.


Stack Frames Browser

Threads: 1 =thread-exited,id="29",group-id="i1"

Function	Line	File
Interactive_evalua...hicalApi__dispatch	1261	C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Interactive_evaluateGraphicalApi	972	C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Interactive_evaluate2	353	C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Interactive_evaluateToStdOut	210	C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Main_translateFile	534	C:/OpenModelica/OMCompiler/Compiler/Main/Main.mo
Main_main2	878	C:/OpenModelica/OMCompiler/Compiler/Main/Main.mo
Main_main	824	C:/OpenModelica/OMCompiler/Compiler/Main/Main.mo
__omc_main	66	C:/OpenModelica/OMCompiler/Compiler/boot/build/_main_omc.c
__tmainCRTStartup	334	C:/repo/mingw-w64-crt-git/src/mingw-w64/mingw-w64-crt/crt/crtexe.c
mainCRTStartup	212	C:/repo/mingw-w64-crt-git/src/mingw-w64/mingw-w64-crt/crt/crtexe.c

Figure 5. The function call stack trace browser showing Modelica and MetaModelica function calls at the top, C functions at the bottom.


Locals Browser

Name	Type	Value
env	replaceable type Any	<uninitialized variable>
exp	replaceable type Any	<uninitialized variable>
expl	replaceable type Any	<uninitialized variable>
finalPrefix	Boolean	false
flowPrefix	Boolean	false
fn_name	String	"getIconAnnotation"
inStatements	record<Glo...ts.ISTMTS>	record<GlobalScript.Statements.ISTMTS>
interactiveStmtLst	list<record...ment.IEXP>>	<1 item>
[1]	record<Glo...ment.IEXP>	record<GlobalScript.Statement.IEXP>
exp	record<Absyn.Exp.CALL>	record<Absyn.Exp.CALL>
function_	record<Abs...REF_IDENT>	record<Absyn.ComponentRef.CREF_IDENT>
name	String	"getIconAnnotation"
subscripts	list<Any>	<0 item>
functionArgs	record<Ab...TIONARGS>	record<Absyn.FunctionArgs.FUNCTIONARGS>
info	record<So...URCEINFO>	record<SourceInfo.SOURCEINFO>
semicolon	Integer	1
inSymbolTable	record<Gl...BOLTABLE>	record<GlobalScript.SymbolTable.SYMBOLTABLE>
lf	replaceable type Any	<uninitialized variable>
mod	replaceable type Any	<uninitialized variable>
model_	replaceable type Any	<uninitialized variable>
n	Integer	1820689132
name	replaceable type Any	<uninitialized variable>
nargs	replaceable type Any	<uninitialized variable>

"getIconAnnotation"

Figure 6. The local variables browser. Both standard Modelica data and MetaModelica data such as trees and lists are shown.

8.3 Separate Compilation

An efficient separate compilation mechanism for algorithmic MetaModelica or Modelica code is available, which is used routinely by the OpenModelica compiler developers to achieve rather fast turnaround time since more than two years. The compiler itself is a large application consisting of more than 250 packages, which is why separate compilation is quite important. Separate compilation of equation models is a separate topic not covered here, and is partly available using the FMI interface. The algorithmic code separate compilation mechanism works as follows:

A restriction has been introduced that all top-level packages are encapsulated and all dependencies of a module must be marked by an import statement. This improves performance in subsequent steps.

An additional useful restriction is that any public function, constant, or type may only refer to other public elements. By introducing this restriction, it is possible to create an interface file for each package that strips out protected elements and algorithm sections. Everything that remains in the resulting file is part of the interface, and loading each and every file (e.g., of the 250 OMC files) in the interface takes less than 0.6 seconds on a standard laptop.

For performance, a distinction between protected and public import elements in Modelica has been introduced. When calculating the list of interface files that a package depends on, we must start with all the import elements in the package that is going to be compiled. For each of these packages, add the public imports to the list of packages that are going to be loaded (and do this recursively if any of those packages contain public imports). This is the list of all interfaces needed to calculate all types used in functions of the package that is going to be compiled. This set is substantially smaller than loading every single interface file.

Thus, to compile a package, the interface dependencies and the main file is loaded. For each function in the package, perform instantiation of that function and send it to the code generator. Compile each of those files with a C compiler and perform linking of the total application.

The total time including linking when updating a file without changing its public interface is 4.5 seconds for typical file (about 6000 lines of MetaModelica) or about 8 seconds for a large file (about 13000 lines). The tests were performed using an Intel Core i7 3820 @3.60GHz. There were about 250 packages in the OpenModelica compiler application used for the measurements.

Regarding Julia, the LLVM just-in-time compiler produces code directly into a binary image in memory. The disadvantage is that compilation is eventually repeated when code is loaded. However, there are some facilities for saving precompiled code to avoid recompilation.

8.4 Just-in-Time and Incremental Compilation

Julia uses LLVM for just-in-time compilation which combines performance with flexibility and interactivity. MetaModelica is currently compiled to C-code that is compiled either using the GNU C compiler or the LLVM Clang compiler. Instead, by directly generating LLVM compatible binary code it would be possible to get faster compilation and also to utilize the just-in-time capabilities of LLVM. Thus, recently we have made prototyping efforts for adapting the OpenModelica compiler backend intermediate representation (IR) to an LLVM compatible form (Andersson and Eriksson, 2018) and to interface it to the LLVM just-in-time compiler (Tinnerholm, 2019). In this way it is possible to obtain just-in-time capabilities with the associated flexibility without dependence on the Julia run-time. Additionally, there is an earlier prototype incremental compilation functionality in OpenModelica (Klinghed and Jansson, 2008).

9 Integration with Julia in Modelica Environments

As previously mentioned, Julia provides a powerful environment and a rapidly growing set of libraries for computational applications. Thus, some kind of integration with Julia seems relevant for many Modelica tools. We have identified a few levels of integration between a Modelica tool and Julia, from less to more integration:

- Level 1. Using Julia as a scripting language and making an API available for calling the Modelica tool from Julia. This have benefits of making Modelica model simulation and analysis available from Julia, e.g. for applications such as model-based control system design, e.g., (Thiele et al, 2019). Such an integration has recently been made available via the OMJulia subsystem (Lie et al, 2019) in OpenModelica
- Level 2a. Introducing an external function declaration facility for Julia functions. The benefits include making Julia functions available to Modelica modelers.
- Level 2b. Generating Julia code from the Modelica tool, i.e., adding another target language in addition to the typical C / C++. The benefits include Julia functions available to Modelica modelers as external functions and leverage some Julia run-time system functions for supporting the tool implementation.
- Level 3. Using Julia as the implementation language for the Modelica tool. This has the advantage of making the powerful Julia language and ecosystem available for tool implementation supporting both numeric and symbolic operations, and with rich libraries.

Regarding Level 3, two approaches are language embedding, i.e., embedding a Modelica-like language subset into the Julia language, or complete implementation from scratch in order to preserve all Modelica semantics.

Language embedding is a quick approach and has been chosen, e.g., by the Modia effort and is discussed in more detail in Section 10. As mentioned, an important disadvantage of such an approach is the loss of the static type checking and safe engineering practices which has been a strong guiding principle of Modelica language design. However, it might be possible to develop a static type system with static type checking for a subset of Julia. Most Julia code will not pass such a type checker, but for code that passes, this may solve the problem of safe engineering practice that is lacking for dynamic languages Some work in that direction is mentioned in (Chung et al, 2016) where a static type checker for a very small subset of Julia has been developed.

Regarding the other approach, implementation from scratch, a quicker approach is automatic translation/porting of code if the existing Modelica tool implementation language is close enough to Julia. Given the strong similarities between MetaModelica and Julia, it might be possible to auto-translate most of the OpenModelica compiler to Julia and thereby obtain a fully compliant Modelica compiler with static type checking implemented in Julia. As a first step, a compatibility package for MetaModelica in Julia has been developed by us, including named pattern matching that was missing. One issue that was discovered is that recursive uniontypes that can be directly defined in MetaModelica are not possible in Julia. However, a solution was found by first declaring an abstract type of the uniontype which then can be referred to in the member structs. Another related issue is that Julia constants and types are declared in the order of the file, whereas in MetaModelica order does not matter. This either requires moving some MetaModelica code around if a very simple MetaModelica to Julia translator was implemented. Other than this, OpenModelica depends on a lot on external C code, which is expected to be the bulk of work to translate the entire compiler to Julia.

Performance is of great importance to OpenModelica but the MetaModelica compiler is primarily a high-level compiler and does not optimize many low-level operations due to maintenance issues of such code. Julia has a different approach where the language was designed to allow for high-performance code. An initial test showed that the Julia garbage collector is twice as fast as the Boehm garbage collector used in OpenModelica. And while the OpenModelica LLVM just-in-time compiler (Tinnerholm, 2019) is not feature-complete, it shows that LLVM just-in-time compilation such as Julia's could bring great performance benefits. If OpenModelica was ported to Julia and Modelica functions would be translated to the internal Julia AST, OpenModelica could gain performance by removing the interpreter and replacing it with running native code.

Further investigations of MetaModelica in Julia will follow, especially with regard to performance.

10 Related Work

OCaml (Minsky, et al, 2013) and Standard ML (Milner et al, 1997) are from the ML family of programming languages. These languages are similar to MetaModelica in that they both use very similar language constructs, statically strong typing and type inference. One major difference is that all variables in MetaModelica have a specific type while in ML each expression has a most general type. MetaModelica can generate error messages that are easier to understand because type inference only has to be performed when calling a polymorphic function. However, this design choice also results in more local variable declarations since all temporary variables need to be declared. This is both positive (one documents what type one expects a variable should have) and negative (one ends up with a number of local variable declarations).

Another group of languages with similar constructs and pattern matching are the dynamically typed languages Lisp, Mathematica, Python, and Julia. Of these, only Julia currently seem to have good enough performance for efficient implementation of core compiler modules. Such dynamic typing is popular for prototyping but is negative from the correctness point of view since certain bugs may remain undetected for a long time and require exhaustive testing for detection. Some languages, like Lisp and Julia, provide meta-programming macros with Quote and Unquote constructs. This enables the use of concrete syntax fragments in meta-programming which may be slightly easier to use that the abstract syntax-oriented approach by the ML languages and MetaModelica, but on the other hand may be less efficiently compilable.

Several authors have used language embedding in a host language for implementing equation-based languages instead of designing a new language such as Modelica or MetaModelica. In this way the concrete and abstract syntax as well as parts of the implementation of the host language can be re-used. On the negative side, one is constrained by the host language regarding expressivity, semantics, and tool facilities (e.g. specific support for small-footprint embedded system code generation recently developed for OpenModelica). Giorgidze and Henrik Nilsson (2011) used this to embed an equation-based language in a functional language, and also used its JIT-compilation facilities for dynamically structure changing models. Erik Frisk (2017) used it for a simple diagnosis equation-based language embedded in Matlab and Python, using the available symbolic toolboxes. Hilding Elmqvist et al (2016) used language embedding of the Modia language prototype into Julia, using meta-programming macros, and also using its JIT-compilation for investigating structure changing models.

11 Conclusions

We have presented the MetaModelica 3.0 language for Modelica-style meta-programming together with its new OMEdit-based development environment. We have also done a short comparison to Julia and conclude that there are many similarities between MetaModelica and Julia. The current OpenModelica environment is the first Modelica environment that integrates meta-programming as well as graphical and textual modeling support and debugging in the same tool. The development environment provides efficient separate compilation with short turn-around time also for applications of several hundred thousand lines of code. Several facilities from the MDT Eclipse plug-in such as go to definition, type, and signature display, are planned to be made available in the new environment. A more efficient compiler frontend is almost completed, as well as a more powerful interface to the OpenModelica code generators. Moreover, further investigation of possible porting of MetaModelica to Julia is planned, which would make possible a Julia-based OpenModelica implementation.

Acknowledgements

This work has been supported by Vinnova in the ITEA OPENPROD, MODRIO, OPENCPS, and EMPHYSIS projects, and in the Vinnova RTISIM project. Support from the Swedish Government has been received from the ELLIIT project, as well as from the European Union in the H2020 INTO-CPS project. The OpenModelica development is supported by the Open Source Modelica Consortium.

References

Patrik Andersson and Simon Eriksson. *Efficient IR for the OpenModelica Compiler*. Maser Thesis report, Linköping University, 202018 | LIU-IDA/LITH-EX-A--2018/001—SE, October 2018.

Peter Aronsson, Peter Fritzson, Levon Saldamli, Peter Bunus and Kaj Nyström. Meta Programming and Function Overloading in OpenModelica. In *Proceedings of the 3rd International Modelica Conference*, Linköping, Sweden, Nov 2003.

Adeel Asghar, Sonia Tariq, Mohsen Torabzadeh-Tari, Peter Fritzson, Adrian Pop, Martin Sjölund, Parham Vasaiely, and Wladimir Schamai. An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation. In *Proc. of the 8th International Modelica Conference* 2011, pp. 739–747. Dresden, Germany, March.20-22, 2011.

Modelica Association. Modelica: A Unified Object-oriented Language for Physical Systems Modeling, Language Specification Version 3.4. May 2017. URL http://www.modelica.org/

Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, Vol. 59, No. 1, pp. 65-98., 2017. http://julialang.org/publications/julia-fresh-approach-BEKS.pdf; see also: http://julialang.org/

Julialang. *Julia Language Documentation*, Release 1.02 Accessed November 14, 2018. www.julialang.org

David Broman and Jeremy Siek J. G. (2012): *Modelyze: a Gradually Typed Host Language for Embedding Equation-Based Modeling Languages*, University of California at Berkeley, No. UCB/EECS-2012-173, 2012. www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-173.html.

Benjamin Chung, Paley Li, and Jan Vitek. Static Typing Without Static Types – Typing Inheritance from the Bottom Up. In *Proc. of 1th Workshop on New Object-Oriented Languages (NOOL) 2016*, In conjunction with ACM SIGPLAN SPLASH Conference, Amsterdam, The Netherlands, October 31, 2017: http://www.it.uu.se/workshop/nool16/nool16-paper4.pdf

Hilding Elmqvist, Toivo Henningsson, and Martin Otter. Innovations for Future Modelica. In *Proc. of Modelica Conference 2017*, Prague, May 15-17, 2017.

Hilding Elmqvist, Toivo Henningsson, and Martin Otter. System Modeling and Programming in a Unified Environment based on Julia. In *Proc of ISoLA 2016*, (Eds) T. Margaria and B. Steffen, Part II, LNCS 9953, pp. 198-217, Oct. 10-14, 2016.

Erik Frisk, Mattias Krysander, and Daniel Jung. A Toolbox for Analysis and Design of Model Based Diagnosis Systems for Large Scale Models. *IFAC World Congress*. Toulouse, France, 2017. https://faultdiagnosistoolbox.github.io/ DOI: https://doi.org/10.1016/j.ifacol.2017.08.504

Fritzson Peter, Adrian Pop, and Peter Aronsson. Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica. In Proceedings of the 4th International Modelica Conference, Hamburg, Germany, March 7-8, 2005

Peter Fritzson, Adrian Pop, and Martin Sjölund. *Towards Modelica 4 Meta-Programming and Language Modeling with MetaModelica 2.0*. Technical reports in Computer and Information Science, No 10, Linköping University Electronic Press. February 2011. URL http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-68361

Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. 1250 pages. ISBN 9781-118-859124, Wiley IEEE Press, 2014.

Peter Fritzson, Adrian Pop, Adeel Asghar, Bernhard Bachmann, Willi Braun, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Alejandro Danós, Rüdiger Franke, Mahder Gebremedhin, Bernt Lie, Alachew Mengist, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, Volker Waurich, Per Östlund. The OpenModelica Integrated Modeling, Simulation, and Optimization Environment. In *Proceedings of the 1st American Modelica Conference*, Cambridge, MA, USA, October, 8-10, 2018. Published by LIU Electronic Press, www.ep.liu.se

George Giorgidze and Henrik Nilsson. Mixed-level Embedding and JIT Compilation for an Iteratively Staged DSL. In Julio Mariño, editor, *Proceedings of the 19th Workshop on Functional and (Constraint) Logic Programming (WFLP 2010)*, volume 6559 of Lecture Notes in Computer Science, pages 48-65, Springer-Verlag, 2011. http://www.cs.nott.ac.uk/~psznhn/Publications/wflp2010-lncs.pdf

Paul Hudak. *The Haskell School of Expression*. Cambridge University Press, 2000.

Kim Jansson and Joel Klinghed. *Incremental compilation and dynamic loading of functions in OpenModelica*. Master's thesis, Linköping University, IDA, June 2008. URN: urn:nbn:se:liu:diva-12329

Bernt Lie, Arunkumar Palanisamy, Alachew Mengist, Lena Buffoni, Martin Sjölund, Adeel Asghar, Adrian Pop, and Peter Fritzson. OMJulia: An OpenModelica API for Julia-Modelica Interaction. In Proc. of the *13th Int. Modelica Conference*, Regensburg, Germany, March 4-6, 2019.

Robin Milner, Mads Tofte, R. Harper, and D. MacQueen, The Definition of Standard ML. MIT Press, Cambridge, MA, USA, 1997.

Yaron Minsky, Anil Madhavepeddy, and Jason Hickey. *Real World OCaml*. O'Reilly, 2013.

Martin Otter and Hilding Elmqvist. Transformation of Differential Algebraic Array Equations to Index One Form. In *Proc. Modelica Conference*, Prague, May 15-17, 2017.

Mikael Pettersson, *Compiling Natural Semantics*. Lecture Notes in Computer Science (LNCS). Vol. 1549. 1999, Springer Verlag.

Adrian Pop, Martin Sjölund, Adeel Asghar, Peter Fritzson, Francesco Casella. Integrated Debugging of Modelica Models. *Modeling, Identication, and Control*, Vol 35, No 2, pp. 93-107, DOI: http://dx.doi.org/10.4173/mic.2014.2.3, ISSN 1890-1328, Aug 2014.

Adrian Pop, Peter Fritzson, Andreas Remar, Elmir Jagudin, and David Akhvlediani. OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In *Proceedings of the 5th International Modelica Conference* (Modelica'2006), Vienna, Austria, Sept. 4-5, 2006.

Adrian Pop and Peter Fritzson, MetaModelica: A Unified Equation-Based Semantical and Mathematical Modeling Language. In D. Lightfoot and C. Szyperski, editors, *Modular Programming Languages*, Vol. 4228 of Lecture Notes in Computer Science, pages 211/229. Springer Berlin / Heidelberg, 2006. DOI:10.1007/11860990_14.

Adrian Pop. *Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages*. Ph.D. Thesis. Linköping Studies in Science and Technology, Dissertation No. 1183, June 5, 2008.

Adrian Pop, Per Östlund, Francesco Casella, Martin Sjölund, Rüdiger Franke. A New OpenModelica Compiler High Performance Frontend. In Proc. of the *13th Int. Modelica Conference*, Regensburg, Germany, March 4-6, 2019.

RelationalAI. *Julia pattern matching Rematch.jl package*, 2018. https://github.com/RelationalAI-oss/Rematch.jl. Accessed Sept. 2018.

Peter van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, 2004.

Tim Sheard. *Accomplishments and Research Challenges in Meta-Programming*. Lecture Notes in Computer Science, 2196:2–.., 2001.

Tom Short. Sims - A Julia package for equation-based modeling and simulations. https://github.com/tshort/Sims.jl 2012.

Martin Sjölund, Peter Fritzson and Adrian Pop. Bootstrapping a Compiler for an Equation-Based Object-Oriented Language. DOI: 10.4173/mic.2014.1.1. *Modeling, Identification and Control*, Vol 35, No 1, pp 1-19, 2014.

Martin Sjölund. *Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models*. Ph.D. Thesis. Linköping Studies in Science and Technology, Dissertation No. 1664, June 1, 2015.

Kevin Squire. *Julia pattern matching Match.jl package*, 2013. https://github.com/kmsquire/Match.jl. Accessed Sept 2018.

Rickard Stallman, R. Pesch, S. Shebs, et al. Debugging with GDB. Free Software Foundation, 2014. URL http://www.gnu.org/software/gdb/documentation/.

Guy Steel and Rickard Gabriel. The Evolution of Lisp. In The second ACM SIGPLAN conference on History of programming languages, HOPLII. ACM, New York, NY, USA, pages 231{270, 1993. doi:10.1145/154766.155373

Bernt Lie, Arunkumar Palanisamy, Alachew Mengist, Lena Buffoni, Martin Sjölund, Adeel Asghar, Adrian Pop, and Peter Fritzson. OMJulia: An OpenModelica API for Julia-Modelica Interaction. In Proc. of the *13th Int. Modelica Conference*, Regensburg, Germany, March 4-6, 2019.

John Tinnerholm. *An LLVM backend for the OpenModelica Compiler*. Master Thesis LIU-IDA/LITH-EX-A--2019/001--SE, Dept. Computer and Information Science, Linköping University, January 2019.

Stephen Wolfram. *The Mathematica Book*, 5th Ed. Wolfram Media, Inc, 2003.

Dirk Zimmer. Equation-Based Modeling of Variable Structure Systems. PhD Dissertation, ETH Zürich. http://ecollection.library.ethz.ch/eserv/eth:1512/eth-1512-02.pdf

Controller Design for a Magnetic Levitation Kit using OpenModelica's Integration with the Julia Language

Bernhard Thiele[1] Bernt Lie[2] Martin Sjölund[3] Adrian Pop[3] Peter Fritzson[3]

[1]Institute of System Dynamics and Control, DLR, Germany, bernhard.thiele@dlr.de
[2]University of South-Eastern Norway, bernt.lie@usn.no
[3]PELAB, Linköping University, Sweden, {martin.sjolund,adrian.pop,peter.fritzson}@liu.se

Abstract

This paper presents a practical application of computer aided control systems design using a new OpenModelica API (OMJulia) which allows to conveniently operate on Modelica models from the Julia language. Julia is a rather young language (Julia 1.0 was released in August 2018) designed to address the needs of numerical analysis and computational science, in particular it already has decent support for the control community. The magnetic levitation application at hand demonstrates how control system design can benefit from a suitable integration between Julia and Modelica. It is based on a commercially available control education kit in which the original controller is replaced by our own digital controller developed in this work. There exists an accompanying but independent paper which introduces the complete OMJulia API.

Keywords: OpenModelica, OMJulia, control, magnetic levitation, Arduino, Julia, Modelica

1 Introduction

Modelica is a well established language for modeling complex technical systems supported by several convenient and powerful modeling and simulation environments. Distinguishing language characteristics are the focus on declarative system descriptions using mathematical equations and a specific approach to object orientation which allows encapsulating component behavior (given by data + equations) into reusable units which can be connected by suitable constraints (connect equations) to build complex systems from manageable building blocks, see e.g., (Modelica Association, 2017; Fritzson, 2015).

However, for many numerical analysis tasks an imperative language is well suited and suggests itself. Indeed, the most prevalent software for computer aided control systems design on the market, MATLAB/Simulink[1], has two parts: MATLAB, a numerical computing environment built around the imperative MATLAB scripting language, and Simulink, a primarily graphical block diagram language which is tightly coupled to MATLAB, for modeling and simulation.

Although the Modelica language also has an imperative part for writing algorithms, its support in tools as scripting language has so far remained limited and rather tool specific. Consequently, no rich ecosystem for typical numerical computing tasks like data analysis and advanced data visualization was developed within the community. There are notable exceptions like the LinearSystems library for linear system analysis and controller design (Baur et al., 2009). However, as of the latest release of the library (Modelica_LinearSystems2 v2.3.4[2]) full support of the library is still limited to the Dymola[3] tool.

OpenModelica[4], similarly to other Modelica tools, provides interfaces to dedicated scripting languages which provide the desired advanced scripting support, inclusive a rich ecosystem for numerical analysis and advanced visualization. Based on OMPython (Ganeson, 2012; Ganeson et al., 2012) an API was developed for simple operation on Modelica models from within Python (Lie et al., 2016). However, in the meantime the rather young language Julia[5] has matured (Julia 1.0 was released in August 2018) and has attracted a growing user base in the scientific computing community. The Julia language was originally designed to address the needs of numerical analysis and computational science, in particular it already has decent support for the control community. This motivated the development of OMJulia, an API for interacting with Modelica models from the Julia language. The OMJulia API is described in detail in an accompanying but independent paper (Lie et al., 2019).

The goal of this paper is to demonstrate the interaction between Julia and Modelica models using one of the most popular applications in control education: A *magnetic levitation system*; see, e.g., (Yoon and Moon, 2016; Lilienkamp and Lundberg, 2004; Craig et al., 1988; Wong, 1986). The intention is to present available tool support using a tangible example, it is *not* in the scope of the paper to propose and validate a controller that improves on existing designs.

[1]The MathWorks, https://mathworks.com.

[2]Modelica_LinearSystems2 library, https://github.com/modelica/Modelica_LinearSystems2.

[3]Dassault Systèmes, https://www.3ds.com.

[4]Open Source Modelica Consortium (OSMC), https://www.openmodelica.org.

[5]Julia language, https://julialang.org.

2 Digital Control for a Magnetic Levitation Kit

Magnetic levitation is a popular application for teaching control theory. A levitating object which apparently defies the law of gravity is an attractive gadget and the underlying physics (unstable plant dynamics) convincingly demonstrate the importance of feedback control. The application is based on a commercially available electromagnetic levitation kit[6] from Zeltom which is targeted at educational applications. The fully assembled unit is shown in Figure 1. The vertical position of the levitating magnet

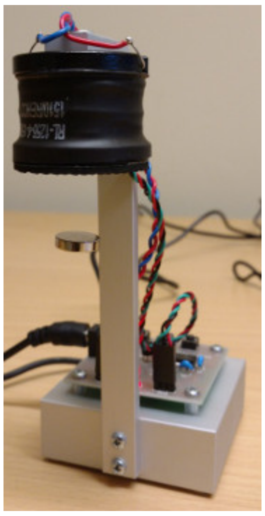


Figure 1. Zeltom's electromagnetic levitation kit.

is measured using a linear Hall effect sensor which is directly attached below the electromagnet. The kit includes a black box microcontroller for controlling the current in the electromagnet.

The goal is to replace Zeltom's controller by our own design.

3 Plant Model

A behavioral model describing the dynamics of the physical system is provided in a technical report by Zeltom (Zeltom LLC, 2009). A schematic diagram of the system is shown in Figure 2, where v is the voltage across the electromagnet, i is the current flowing through the electromagnet, R is the resistance and L the inductance of the electromagnet, e is the voltage across the Hall effect sensor, d is the distance between the Hall sensor and the levitating magnet, m is the mass of the levitating magnet, and f is the force on the levitating magnet generated by the electromagnet.

The nonlinear dynamic equations as described in (Zeltom LLC, 2009) are reproduced below.

[6]Zeltom Electromagnetic Levitation System, `http://zeltom.com/product/magneticlevitation`.

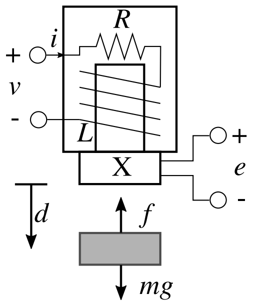


Figure 2. Schematic of the magnetic levitation system.

Approximated force from the electromagnet on the levitating magnet:

$$f = k\frac{i}{d^4}, \tag{1}$$

approximated voltage across the Hall effect sensor:

$$e = \alpha + \beta\frac{1}{d^2} + \gamma i, \tag{2}$$

Newton's second law:

$$m\frac{\mathrm{d}^2 d}{\mathrm{d}t^2} = mg - f, \tag{3}$$

Kirchhoff's voltage Law:

$$v = Ri + L\frac{\mathrm{d}i}{\mathrm{d}t}, \tag{4}$$

where k is a geometry dependent constant, α, β, γ are constants that depend on the Hall sensor and the geometry, and g is the standard gravity constant. The system's parameters are listed in Table 1; the values are from (Zeltom LLC, 2009) and from own measurements.

Table 1. System parameters.

Parameter	*Value*	*Unit*
k	$17.31 \cdot 10^{-9}$	$\mathrm{kg \cdot m^5/A \cdot s^2}$
α	2.44	V
β	$1.12 \cdot 10^{-4}$	$\mathrm{V \cdot m^2}$
γ	0.26	V/A
R	2.41	Ω
L	$15.03 \cdot 10^{-3}$	H
m	$3.02 \cdot 10^{-3}$	kg

Letting v be the control input and e be the measured output, these nonlinear equations can be readily transcribed into a Modelica model (condensed for saving space):

```
model MagLevNL
  parameter Real R=2.41, L=15.03e-3,
      m=3.02e-3, k=17.31e-9, alpha=2.44,
      beta=1.12e-4, gamma=0.26;
  input Real v;
  output Real e;
  Real i, d, d_der, f;
  constant Real g=9.81;
equation
  f = k*i/d^4;
  e = alpha + beta/d^2 + gamma*i;
  der(d) = d_der;
  m*der(d_der) = m*g - f;
  v = R*i + L*der(i);
end MagLevNL;
```

For the purpose of controller design it is typically necessary to work with a linearized version of the plant dynamics. The goal for the magnetic levitation system is to design a controller which stabilizes the plant in an equilibrium position. Therefore, the system needs to be linearized around an equilibrium position of the nonlinear plant. Hence, the first step is to determine an equilibrium position. It would be convenient to have a direct OMJulia API function for this task, similar to

```
mlNL = OMJulia.OMCSession()
mlNL.ModelicaSystem("MagLevNL.mo",
    "MagLevNL")
state_e, u_e, y_e =
    mlNL.findEquilibrium(["d=0.02",
    "d_der=0"])
```

where `findEquilibrium(..)` would search for an equilibrium position under constraints that can be set as function arguments. The function would return the value of the state variables, as well as the value of the inputs and outputs at the equilibrium position. Here, an equilibrium is sought under the constraints that the levitating magnet, levitates at a distance of 2 cm below the sensor.

Unfortunately, such a function is not (yet) available in OMJulia[7]. However, it is possible to modify the Modelica model and impose the equilibrium constraints within a steady-state initialization problem as shown in the listing below. Notice that `input v` was turned into a parameter with unknown value (`fixed=false`) which has the effect that the value is determined during initialization[8]. This is needed since in Modelica a variable which is declared as input is treated as a known, which would result in an overspecified initialization problem below. In order to search for the (unknown) voltage `input` at which the system stays at an equilibrium with the prescribed constraints the Modelica tool needs to treat the voltage input as an unknown.

```
model MagLevNL_SteadyState
  parameter Real R=2.41, L=15.03e-3,
      m=3.02e-3, k=17.31e-9, alpha=2.44,
      beta=1.12e-4, gamma=0.26;
  parameter Real d0 = 0.02 "Prescribed
      equilibrium position";
  parameter Real v(start=0.5, fixed=false)
      "Unknown equilibrium voltage across
      the electromagnet";
  output Real e;
  Real i, d, d_der, f;
  constant Real g=9.81;
equation
  f = k*i/d^4;
  e = alpha + beta*1/d^2 + gamma*i;
  der(d) = d_der;
  m*der(d_der) = m*g - f;
  v = R*i + L*der(i);
initial equation
  d = d0;
  der(d) = 0;
  der(d_der) = 0;
  der(i) = 0;
end MagLevNL_SteadyState;
```

With this model the OMJulia API can be used to retrieve the plant's values at the equilibrium position and use them for linearizing the plant at this equilibrium position. Since the OMJulia API does not (yet) allow to conveniently set start values, the following small modification to the MagLevNL model is introduced, in order to set the start values as parameters:

```
model MagLevNL
  // ... same as previously
  parameter Real i0, d0, d_der0;
  Real i(start=i0,fixed=true),
      d(start=d0,fixed=true),
      d_der(start=d_der0,fixed=true), f;
  // ... same as previously
end MagLevNL;
```

Using this modified model the OMJulia API allows to retrieve the linearized representation of the plant model as shown in the listing below.

```
mlNLe = OMJulia.OMCSession()
mlNLe.ModelicaSystem(
    "MagLevNL_SteadyState.mo",
    "MagLevNL_SteadyState")
mlNLe.setParameters(["d0=0.02"])
mlNLe.simulate()
sol = mlNLe.getSolutions(["v", "i", "d",
    "d_der"])
v_e = sol[1][1]  # input v at equilibrium
i_e = sol[2][1]  # state i at equilibrium
d_e = sol[3][1]  # must be equal to d0
d_der_e = sol[4][1]  # must be 0

mlNL = OMJulia.OMCSession()
mlNL.ModelicaSystem("MagLevNL.mo",
    "MagLevNL")
mlNL.setInputs(["v=$v_e"])
mlNL.setParameters(["i0=$i_e", "d0=$d_e",
    "d_der0=$d_der_e"])
A,B,C,D = mlNL.linearize()
```

The final call to the `linearize()` function retrieves a tuple of 2D arrays (matrices) which encode the linearized model in a state space representation ($\dot{x} = Ax + Bu, y = Cx + Du$). The values can be easily inspected, e.g., by

[7] Tools like Wolfram Mathematica (Wolfram Research) or Maple (MapleSoft) support functions for finding local equilibrium points of nonlinear systems. For example Wolfram Mathematica 11.3 introduced a function named "FindSystemModelEquilibrium" which works with (imported) Modelica models and provides respective functionality.

[8] Alternatively, it is possible to declare v as "`Real v(start=0.5, fixed=false)`" and add an equation "`der(v) = 0`".

printing them to the console window (number of digits truncated for readability):

```
julia> println("v_e=$v_e, i_e=$i_e,
    e_e=$e_e")$
v_e=0.66, i_e=0.27, e_e=2.79
julia> println("A=$A\nB=$B\nC=$C\nD=$D")
A=[0.0 1.0 0.0; 1962.0 0.0 -35.8237; 0.0
    0.0 -160.346]
B=[0.0; 0.0; 66.5336]
C=[-28.0 0.0 0.26]
D=[0.0]
```

4 Control Design

The Julia ecosystem provides various packages which can support a control design process. The OMJulia bridge to OpenModelica allows to combine the strength of those packages with the powerful modeling and simulation infrastructure of a Modelica tool. This section will demonstrate some possibilities.

The magnetic levitation system is open-loop unstable, which can be quickly checked using the ControlSystems.jl package[9]. Function `mlLin=ss(A, B, C, D)` creates a state-space model from the previously retrieved matrices of the linearized magnetic levitation model. Function `pole(mlLin)` returns its poles.

```
julia> using ControlSystems
julia> mlLin = ss(A,B,C,D)
julia> pole(mlLin)
3-element Array{Float64,1}:
    44.294469180700204   -44.2944691807002
       -160.346
```

It is known that a PD controller is capable of stabilizing a magnetic levitation system. Indeed, Yoon and Moon have shown in (Yoon and Moon, 2016) that the system at hand can be stabilized by a simple PD analog controller. A particular challenge in respect to stabilizing a magnetic levitation system is designing a reasonable *robust* controller. A measure for the robustness of a design is the sensitivity function S, which describes the transfer function from an external disturbance to the process output. Lower values of $|S|$ suggest further attenuation of the external disturbance (hence, lower is better). The following paragraphs briefly introduces the Julia code used for sensitivity analysis of the controlled system.

A PD controller is described by the transfer function

$$C_{PD}(s) = K_p(T_d s + 1), \tag{5}$$

where K_p is the proportional gain, and T_d is the derivative time parameter. The listing below uses the construct `s = tf("s")` to create a continuous-time transfer function `s`, which enables a convenient notation for creating transfer functions using standard mathematical operators like `PD = Kp*(Td*s + 1)`.

[9]ControlSystems.jl, https://github.com/JuliaControl/ControlSystems.jl.

The plant's state space representation from above can be converted into a transfer function representation $G(s)$ using function `tf(..)`, e.g., `G = tf(mlLin)`. Using the plant's transfer function $G(s)$, the open-loop transfer function is given by a serial connection of controller and plant,

$$P_{PDol}(s) = C_{PD}(s)G(s), \tag{6}$$

which can be achieved using the `series(..)` function in Julia. The sensitivity function is then given by

$$S(s) = \frac{1}{1 + P_{PDol}(s)}. \tag{7}$$

Since poles are not canceled automatically, the function `minreal(..)`[10] is used to obtain a minimal transfer function representation.

Function `bodeplot(..)` is used for plotting the magnitude of the sensitivity function. Using the Julia Interact.jl package[11] together with functions from ControlSystems.jl allows for interactive plots in which the controller's parameters can be tuned experimentally. The Interact package provides means to create small GUIs in Julia based on web technology. It defines the macro `@manipulate` which sets up sliders for varying the parameters within the specified range.

```
using Interact
s = tf("s")
@manipulate for Kp=3:.5:20, Td=0.01:.01:0.1
  PD = Kp*(Td*s + 1)
  mlLinPDol = series(PD,tf(mlLin))
  mlLinPDSensitivity =
      minreal(1/(1+mlLinPDol))
  bodeplot(mlLinPDSensitivity,
      plotphase=false, yscale=:identity,
      yticks=0:0.1:2, title="Sensitivity")
end
```

Evaluating the above code in an IJulia/Jupyter session gives a result as depicted in Figure 3. The two sliders at the top allow to change the PD controller's parameters. When the parameters are changed, the plot is immediately updated.

5 Nonlinear Closed-Loop Model

After an acceptable design (based on the linearized model) has been found, the controller can be tested and further tuned by plugging it into the nonlinear Modelica model.

In the present example the PD controller can be easily transcribed into Modelica code and can be added appropriately to the MagLevNL model in order to close the loop between controller and plant. Let the resulting model be named “MagLevNLPD” (the complete listing is given in Appendix A).

[10]Function `minreal(..)` creates a minimal transfer function representation by canceling pole-zero pairs.

[11]Interact.jl, https://github.com/JuliaGizmos/Interact.jl.

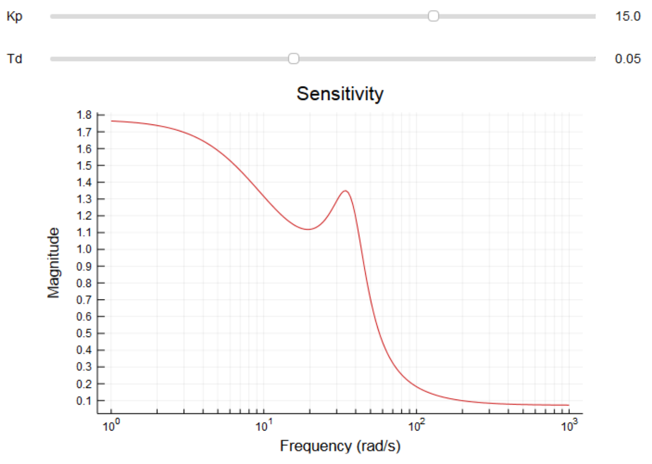


Figure 3. Interactive sensitivity plot for the magnetic levitation system in which the controller parameters can be varied using sliders.

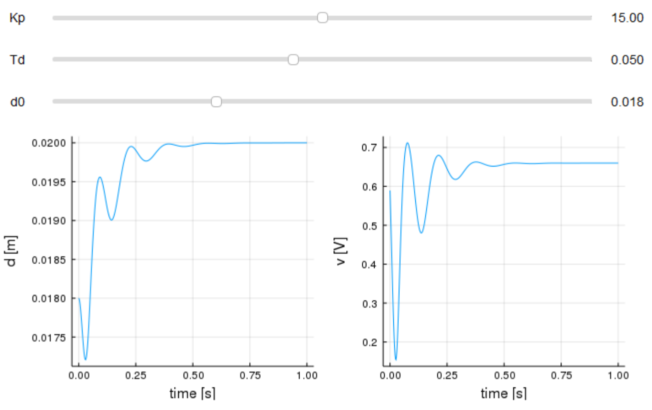


Figure 4. Simple interactive GUI with sliders for setting parameters of the closed-loop nonlinear magnetic levitation Modelica model using the OMJulia interface. Changing a slider will immediatly trigger a new simulation and update the plots.

Combining OMJulia with the Interact package allows to quickly create small GUIs for interactive experimentation with a Modelica model. The Julia code below creates sliders for varying the controller parameters, as well as to vary the initial distance d_0 of the levitating magnet. Since the controller is designed for keeping an equilibrium position at $d = 0.02\,\text{m}$, it is interesting to explore how the closed system behaves for small displacements, where $d_0 \neq 0.02\,\text{m}$.

```
using OMJulia, Plots, Interact
mlNLPD = OMJulia.OMCSession()
mlNLPD.ModelicaSystem("MagLevNLPD.mo",
    "MagLevNLPD")
@manipulate for Kp=7:0.5:23,
    Td=0.01:0.01:0.1, d0=0.015:0.0002:0.025
  mlNLPD.setParameters(["Kp=$Kp",
      "Td=$Td", "d0=$d0"])
  mlNLPD.simulate()
  sol = mlNLPD.getSolutions(["time", "d",
      "v"])
  time, d, v = sol[1], sol[2], sol[3]
  p1 = plot(time, d, label="",
      xlabel="time [s]", ylabel="d [m]")
  p2 = plot(time, v, label="",
      xlabel="time [s]", ylabel="v [V]")
  plot(p1, p2, layout=(1,2))
end
```

Figure 4 shows a screenshot of the resulting GUI when evaluating the above code in an IJulia/Jupyter session. The start value of d is set to $d_0 = 18\,\text{mm}$, hence two millimeters closer to the electromagnet than the set reference distance of 20 millimeters. The left plot shows how the distance d starts at the prescribed start value and is regulated to the reference distance of 20 millimeters. The right plot shows the voltage v (the actuating variable) that the controller sets to the electromagnetic actuator. Notice that the voltage remains in reasonable limits (no actuator saturation). However, further exploration (using the same controller parameters) showed that the closed loop stabilization for the nonlinear model fails quickly when choosing start values d_0 which are greater than the reference distance of 20 millimeters.

6 Digital Control

For a practical implementation of the presented PD controller, the derivative "D" part is first approximated by a "DT$_1$" element before the controller is discretized in a second step. Finally, hardware characteristics of the target controller are considered in a nonlinear closed-loop, sampled-data model.

The "D" part can be approximated by $sT_d \approx \frac{sT_d}{1+sT_d/N_d}$, where N_d limits the gain at high frequencies (typically: $3 \leq N_d \leq 20$). Therefore, the structure of the controller becomes

$$C_{PDT_1}(s) = K_P\left(\frac{T_d s}{\frac{T_d}{N_d}s+1}+1\right). \tag{8}$$

6.1 Discrete-Time Approximation

Using backward differences for approximation, transfer function (8) can be transformed into a pulse-transfer function by substituting s by s' using the formula

$$s' = \frac{z-1}{zh}, \tag{9}$$

where h is the sampling period and z is the Z-transform variable, resulting in the pulse-transfer function

$$C_{PDT_1}(z) = K_p\left(\frac{T_d N_d(z-1)}{(T_d+N_d h)z-T_d}+1\right). \tag{10}$$

The pulse-transfer function can be readily transformed into a recurrance relation which directly translates into Modelica code. The listing below shows a condensed version of the discretized controller using Modelica's clocked synchronous language elements.

```
block Controller
  parameter Real Kp=15, Td=0.05, Nd=5,
      h=0.0005, v_e=0.66, e_e=2.79;
  input Real du_set "Setpoint delta
      voltage (=0 for d=>0.02)";
  input Real e "Measured voltage across
      the Hall effect sensor";
  output Real v "Output voltage to the
      electromagnet";
protected
  Real Dpart(start=0), de_e, du(start=0),
      dy, ad, bd;
equation
  // Measured delta voltage at OP
  de_e = e - e_e;
  // input to PD(T1) control law
  du = du_set - de_e;

  // Control law
  ad = Td/((Td + Nd*h));
  bd = Td*Nd/(Td + Nd*h);
  Dpart =  ad * previous(Dpart) + bd * (du
      - previous(du));
  dy = Kp*(du + Dpart);

  // Output voltage to electromagnet
  v = dy + v_e;
end Controller;
```

6.2 Target Hardware

The popular Arduino Uno board[12] is used as implementation hardware for the control algorithm. It is based on the Microchip ATmega328P microcontroller, has six analog inputs supporting 10-bit analog-to-digital conversion (ADC) for input voltages between zero and five volts, and 14 digital input/output pins of which six can be used as pulse-wide modulation (PWM) outputs. The frequency of the PWM outputs is configurable and a simple interface exists in which the PWM duty cycle can be set with a resolution of 8-bit.

In our application the voltage across the Hall effect sensor is read using one of the analog inputs. The voltage to the electromagnet is set by a PWM output driving a MOSFET which is connected to a DC voltage regulator fed from an external power supply. A breadboard is used for the implementation of the supporting electronics (see Figure 8).

6.3 Sampled-Data Model

A model of the closed-loop, sampled-data system can be built conveniently with the help of the Synchronous library (Otter et al., 2012). Figure 5 shows a diagram view in which the nonlinear continuous-time magnetic levitation plant model is connected to the discrete-time (clocked) controller model using sample and hold blocks from the Synchronous library. The controller is activated by a periodic clock with a sampling period of 500 μs. The upper controller input specifies the setpoint of the controller. The setpoint is 0 V for the equilibrium position

[12]Arduino, `https://arduino.cc`.

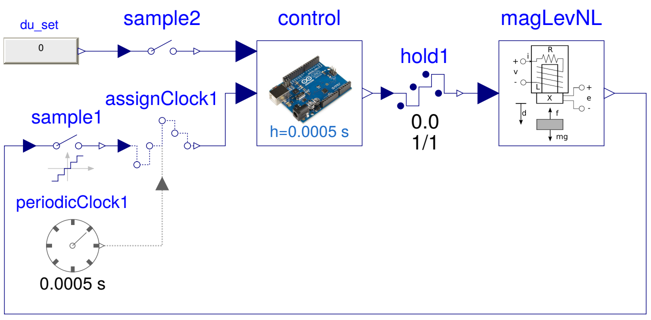


Figure 5. Closed-loop magnetic levitation system with clocked controller model.

for which the controller is designed (i.e., in the presented design the levitating object is at $d = d_e = 0.02\,\mathrm{m}$, the Hall effect sensor output is $e = e_e = 2.79\,\mathrm{V}$). Modifying the setpoint allows to influence the position of the levitating magnet.

The utilized sample and hold blocks allow modeling additional real-world effects like noise, quantization effects of digital-analog and analog-digital conversions, sensor and actuator limitations, and computational delays. In the displayed model the sample and hold blocks are parametrized so that they reflect the capabilities of the target hardware as described in Section 6.2. The measurement variable e is limited between $0\,\mathrm{V} \leq e \leq 5\,\mathrm{V}$ using 10-bit quantization. The actuating variable v is limited between $0\,\mathrm{V} \leq v \leq 1.3\,\mathrm{V}$ using 8-bit quantization.

Simulating the sampled-data model given above using the same scenario as for the nonlinear continuous-time model in Section 5 reveals a severe control degradation for the considered digital controller. Further simulation experiments reveal that this is mainly due to ADC quantization effects of the Hall effect sensor output. Figure 6 shows results plots[13] for simulating with different ADC settings, while the other settings, e.g., computational delay of one sampling period and 8-bit quantization of the actuating variable, are unchanged. The upper plot shows the distance d of the levitating object. The two lower plots show the sampled and quantized Hall sensor output e (= `sample1.y`) and the quantized actuating variable v (= `hold1.y`) in a narrow time window ($t \in [4.00\,\mathrm{s}, 4.03\,\mathrm{s}]$).

Although the levitating object can be stabilized in all simulated cases, it shows persisting oscillations for the case of an ADC with 10-bit resolution over the range $[0\,\mathrm{V}, 5\,\mathrm{V}]$. For this setting, the actuating variable exhibits large, high frequency oscillations. Increasing the quantization resolution mitigates this adverse effect and restores

[13] Apart from using OMJulia for controlling the complete simulation (as shown in Section 5), it is also possible to use the Julia CSV package for simply importing an OpenModelica (CSV-) result file into Julia for postprocessing. For example, plotting variable `magLevNL.d` from a CSV-result file can be achieved by:

```
using Plots, CSV
r = CSV.read("myresultfile.csv")
plot(r[Symbol("time")], r[Symbol("magLevNL.d")])
```

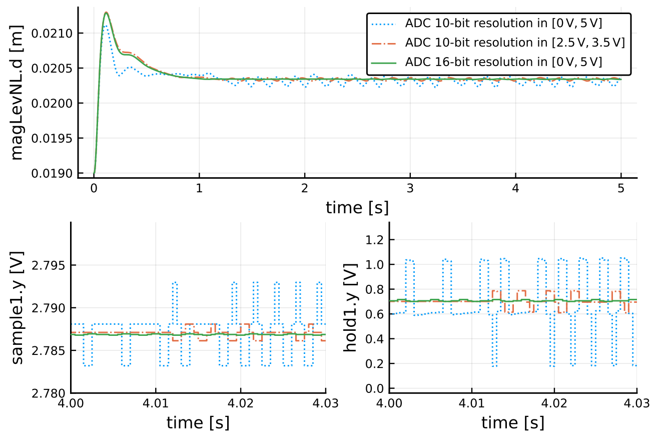


Figure 6. Simulation results of the sampled-data model for different ADC quantization settings and an initial distance $d_0 = 0.019\,\text{m}$.

a behaviour which is closer to the continuous-time controller. Besides increasing the ADC resolution (e.g., to 16-bit), the simulation results suggest that a 10-bit ADC resolution is fine, if it is available within the (smaller) relevant operating range of the sensor, e.g., $[2.5\,\text{V}, 3.5\,\text{V}]$. This can be achieved by using a suitable signal conditioning circuit for mapping the signal's operating range to the full-scale voltage range of the ADC.

6.4 Real-Time Target Code

In a first approach, the Modelica code for the demonstrator presented in Section 7 was hand translated to C in order to compile and upload it to Arduino. This is rather straightforward, since the control algorithm is short and the Arduino environment is easy to use.

However, particularly for more complex models, it would be beneficial to automatically generate the target code, instead of manually converting the controller models to compact C code. This is quicker and less error prone than manual translation. One big challenge is to produce target code that fits into very small foot-print platforms.

For these reasons we have developed an experimental version of an embedded target simple code generator[14] for OpenModelica aimed at very restricted platforms such as the Atmel AVR 8-bit microcontrollers. The regular C-code generator creates huge data structures and contains much debugging information while the run-time system contains many numerical solvers and is around 6 MB in size (of which 0.5 MB is textual strings for error messages). This regular C-code is intended to run on powerful desktop CPUs where the code size does not matter much and it proved difficult to try to strip out unnecessary code when targeting embedded systems. The largest of the 8-bit AVR processor MCUs (Micro Controller Units) have 16 kB SRAM. One of the smaller ones (ATmega328P; Arduino Uno) has 2 kB SRAM.

[14]The embedded code generation target for OpenModelica can be activated by passing the option `--simCodeTarget=ExperimentalEmbeddedC` to the OMC.

The embedded target code generator was designed to generate code for constructs that are easy to compile. For example, it does not support arrays, strongly connected components, or initialization, but still works fine for many models since the OpenModelica compiler will convert many complex constructs into simpler ones during the compilation process, e.g., make array equations into scalar equations. Instead of having a big run-time system that is linked in (as is the case for the regular code generator), the code generator will generate the needed C-functions corresponding to the Modelica and run-time functions called.

As can be seen from Table 2, the experiment was so far successful. The regular stripped-down code generator

Table 2. Code generator comparison. Regular vs Simple.

	Regular stripped-down source-code FMU targeting 8-bit AVR processor	*Simple code generator targeting 8-bit AVR processor*
Minimal model (0 equations)	43 kB flash memory, 23 kB variables (RAM)	130 B flash memory, 0 B variables (RAM)
Target system including controller	68 kB flash memory, 25 kB variables (RAM)	3350 B flash memory, 169 B variables (RAM)

with almost everything stripped out except the main simulation loop (it includes no solvers or numerical routines except the used ones) already reduces the code foot-print significantly compared to the standard desktop version. However, it is still too large for very small foot-print platforms like the Arduino Uno. The simple code generator allows a further reduction in size which makes it suitable for very small foot-print platforms.

The clocked controller model from Figure 5 needs to be adapted in order to be suited as input to the experimental embedded target code generator. The embedded target code generator in the development branch for the upcoming OpenModelica v1.14 release does not yet support the synchronous clocked language elements, nor does it support when-equations for modeling sampled systems. As a workaround the clocked controller equations can be rewritten as an algorithm and placed into an algorithm section. In the generated code this algorithm section is called periodically using a base rate which can be specified during translation[15].

Figure 7 shows the input model for the code generator. The model uses blocks for interfacing to hardware facilities of the Arduino Uno like ADC or PWM units. These hardware interface blocks are available in the Modelica_DeviceDrivers library (Thiele et al., 2017). The model assumes that a signal conditioning circuit is used for mapping the Hall effect sensor voltage around the op-

[15]For example, when using OpenModelica's scripting interface the base period can be specified by providing the `stepSize` argument to the `translateModel(..)` function.

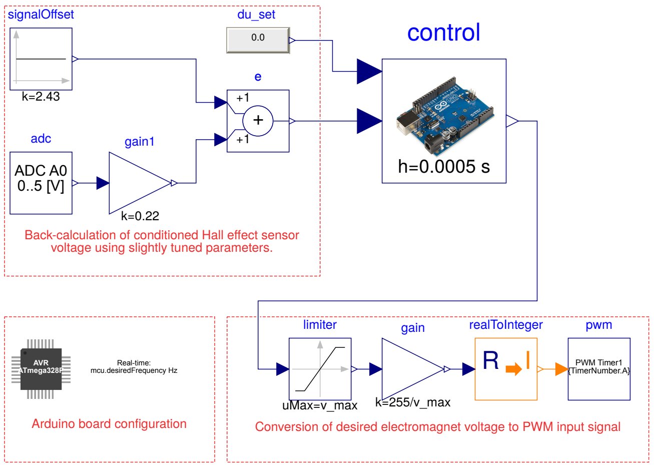

signalOffset
du_set
0.0
control
k=2.43
e
+1
+
+1
adc
gain1
ADC A0
0..5 [V]
k=0.22
h=0.0005 s
Back-calculation of conditioned Hall effect sensor voltage using slightly tuned parameters.
AVR ATmega328P
Real-time: mcu.desiredFrequency Hz
Arduino board configuration
limiter
gain
realToInteger
pwm
uMax=v_max
k=255/v_max
PWM Timer1 (TimerNumber.A)
Conversion of desired electromagnet voltage to PWM input signal

Figure 7. The input model for the code generator consisting of the control algorithm and hardware related blocks.

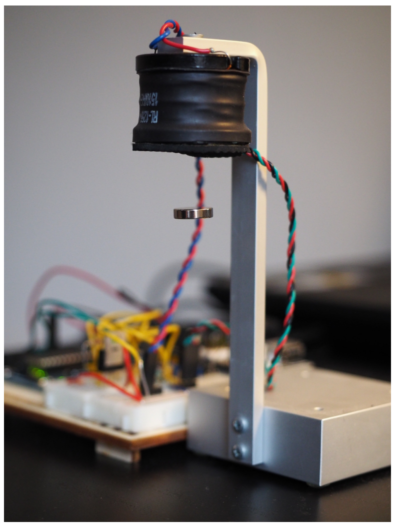


Figure 8. Arduino controlled electromagnetic levitation system.

erating point to the full-scale voltage range of the ADC. Notice that the parameters for back-calculation of the conditioned Hall effect sensor signal deviate slightly from the theoretical values ($k = 2.5$ for the signal offset and $k = 0.2$ for the signal gain). This is the result of tuning the parameters for the actual demonstrator with its (non-ideal) supporting electronics.

Most of the parameters in the hardware interface blocks are at their default values. However, several interesting parameter settings are not visible on the diagram layer. The microcontroller block is set to the ATmega328P platform and its internal parameter `desiredPeriod` is set to 0.0005 s. The real-time block is configured to use `Timer0` for the real-time synchronization. The PWM block is configured to use `Timer1` with a prescaler value of “1/8”.

7 Demonstrator

Figure 8 shows a setup in which Zeltom’s controller has been replaced by an Arduino Uno and supporting electronics.

It was possible to stabilize the levitating mass for several minutes at a time using the presented controller and the experimental hardware setup with a 10-bit ADC resolution in the range of $[0\,\mathrm{V}, 5\,\mathrm{V}]$, but the magnet showed clearly visible oscillations around the equilibrium position and was very sensitive to disturbances, e.g, a tiny push against the table would destabilize the mass instantly. Motivated from the simulation results in Figure 6, a signal conditioning circuit based on Texas Instrument’s INA333 instrumentation amplifier was developed to map the operating range $[2.5\,\mathrm{V}, 3.5\,\mathrm{V}]$ of the Hall effect sensor signal to the full-scale voltage range of the ADC. As suggested by the simulation results, this attenuated the oscillation and lead to a greatly improved robustness in maintaining the equilibrium position.

8 Conclusion

The paper shows how computer aided control system design based on Modelica models can benefit from the new OpenModelica OMJulia API which allows joint interaction between the Modelica and Julia ecosystems. For practical illustration a complete magnetic levitation application is presented with sufficient details so that the example can be readily reproduced, e.g., in the context of a lab session in control education.

While Modelica excels in modeling and simulation of complex technical systems, Julia can provide the numerical analysis, optimization and advanced visualization capabilities, including specialized packages for control engineering. Simple web technology based GUIs can be created in Julia in just a few lines of code, which allows interactive experimentation with Modelica simulation models giving immediate feedback to the user, e.g., by updating key performance plots. The magnetic levitation application aims at illustrating how a carefully designed API has the potential to leverage attractive synergies between the two languages.

Acknowledgements

This work has been supported by Vinnova in the ITEA OPENCPS and EMPHYSIS projects, and in the Vinnova RTISIM project. Support from the Swedish Government has been received from the ELLIIT project. The OpenModelica development is supported by the Open Source Modelica Consortium.

References

Marcus Baur, Martin Otter, and Bernhard Thiele. Modelica Libraries for Linear Control Systems. In Francesco Casella, editor, 7th *Int. Modelica Conference*, Como, Italy, September 2009. doi:10.3384/ecp09430068.

Kevin Craig, Thomas Kurfess, and Mark Nagurka. Magenetic levitation testbed for controls eduction. In *Proceedings of the ASME Dynamic Systems and Control Division*, volume 64, 1988.

Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach.* Wiley-IEEE Press, Piscataway, NJ, second edition, 2015. ISBN 978-1-118-85912-4.

Anand Ganeson. Design and Implementation of a User Friendly OpenModelica - Python interface. Master's thesis, Linköping University, 2012.

Anand Ganeson, Peter Fritzson, Olena Rogovchenko, Adeel. Asghar, Martin Sjölund, and Andreas Pfeiffer. An OpenModelica Python Interface and its Use in PySimulator. In Martin Otter and Dirk Zimmer, editors, 9th *Int. Modelica Conference*, Munich, Germany, September 2012. doi:10.3384/ecp12076537.

Bernt Lie, Sudeep Bajracharya, Alachew Mengist, Lena Buffoni, Arunkumar Palanisamy, Martin Sjölund, Adeel Asghar, Adrian Pop, and Peter Fritzson. API for Accessing OpenModelica Models from Python. In *Proceedings of EuroSim 2016*, Oulu, Finland, September 2016.

Bernt Lie, Arunkumar Palanisamy, Alachew Mengist, Lena Buffoni, Martin Sjölund, Adeel Asghar, Adrian Pop, and Peter Fritzson. OMJulia: An OpenModelica API for Julia-Modelica Interaction. In Anton Haumer, editor, 13th *Int. Modelica Conference*, Regensburg, Germany, March 2019.

Katie A. Lilienkamp and Kent Lundberg. Low-cost magnetic levitation project kits for teaching feedback system design. In *Proceedings of the 2004 American Control Conference*, volume 2, pages 1308–1313 vol.2, June 2004. doi:10.23919/ACC.2004.1386755.

Modelica Association. Modelica - A Unified Object-Oriented Language for Systems Modeling - Version 3.4. Standard Specification, April 2017. URL http://www.modelica.org/.

Martin Otter, Bernhard Thiele, and Hilding Elmqvist. A Library for Synchronous Control Systems in Modelica. In Martin Otter and Dirk Zimmer, editors, 9th *Int. Modelica Conference*, Munich, Germany, September 2012. doi:10.3384/ecp1207627.

Bernhard Thiele, Thomas Beutlich, Volker Waurich, Martin Sjölund, and Tobias Bellmann. Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library. In Jiří Kofránek and Francesco Casella, editors, 12th *Int. Modelica Conference*, Prague, Czech Republic, May 2017. doi:10.3384/ecp17132713.

T. H. Wong. Design of a Magnetic Levitation Control System - An Undergraduate Project. *IEEE Transactions on Education*, E-29(4):196–200, Nov 1986. ISSN 0018-9359. doi:10.1109/TE.1986.5570565.

Myung-Gon Yoon and Jung-Ho Moon. A Simple Analog Controller for a Magnetic Levitation Kit. *International Journal of Engineering Research & Technology (IJERT)*, 5(3):94–97, March 2016.

Zeltom LLC. Electromagnetic Levitation System - Mathematical Model, June 2009. URL http://zeltom.com/documents/emls_md.pdf.

A Listing of the Nonlinear Closed-Loop MagLev Model

The complete listing of the nonlinear closed-loop Modelica model used in Section 5.

```
model MagLevNLPD
  // Parameters MagLev
  parameter Real R=2.41, L=15.03e-3,
      m=3.02e-3, k=17.31e-9, alpha=2.44,
      beta=1.12e-4, gamma=0.26;
  // Equilibrium point (values actually
      depend on parameters above!)
  parameter Real v_e=0.659957,
      e_e=2.791198;
  // Setting initial conditions to values
      at equilibrium point
  parameter Real d0=0.02, d_der0=0,
      i0=0.273841;
  // Variables MagLev
  Real d(start=d0, fixed=true),
      d_der(start=d_der0, fixed=true),
      i(start=i0, fixed=true), v, f, e;
  constant Real g=9.81;
  // Parameters PD
  parameter Real Kp=15, Td=0.05;
  parameter Real du_set=0 "Desired
      setpoint OP delta voltage of PD
      controller";
  // Variables PD
  Real u,y;
equation
  u = du_set - (e - e_e) "Input to the PD
      controller (negative feedback loop)";
  y = Kp*(u + Td*der(u)) "Ideal PD
      controller";
  v = y + v_e "Controller output to the
      plant";
  // Nonlinear MagLev plant equations
  f = k*i/d^4 "(1) force applied by the
      electromagnet on the levitating
      magnet";
  e = alpha + beta*1/d^2 + gamma*i "(2)
      voltage across the Hall effect
      sensor";
  der(d) = d_der;
  m*der(d_der) = m*g - f "(3) Newton's
      second law that";
  v = R*i + L*der(i) "(4) Kirchhoff's
      voltage law";
end MagLevNLPD;
```

Towards a High-Performance Modelica Compiler

Giovanni Agosta[1] Emanuele Baldino[1] Francesco Casella[1] Stefano Cherubin[1] Alberto Leva[1] Federico Terraneo[1]

[1]DEIB, Politecnico di Milano, Italy, `given_name.family_name@polimi.it`

Abstract

The use of Modelica as a modelling and simulation language is progressively replacing hand-tuned simulation code written in traditional imperative programming languages. This adoption is fuelled by the availability of libraries to target different application domains, as well as optimizations in Modelica implementations that allow to address larger problems. However, the effort required to extend existing Modelica implementations to support large scale models may not be economically sustainable by the Modelica community. To overcome this barrier, we believe a perspective change is required. Instead of developing, maintaining and optimizing a dedicated codebase, we propose to develop a Modelica implementation by relying on the LLVM state-of-the-art compiler framework. Although this approach will require a higher initial development effort, we believe that it will lead to significantly improved performance as well as lower overall cost. The paper discusses a possible roadmap for such a development, and presents a very early prototype implementation that exploits array structures by avoiding scalar expansion during the code generation process.

Keywords: Modelica Tools, Large-scale model simulation, Compilers, LLVM

1 Introduction

The high-level, declarative nature of the Modelica language has secured it a widespread adoption across industry and academia alike, bringing DAE-based modeling to many fields where custom simulation codebases had to be developed and maintained.

The performance of mainstream Modelica tools when handling large models has recently improved, mainly thanks to the introduction of sparse solvers (see, e.g. (Braun et al., 2017)). However, for systems approaching or exceeding the one-million equation target the code generation time is unacceptably large, as well as the memory footprint of the generated simulation code, which also has an impact on simulation speed due to CPU cache misses. Efficient simulation of large scale systems, with hundred of thousands to millions of equations, can today only be done with an acceptable compilation and execution performance through hand-written and hand-tuned simulation code. Large-scale models are typical of – and increasingly common in – a variety of relevant application fields: *smart grids* (Vialle et al., 2017) where there is the need to simulate the stability of an electrical network, *detailed thermal simulations* (Leva et al., 2016) that require to partition physical objects in a large number of finite volumes, *coarse-scale fluid dynamics* models for simulation studies targeted to energy efficiency (Bonvini and Leva, 2011). Several more examples could be reported that we omit for brevity.

In crafting hand-written simulation codes optimized to scale to millions of equations, the human designer follows an integrated approach, by coordinating optimizations that are specific of the simulation domain (such as exploiting sparsity in the model, causalization and tearing) and optimizations specific of the computer architecture domain (such as loop optimizations, cache optimizations, vectorization and parallelization).

Although the need to extend existing Modelica implementations to support large models is recognized by the Modelica community (Frenkel et al., 2011; Casella, 2015), significant effort is still required to effectively support large-scale systems. Existing Modelica toolchains are mainly targeted at medium-sized models, and therefore perform heavy structural analysis optimization passes along the translation process. These operations scale poorly for large-scale system. Furthermore, the C-code generation phase does not take into account architectural optimizations, and simply generates unoptimized C code. This approach passes the burden of optimization to the C compiler, to the detriment to both the overall translation efficiency and runtime performance.

A major issue concerning the generation of C code (or any other imperative language, for that matter) is that it is structurally impossible to make the compiler aware of structural properties of the code that could allow further optimizations. Such properties are an obvious consequence of the structural properties of the Modelica code. They can only be preserved by skipping the generation of an intermediate imperative code and by using an intermediate representation instead. One such property, for example, is guaranteeing the absence of pointer aliasing. A C compiler could in principle infer some of such properties from the generated C code, but there is no guarantee that such inference is complete and a lot of time would be wasted recovering information that was already known in the beginning. Moreover, existing Modelica workflows lose additional information during the flattening phase, such as arrays and looping constructs, and as a consequence the generated C code does not exploit existing CPU architectures effectively.

Significant performance improvements of Modelica tools could thus be achieved if an integrated approach is adopted, where high-level information from the Modelica source, instead of being transferred to an imperative language compiler, is used directly to produce architecture-optimized machine code, effectively resulting in a Modelica-to-binary-code workflow.

Summarizing, in this paper we argue that to scale the Modelica language to large-scale problems, a change of perspective is required, where a Modelica *compiler* – not just a translator – can perform model-specific and architectural-specific optimizations in an integrated way. Our proposal aims at improving the code generation process without any impact on the Modelica syntax and semantics. Thus, being fully compatible with existing Modelica models and libraries. However, our vision involves the re-design of portions of the existing compilation-related language specifications – e.g. the flattening.

We argue that to achieve this result in a cost-effective way, and without redesigning from scratch a complex code generation infrastructure and porting it to existing and future CPU architectures, said Modelica compiler has to be integrated in an existing compiler framework. For this reason, we propose to design a Modelica compiler integrated in LLVM (Lattner and Adve, 2004), which is a state of the art compiler framework, designed with the explicit goals of modularity and extensibility. The authors form an inter-disciplinary research group within the Dipartimento di Elettronica, Informazione e Bioingegneria of Politecnico di Milano, which includes strong competences in the areas of Modelica and object-oriented modelling and simulation, Computer Architectures, and Compiler Design.

This on-going work is today at a very early stage of development. The main goal of this paper is thus to present this group's vision and roadmap, as well as to present some initial results of a very early prototype.

This paper is organized as follows. Section 2 summarizes the state of the art of the support of large-scale models in Modelica tools. Section 3 shows a motivating example for the proposal, while Section 4 presents in detail our roadmap toward the development of a highly optimized Modelica compiler for large-scale systems. Section 5 illustrates the activities we carried out so far, and finally Section 6 ends the paper with some concluding remarks.

2 State of The Art

We now briefly describes the state of the art in order to motivate the presented research. Section 2.1 looks at the matter from the Modelica side, evidencing in particular some emerging application domains that require a technological evolution on the part of Modelica tools. Section 2.2 conversely takes the compiler technology standpoint, in a view to sketching out how recent developments in that domain could help realize the mentioned evolution.

2.1 The Modelica Side

As discussed in (Casella, 2015), the architecture of current mainstream Modelica tools was designed with individual systems in mind: one robot (possibly two cooperating robots), one hybrid car, one power plant, one air conditioning system, etc., which could be handled by expanding the system model all the way down to its scalar equations, performing optimization on them, and eventually generating code to solve them with ODE solvers using dense matrix algebra. In fact, the very same Modelica Language specification (The Modelica Association, 2017) describes the flattening process with reference to individual scalar variables.

Unfortunately, this approach does not scale well when large-scale systems and systems of systems are modelled. The potential application domains include power generation and transmission systems, smart grids, smart district heating systems with heat pumps (possibly integrated with smart grids), simulation of large fleets of interacting autonomous cars, building energy management simulation (BEMS), and all kinds of future internet-of-things and cyber-physical systems, whose behaviour is the results of the interaction of a large number of physical entities, interacting through a communication network and controlled by centralized and distributed control systems.

The availability of high-quality, open-source, general-purpose sparse solvers such as IDA, Kinsol, and KLU has recently triggered an effort to include support of sparse solvers in Modelica tools, as well as alternative approaches to the simulation of Modelica models that do not rely on the causalization of the system equations but use direct DAE solvers once the system has been symbolically brought to index 1, see (Braun et al., 2017). However, the structural analysis of the system equations, and the consequent code generation, is still carried out on a fully flattened and expanded system.

Some work has been carried out in the past on methods to carry out the structural analysis of the system while keeping repetitive structures such as arrays of variables and loop equations as atomic entities, see (Arzt et al., 2014), possibly also considering issues such as CPU cache misses in the generated code, see (Schuchart et al., 2015). (Zimmer, 2009) proposed methods to exploit the object-oriented structure of large system models, rather than going through full flattening of the equations, in order to come up with more efficient code generation strategies. Unfortunately, all these attempts have remained confined to the stage of concept or prototype implementation, but never made it into mainstream Modelica compiler technology.

2.2 The Compiler Technology Side

Compiler technology, while being from several points of view a mature research field, is still evolving. Modern compilers are very costly to develop, ranging in the tens to hundreds of person-years to reach full maturity when starting from scratch [1]. As a result, the ability to translate to and from multiple source and target languages is a highly desirable feature, as it allows to pool resources in the development of the large portion of a compiler that is neither target-dependent nor source-dependent.

Since many compiler transformations are fairly general (e.g., loop transformations (Bacon et al., 1994; Grosser et al., 2011) such a *loop unrolling* or *loop tiling* apply in the same way to all loops with the same induction variable evolution), re-implementing them for a new language is unlikely to provide any beneficial effect, and is instead likely to cost additional time in development, optimization, and bug fixing. Actually, the benefits of pooling development resources in this manner are so massive that it is preferrable to abstract some target and language properties (e.g., the size of C integer types for a given target machine) into codified data structures in order to maximise the fraction of the compilation that can be handled by otherwise target-independent, source-independent tools. Thus, a modern compiler is usually implemented on top of a *compiler framework*, a collection of libraries for manipulating and storing an *intermediate representation*, that is a set of data structures that are semantically equivalent to the original program.

As a result of this trend, the GNU Compiler Collection (GCC) dominated the compiler market for decades. However, advanced software does not always age well, and adding more and more optimisation passes forced GCC to stretch the limits of its original design, for instance by adding multiple intermediate representations to supplement the original RTL, which was deemed too low-level to allow certain optimisations.

Nowadays, the industry is increasingly supporting the LLVM compiler framework (Lattner and Adve, 2004) as a more streamlined and modern alternative, leveraging a single, low-level intermediate representation, but capitalising an improved ability to perform loop transformations on lower lever representations. Thanks to the support from multiple large companies such as Google, Apple, Arm, and Sony, LLVM was able to catch up a 20+ years development gap, reaching a position of industry standard in a mere decade from its introduction in 2004. As anticipated and better detailed in Section 4 later on, we propose to develop a Modelica compiler based on LLVM. This choice will in our opinion entail advantages as for both simulation code efficiency and compiler maintainability.

Regarding efficiency, some optimizations were already mentioned, namely operating under the guarantee of no pointer aliasing. Others are for example loop optimizations, on which some words are spent later on. In addition, not going through an imperative language allows the compiler to preserve the non-ordered character of the model (equations in Modelica) as opposite to the ordered nature e.g. of C vectors. When vectors are created to host variables in current mainstream Modelica tools, the order in which these occupy a vector is chosen without any conscience of the consequences on the final machine code. For example, on two different architectures, the same vector order can result in very different cache management efficiencies. A C compiler is inherently incapable of swapping two elements in a vector, as this would alter the semantics of the C program. The same operation however does not alter the semantics of the model, for which the order of variables in vectors is irrelevant.

Coming to the creation and maintenance of the envisaged compiler, a distinctive feature of this research is that LLVM-based compiler development mostly concerns imperative languages, while Modelica is *declarative*. Despite the problems that will surely be encountered, adopting the LLVM framework is keen to produce benefits also from this viewpoint. When considering a highly specialized declarative language such as Modelica, one may come to the wrong conclusion that the majority of transformations required by the code generation process will be strictly language-dependent, and thus to be developed from scratch. In fact, this is actually not the case, as most of the primitives that are provided in an optimized way by the LLVM framework can readily be used in the Modelica context. For example, the well-known equation-variable matching phase of the code generation process starting from Modelica models corresponds to a graph manipulation problem for which LLVM provides the basic data structures (nodes, arcs). In fact, it turns outs that the standard matching algorithm is already implemented efficiently in LLVM [2], because it represents the basic foundation of other types of data-flow analysis, so it can be readily re-used.

3 Motivating Example

As a motivating example for our research, in this section we show and briefly comment an experiment that was performed to understand the current scalability gap between Modelica toolchains and optimized handwritten code.

Consider the Modelica benchmark code in Listing 1. The code represents a simple 1D thermal model, describing thermal conduction in a solid copper wire divided in *nx* sections of equal length. Just as in the ScalableTestSuite (Casella, 2015), this model can be simulated with a progressively large *nx*, to observe the scalability of a Modelica toolchain.

The model was tested with a number of equations ranging from 10 to 1 million, using OpenModelica 1.13.0 and Dymola 2018. In both toolchains, an explicit Euler integration algorithm was used. The platform used to run the experiments is a NUMA node with two Intel Xeon E5-2630 V3 CPUs (@3.2 GHz), and 128 GB of DDR4 mem-

[1]See `https://news.ycombinator.com/item?id=16469218` for the development cost of GCC and related tools by Cygnus, estimated in 250 M$ over 10 years by founder D. Henkel-Wallace, and `http://www.ace.nl/compiler/cosy.html` for the development cost of CoSy, estimated in 200 person/years.

[2]See `http://llvm.org/doxygen/SCCIterator_8h_source.html`

Listing 1. Thermal conduction benchmark.

```
model Thermal1D
parameter Integer nx = 1000;
parameter Real area = 0.0005^2*3.14;//m^2
parameter Real nlength = 0.1;       //m
parameter Real conductivity = 401;  //W/m.K
parameter Real specheatcap = 385;   //J/Kg.K
parameter Real density = 8960;      //Kg/m^3
parameter Real Thi = 400+273.15;    //K
parameter Real Tlo = 20+273.15;     //K
parameter Real g =
  conductivity * area / nlength;
parameter Real c =
  specheatcap * density * area * nlength;
Real T[nx];
initial equation
  for x in 1 : nx loop
    T[x] = Tlo;
  end for;
equation
  c * der(T[1])  =   g * (T[2]    - T[1])
               + 2*g * (Thi     - T[1]);
  c * der(T[nx]) =   g * (T[nx-1] - T[nx])
               + 2*g * (Tlo     - T[nx]);
  for x in 2 : nx-1 loop
    c * der(T[x]) = g * (T[x-1] - T[x])
                + g * (T[x+1] - T[x]);
  end for;
annotation(experiment(StartTime = 0,
StopTime = 100000, Tolerance = 1e-6,
   Interval = 20));
end Thermal1D;
```

Listing 2. Optimized C++ implementation.

```
#include <cstdio>
#include <cstring>
#include <string>
#include <chrono>
#include <algorithm>

using namespace std::chrono;

// #define PRINT

int main(int argc, char *argv[])
{
    if(argc<2) return 1;
    int N = std::stoi(argv[1]);
    const int Nsteps = 5000;
    const double h = 20.0;
    const double g = 0.00314785;
    const double c = 0.2707936;
    const double Thi = 400.0 + 273.15;
    const double Tlo = 20.0 + 273.15;
    double *x=new double[N];
    double *xo=new double[N];
    for(int i = 0; i < N; i++) xo[i] = Tlo;
    FILE *fh=fopen("log.csv","w");

    auto a = steady_clock::now();
    for(int j = 0; j < Nsteps; j++)
    {
        x[0] = (1.0-3.0*g*h/c) * xo[0]
             + g*h/c * xo[1]
             + 2.0*g*h/c*Thi;
        for(int i = 1; i < N-1; i++)
            x[i] = g*h/c * xo[i-1]
                 + (1.0-2.0*g*h/c) * xo[i]
                 + g*h/c * xo[i+1];
        x[N-1] = (1.0-3.0*g*h/c) * xo[N-1]
               + g*h/c * xo[N-2]
               + 2.0*g*h/c*Tlo;
        std::swap(x,xo);
        #ifdef PRINT
        for(int i = 0; i < N; i++)
            fprintf(fh,"%e,",xo[i]);
        fprintf(fh,"\n");
        #endif //PRINT
    }
    auto b = steady_clock::now();
    auto e = duration_cast<
             duration<double>>(b-a)
             .count();
    printf("Simulation time %f\n",e);
    fclose(fh);
    delete[] x;
    delete[] xo;
}
```

ory (@1866 MHz) on a dual channel memory configuration. The operating system is Ubuntu 16.04 with version 4.4.0 of the Linux kernel. The compiler used is CLANG version 3.8.0 for OpenModelica, and GCC 7.3.0 for Dymola. The model was also manually translated in optimized C++ code, using an explicit Euler integration algorithm, exploiting the sparsity in the model, and preserving the contained loop constructs. The optimized version can be found in Listing 2.

Tables 1-3 show the results. The simulation column reports only the time for the integration of the differential equations, excluding initialization time and the time required to save results to disk. The binary code size column is only the part of the executable file containing assembly instructions (the .text section), in order to not take into account other metadata such as debug symbols that could be present in the executable. The source code size column is the sum of the size of all C and header files produced by the translator.

From the tables, two main facts can be noted. First, the current generation of Modelica translators is, at least in this simple example, around two orders of magnitude slower than hand-tuned code. Second, the tables evidence the effects of the loss of model structure in current Modelica translators. The flattening of looping constructs results in a source and binary code size that grows linearly with respect to the number of equations in the system, while this does not happen in the handwritten code. As a smaller code size translates to better cache locality, this difference can at least partially explain the improved simulation performance of hand tuned code.

However, there is no theoretical reason why an optimizing Modelica compiler could not generate as efficient

Table 1. OpenModelica Thermal1D simulation time, binary and source code size

Equations	Simulation	Binary	Source
10	40 ms	34.6 KByte	46.1 KByte
100	43 ms	101 KByte	183 KByte
1000	331 ms	775 KByte	1.55 MByte
10000	2.49 s	7.37 MByte	15.6 MByte
100000	69.0 s	73.7 MByte	160 MByte
1000000	Stopped after 2 hours		

Table 2. Dymola Thermal1D simulation time, binary and source code size

Equations	Simulation	Binary	Source
10	37.6 ms	193 KByte	8665 Byte
100	49.1 ms	238 KByte	53.7 KByte
1000	274 ms	690 KByte	527 KByte
10000	2.64 s	5.17 MByte	5.35 MByte
100000	61.9 s	50.9 MByte	55.7 MByte
1000000	Stopped after 2 hours		

Table 3. Handwritten C++ Thermal1D simulation time, binary and source code size

Equations	Simulation	Binary	Source
10	46 μs	4922 Byte	1258 Byte
100	257 μs	4922 Byte	1258 Byte
1000	3.72 ms	4922 Byte	1258 Byte
10000	34.2 ms	4922 Byte	1258 Byte
100000	401 ms	4922 Byte	1258 Byte
1000000	3.62 s	4922 Byte	1258 Byte

code as the handwritten one—a remark that in our opinion motivates our research path.

Finally, it is also worth noticing that the time required by the Modelica translators to produce the C code and compile it can significantly exceed the simulation time. Considering the 100000 equations benchmark, OpenModelica took 55 minutes, while Dymola took 4 minutes and 20 seconds. Furthermore, the 1 million equations benchmark was stopped for both Dymola and OpenModelica after two hours, and the simulation had not yet started. Compiling the handwritten C++ code took only 183ms, as the code size is independent on the model size, although a fair comparison cannot be made due to the time required to manually translate the Modelica code to C++.

4 Roadmap

Given the considerations laid out so far, it is the authors' opinion that producing highly optimized binary code from a Modelica model is possible. The process we envision first translates the Modelica code into an LLVM intermediate representation (LLVM-IR), and then turns that directly into architecture-optimized machine code. Such an approach exploits all the structural information and metadata that comes from the original Modelica model to the fullest extent.

We also believe that, given the functionality offered by the LLVM framework, this objective can be achieved with an effort that will be abundantly rewarded in terms of efficiency, scalability and, last but not least, maintainability. The performance of the LLVM framework will further improve over time thanks to the efforts of the very active community working on it, which is much wider than the community of Modelica tool developers.

The roadmap laid out here is based on three main assumptions:

- in the future there will be a growing interest in the simulation of large-scale, modular Modelica models of ever-increasing size;
- such large-scale models are built by connecting a very large number of instances of a relatively small number of models, which only differ by the numerical values of their parameters – this is possibly (but not necessarily!) done via arrays of variables and models;
- virtually all modular models are characterized by local interaction, i.e., most (if not all) the components in the system interact with a small number of neighbours only, which means that the corresponding DAE system has a very high degree of sparsity and O(N) non-zero entries in the incidence matrix, N being the number of instantiated models.

The Modelica compiler we are aiming to build will exploit these features to achieve highly efficient and optimized simulation code generation and execution. This will be obtained by working on two lines of development, Line A and Line B, which are orthogonal and can be carried out simultaneously. Line B is partitioned into two subsequent phases, as shown in Figure 1.

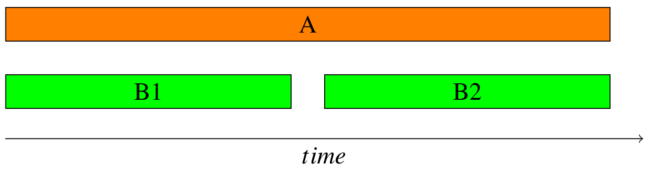


Figure 1. The development of our proposed compiler will follow two lines, Line A (compiler backend) and Line B (compiler frontend), partitioned into Phase B1 and Phase B2.

4.1 Line A

Line A focuses on the improvement of the Modelica compiler backend by directly integrating with the LLVM compiler framework. The Modelica code will be translated directly into an LLVM-IR, which retains all the structural information that can be extracted from the original Modelica code. This will allow the generation of machine code which is optimized thanks to this information, as well as all the available information about the target hardware architecture.

The traditional intermediate C (or C++) code generation will thus be skipped, drastically reducing the code generation time while at the same time allowing more optimizations to be performed faster.

4.2 Line B

Line B focuses on the Modelica compiler frontend, which extends from the Modelica source code parsing to the the transformation of the DAE equations in a form that can be passed to a DAE solver.

4.2.1 Phase B1

The traditional Modelica code generation toolchains are based on the complete flattening of the object-oriented features and on the expansion of arrays and unrolling of `for` loops. In fact, the very same Modelica Language Specification is written with this assumption in mind.

The goal of this line is to preserve arrays, `for` loops, and in general the object-oriented structure of the models as much as possible, in order to factor out common behaviour (= equations) in large-scale models. Thus it is possible to achieve much faster code generation, a much smaller memory fooprint, and hence much faster code execution thanks to the vastly reduced chances of cache misses, among other optimizations sought after in Line A. Of course the generated code should eventually lead to the solution of a system of equations which is equivalent to the one that would be obtained by applying the full flattening and expansion mentioned in the Modelica Specification.

The main idea is that the machine-code function to compute the residuals of DAE (and their directional derivatives) in an object which is instantiated many times in the system should only be generated once and then called many times using different inputs and outputs corresponding to the specific variables of each instance.

The concept should be then extended to cover hybrid systems, involving the equations in when clauses, the clocked equations and the zero-crossing functions which are also repeated many times in the large-scale model. The very nice feature of this approach is that the code generation time and the generated machine code footprint scales as $O(1)$ for a large system with N components.

This approach requires the use of direct sparse DAE solvers, such as IDA, which avoids the need of causalization and allows to preserve an N:1 mapping (possibly with some optimizations such as alias elimination) between each equation in `for` loops or model arrays and the corresponding function computing the residual of the equation in the DAE system. This would not possible if the system were causalized, turning it into a set of ODEs, because in general the causalization destroys such N:1 correspondence, depending on the specific causality relationships in the overall system model.

In fact, such an N:1 mapping can be applied to the vast majority of the DAE equations a typical large-scale system model. However, a small set of equations remains that needs a special handling, that will be carried out following the traditional approach for simplicity.

The first sub-set in this set of equations is given by the equations corresponding to *flow* variables in connection sets. Assuming there are no redundant connection statements in a connection set, a statement such as `connect(a,b)` can be directly mapped into the equations $0 = a.v_{nf} - b.v_{nf}$ with an N:1 mapping only for the non-flow variables v_{nf}. The equations for flow variables, instead, can only be generated once the connection sets have been computed, a task that can only be performed by analyzing the fully assembled system; only one flow equation per connection set is eventually generated.

The second sub-set is given by the auxiliary equations needed to generate the results of `inStream()` operators. Also in this case it is necessary to analyze the connection sets of the full system model, since the expression of the results of the `inStream()` operator depends on the cardinality of the set and on the `min` attribute of the flow variables of each involved connector.

The third sub-set involves DAE systems of index greater than one. If the system is known a-priori to have index one, as it is e.g. the case of phasor-based power generation and transmission system models, then this set is empty and there is no need of further processing. The a-priori assumption of structural index one could be declared by a suitable annotation of the model. Otherwise it is necessary to flatten and expand the system model all the way down to scalar components, run the matching algorithm and in case of failure due to structural high index, run Pantelides' algorithm, which will identify algebraic constraint equations between variables that appear differentiated in the model, differentiate them and add them to the original set of DAEs. The dummy-derivatives algorithm will also require to select the state set (statically or dynamically), and to demote some derivatives to dummy derivatives, hence a provision must be made to identify as dummy derivatives some elements of un-expanded arrays of derivatives, that are handled by the $O(1)$ efficient code described above. Eventually, the DAE solver will be passed the residuals and Jacobian of a reduced-order index-1 system.

Note that in Phase B1 we still need to expand all non index-1 systems, as in the current state-of-the-art Modelica compilers. We postpone the exploitation of optimization opportunities for higher-index system to the B2 phase. This milestone partitioning allows us to reach a working compiler in shorter time by prioritizing the optimization of index-1 systems.

Summing up, a straightforward implementation of the tool requires to fully flatten and expand the model to scalar components, build the connection sets on it, and then run the standard structural analysis algorithms on it. Note that this fully expanded model *will not be used directly for code generation*, but only to perform structural analysis. The actual code generation process will start from an unexpanded version of the model, in order to achieve $O(1)$ performance as much as possible.

This processing phase on the fully expanded model requires $O(N)$ time, so it doesn't scale as well as the generation of equations that have an N:1 mapping, which scales as $O(1)$ as discussed above. However, experience carried out by some of the authors with the OpenModelica compiler shows clearly that the time and memory resources involved in these specific phases of the processing of a fully flattened model are a tiny fraction (5–10% at most) of the total. Hence, a performance improvement of at least one order of magnitude is expected by following this approach, compared to the traditional approach of running all the code generation phases on the fully flattened system.

As to the runtime performance of the executable simulation code, it has to be noted that the cardinality of the additional set of equations that need to be generated from the fully expanded model (connection equations for flow variables, equations defining `inStream()` outputs and equations differentiated by Pantelides' algorithm) is again a very tiny fraction (a few percentage point at most) of the total number of equations of the system. This means that the penalty on the runtime performance and memory footprint of these equations not being handled in an array- and object-oriented-structure-preserving way will be very small, compared to what happens when a traditional full flattening and expansion approach is followed.

4.2.2 Phase B2

Once the development of Phase B1 is complete, it would be possible to focus on the modularization of the structural analysis algorithms. The current state-of-the-art approaches work only on scalar variables. In this phase, a generalized versions of such algorithms will be designed, which can handle entire arrays and sets of equations from `for` loops, or from arrays of models as individual `E` and `V` nodes.

On one hand, this modularization would further improve the performance and the scalability of the tool. On the other hand, developing such generalized algorithms that work efficiently in all cases could turn out to be quite a hard task. Therefore, the ratio between the development effort and the performance gains is probably going to be much less spectacular than the one that can be achieved by completing Phase B1. It is the authors' opinion that this kind of optimization is worth considering only after the optimizations described in Phase B1 have been fully exploited.

5 On-Going Work

Since the second half of 2018, we started the implementation of a compiler prototype to materialize the effort discussed in Section 4. The development of such prototype is being addressed in a master's thesis work that aims at demonstrating the benefits of the efficient exploitation of arrays and equation loops. Due to time and resource limitations, the current prototype is still in a very preliminary state.

The focus of this thesis work is avoiding the generation of redundant code that is obtained when the conventional flattening-based approach based is followed. At the moment authors are writing, our prototype is limited to the handling of flat models with no object-oriented structures. Structural analysis is still not implemented, so that only the dense version of the IDA DAE solver can be used. Last, but not least, the direct LLVM-IR code generation is not yet in place, and the prototype still generates C code that is then compiled by clang into executable code.

As a consequence, the performance of our compiler prototype on large-scale models is still very far from the objectives stated in the roadmap. That said, our prototype can handle simple Modelica models with arrays and for loops, producing correct simulation results. The improvements currently supported by our prototype lie in the code generation stage. We aim at the preservation of the data- and code-structures concepts as they are written in the Modelica source code. In particular, we avoid to perform the *vector expansion* whenever it is possible. Thus, we generate a residual function that features loops over variables. Our compiler generates a compact code that better exploits instruction locality with respect to the code generated by OpenModelica.

Another improvement over the OpenModelica code generation consists in the reduced modularity of the residual function. OpenModelica generates a single C function for each equation to be described. This approach is fairly convenient for debugging purposes, as it allows to trace the effects of the single equation from the source code to the executable binary. However, it implies a non-trivial overhead due to function call instructions at runtime, and this overhead is not paid back through code reuse, as these functions are only called once. We reduce this overhead during the code generation stage in two ways. The first optimization is a direct consequence of the loop preservation: whenever there is an equation within a loop body, we reuse the same code at each iteration of the loop. The second optimization consists in the inlining of functions, to be performed before the code generation. Instead of generating an independent C function and respective call instructions to invoke it, we directly place the code of that function in the residual function. This second optimization can be performed at almost zero cost during the code generation stage, as opposed to later forcing the compiler to analyze the emitted code as a whole.

Although preliminary results on small dense systems supported by our prototype are promising, actual direct performance comparisons on meaningful test cases against mainstream Modelica translators will become significant as soon as the support for the sparse version of the IDA solver is implemented. This effort and the replacement of the intermediate C code generation pass with the LLVM-IR one are planned to be carried out in 2019.

6 Conclusions

In this paper, we are proposing to realise a Modelica compiler based on a compiler framework, namely LLVM. The presented research is carried out by a group at the Politecnico di Milano, putting together knowledge and experience about Modelica and its use for a wide variety of applications, about computer architectures, and about compiler science and technology.

We have motivated our proposal based on current trends observed in the problems that Modelica models need to address, with particular (yet not exclusive) reference to large-scale systems.

We have argued that not passing through the generation of source code in an imperative language can yield improvements in terms of wider optimization possibilities, as the semantics of a language like C inherently causes a loss of information about the semantics of the original model, that could be exploited to tailor the code to its target architecture.

We have also noticed that the huge effort spent, and the vast community involved in compiler frameworks, quite certainly entail future benefits in terms of compiler standardization and maintainability.

We have shown a motivating example to support our statements, defined a roadmap for future activities, and briefly described what we carried out so far.

We hope that this paper fosters a discussion in the Modelica community, and that our proposal can be a basis for a future generation of efficient, architecturally flexible and easily maintainable Modelica compilers.

References

Matthias Arzt, Volker Waurich, and Jörg Wensch. Towards utilizing repeating structures for constant time compilation of large Modelica models. In David Broman and Peter Pepper, editors, *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 35–38, Berlin, Germany, Oct 10 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666207.

David F. Bacon, Susan L. Graham, and Oliver J. Sharp. Compiler transformations for high-performance computing. *ACM Comput. Surv.*, 26(4):345–420, December 1994. ISSN 0360-0300. doi:10.1145/197405.197406. URL `http://doi.acm.org/10.1145/197405.197406`.

M. Bonvini and A. Leva. Object-oriented sub-zonal modelling for efficient energy-related building simulation. *Mathematical and Computer Modelling of Dynamical Systems*, 17(6): 543–559, 2011. doi:10.1080/13873954.2011.592143.

W. Braun, F. Casella, and B. Bachmann. Solving large-scale Modelica models: new approaches and experimental results using OpenModelica. In *Proc. 12th International Modelica Conference*, pages 557–563, Prague, Czech Republic, 2017. doi:10.3384/ecp17132557.

F. Casella. Simulation of large-scale models in Modelica: State of the art and future perspectives. In *Proc. 11th International Modelica Conference*, pages 459–468, Versailles, France, 2015. doi:10.3384/ecp15118459.

J. Frenkel, C. Schubert, G. Kunze, P. Fritzson, M. Sjölund, and A. Pop. Towards a benchmark suite for Modelica compilers: Large models. In *Proc. 8th International Modelica Conference*, pages 143–152, Dresden, Germany, 2011. doi:10.3384/ecp11063143.

Tobias Grosser, Hongbin Zheng, Raghesh Aloor, Andreas Simbürger, Armin Größlinger, and Louis-Noël Pouchet. Polly-polyhedral optimization in llvm. In *Proceedings of the First International Workshop on Polyhedral Compilation Techniques (IMPACT)*, volume 2011, page 1, 2011.

C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. 2004 International Symposium on Code Generation and Optimization*, pages 75–86, Palo Alto, CA, USA, 2004. doi:10.1109/cgo.2004.1281665.

A. Leva, F. Terraneo, and W. Fornaciari. Event-based control as an enabler for high power density processors. In *Proc. 2nd International Conference on Event-based Control, Communication, and Signal Processing*, pages 1–8, Krakow, Poland, 2016. doi:10.1109/EBCCSP.2016.7605253.

Joseph Schuchart, Volker Waurich, Martin Flehmig, Marcus Walther, Wolfgang E. Nagel, and Ines Gubsch. Exploiting repeated structures and vectorization in modelica. In *Proc. 11th International Modelica Conference*, pages 265–272, Versailles, France, Sep 21–23 2015. doi:10.3384/ecp15118265.

The Modelica Association. Modelica - A unified object-oriented language for physical systems modeling - Language specification version 3.4. Online, 4 2017. URL `https://www.modelica.org/documents/ModelicaSpec34.pdf`.

S. Vialle, J.P. Tavella, D. Cherifa, R. Corniglion, M. Caujolle, and V. Reinbold. Scaling FMI-CS based multi-simulation beyond thousand FMUs on Infiniband cluster. In *Proc. 12th International Modelica Conference*, pages 673–682, Prague, Czech Republic, 2017. doi:10.13140/RG.2.2.14481.63847.

D. Zimmer. Module-preserving compilation of Modelica models. In *Proc. 7th International Modelica Conference*, pages 880–889, Como, Italy, 2009. doi:10.3384/ecp09430028.

SESSION 3C: MECHANICS & TRANSPORT

Overview of the DLR RailwayDynamics Library

Andreas Heckmann[1] Marc Ehret[1] Gustav Grether[1] Alexander Keck[1] Daniel Lüdicke[1]
Christoph Schwarz[1]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR), Germany,
andreas.heckmann@dlr.de

Abstract

The newly released commercial DLR RailwayDynamics Library is intended to support the design, optimization and control development as well as hardware- and software-in-the-loop testing of railway vehicles mainly on the system level. To this aim, it provides the capability to consider vehicle dynamics issues such as traction, comfort and safety in multi-domain engineering tasks by preparation of vehicle, track, wheel-rail contact models and roller rig scenarios on different levels of detail.

Exploiting several precursor papers on specific railway modeling topics, their models have been collected and reorganized in order to propose a sound modeling framework dedicated to railway dynamics.

The paper gives an overview on particular concepts and ideas of the library, presents several application examples and discusses two approaches to organize multi-domain modeling.

Keywords: railway vehicle dynamics, wheel-rail contact, multi-domain vehicle modeling

1 Introduction

1.1 Background

A high level of safety and comfort as well as sustainability and protection of natural resources are high-level objectives of the DLR-internal, long-term research project Next Generation Train (NGT) and compiled to vehicle dynamics, suspension design, running gear development and active guidance control as associated tasks of the DLR Institute of System Dynamics and Control (SR).

Despite the significance of hardware testing, modeling, simulation and optimization remain the dominating tools in research efforts on safety enhancements, function upgrades, comfort improvement and reduction of wear, energy consumption and life-cycle costs. These tools offer the opportunity to examine and evolve new technical concepts in early design phases without implementation risks and by comparable low costs. Moreover, the use of Modelica is in particular attractive since it provides the capability to cover multi-domain engineering tasks in one consistent simulation environment, cf. (Carrarini et al., 2010).

Hence, it is not surprising that there already exists a slew of Modelica publications that report on NGT and cooperation project results in railway engineering such as on energy flows in electric railway networks (Heckmann and Streit, 2012), on wheel-rail contact (Heckmann et al., 2014a), running gear (Schwarz et al., 2015) and pneumatic brake system modeling (Ehret, 2018) and on crosswind stability assessment (Heckmann and Grether, 2017).

1.2 Objectives

The idea of the present paper is to gather those models and experiences from the work quoted above, organize and propose a modeling framework dedicated to railway dynamics and running gear design. This includes the consideration of the nonlinear wheel-rail contact in normal and in tangential direction in view of the wheel and rail profile geometry.

Besides analyzing classical vehicle dynamics topics such as traction, comfort and safety, the capability to work on multi-domain engineering tasks is a specific focus of the RailwayDynamics Library. In fact, railway vehicles also employ multiphysical subsystems such as pneumatic friction brakes and air suspensions, electrical engines to provide propulsion and to regenerate energy, Diesel-electric or Diesel-hydraulic drive trains and so on.

With this background, the commercial DLR RailwayDynamics Library is supposed to support holistic system design, optimization and hardware or software in-the-loop testing by provision of vehicle dynamics models that may be scaled and adapted with respect to the required modeling level.

An overview on the library structure is given in the following section. Some particularities of railway modeling are presented in Section 3, while Section 4 contains elaborate example applications. Section 5 is a discussion on different approaches for multi-domain modeling. Section 6 concludes the paper and gives an outlook.

2 Overview

2.1 Library Structure

Figure 1 presents the main subpackages of the library. Each major subpackage and its models are additionally marked by using different fundamental icon colors, namely light grey, light green, light red or light blue.

The *General* subpackage contains multi-purpose models for all kind of analysis in the context of railway dynamics and also covers the operation of test rigs. It includes lateral, longitudinal, vertical, roll, pitch and yaw dynam-

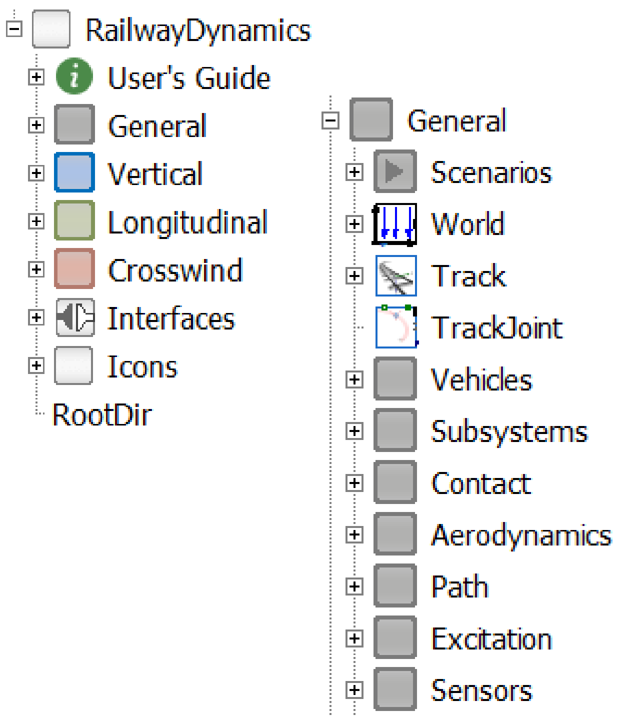


Figure 1. Structure of the RailwayDynamics Library

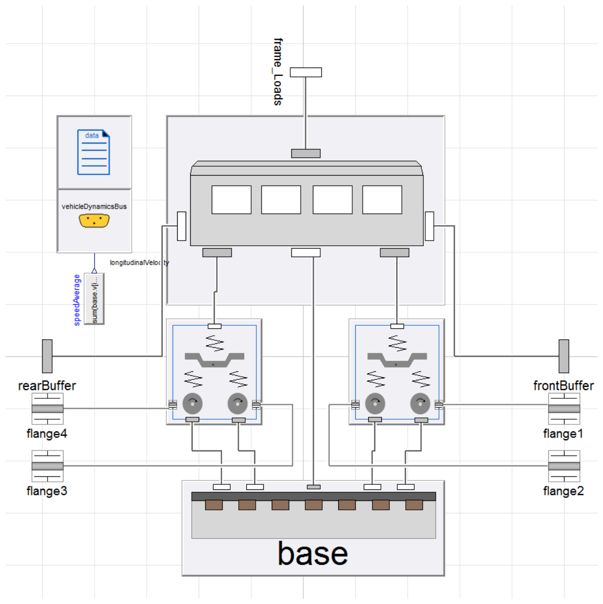


Figure 2. Structure of vehicle model

ics which may be investigated for comfort, traction, safety purposes among others.

The *Vertical* subpackage gathers specific models with vertical degrees of freedom, only, which could be used for preliminary surveys on vibration comfort and the associated lay-out of suspensions.

The *Longitudinal* subpackage is intended to be used to study traction and braking maneuvers of trains, which explicitly requires to consider the longitudinal motion of railway vehicles and the associated rotational motion of wheels or wheelsets, respectively.

The *Crosswind* subpackage is tailored for quasistatic crosswind stability analysis according to Sec. 5.4.3 in (EN 14067-6: 2010).

Generally speaking, models from the *Vertical*, *Longitudinal* or *Crosswind* subpackage are specializations and are supposed to replace models from the *General* subpackage in order to focus on more specific analysis goals and balance the computational resources according to the needs on hand.

2.2 Vehicle Substructuring

The example vehicle model dedicated to one single car in Figure 2 gathers submodels to represent

- the railroad base as an aggregation of *track joints* and *track panels* to be further explained in Section 3.2 and Section 3.3,
- two running gears, which include wheelsets, bogie frames, primary and secondary suspensions,
- and the carbody.

The vehicle model composition in Figure 2 serves as a template and is employed throughout the complete library. Each submodel may be replaced by another submodel that stems from the same partial base class.

Four *flange with bearing* connectors from the Modelica Standard Library (MSL) and its Multibody subpackage, respectively, allow for the application of traction torques from the outside to be transferred to the wheelsets and, as an option supported at the bogie frame.

Two multibody *frame* connectors called *rear* and *front buffer* enable the connection to leading of traveling cars.

The *vehicle dynamics bus* here transmits one signal which is the longitudinal velocity of the car.

Data on masses, fundamental geometry, primary and secondary suspensions are collected by one *data* record, to be further explained in the following section.

2.3 Data Concept

The collection of fundamental vehicle data in one record and their propagation to submodels is useful to retain control and a clear view on the parametrization of the model. However, almost each vehicle requires different data and needs a particular record structure since e.g. the options to design railway suspensions are numerous.

Therefore, it appears useful to declare the *Data* record locally as a specifically tailored encapsulated record as follows:

```
model Locomotive
  import RGV=
      RailwayDynamics.General.Vehicles;
  encapsulated record Data
    extends RGV.partialVehicleData;
    ...
  end Data;
  parameter Data data;
  ...
end Locomotive;
```

The submodels of the vehicle here called *Locomotive* are supposed to refer to the above declared record in the following way:

```
model LocomotiveBogie
  import RGS=
      RailwayDynamics.General.Subsystems;
  extends RGS.RunningGear.Bogie(
    redeclare Locomotive.Data data)
  ...
end LocomotiveBogie;
```

3 Railway Modeling Particularities

In order to introduce the particularites of the RailwayDynamics Library, Figure 3 shows a simple scenario, namely a single wheelset running along curved track.

3.1 Track

The track instance is mandatory for every model which uses the RailwayDynamics Library except the later on presented roller rig environment. It contains information on some global parameters using the Modelica *inner / outer* mechanism and defines

- the path as a function of the path length parameter s, i.e. the 3D curve $\boldsymbol{r} = \boldsymbol{r}(s)$, the vehicle is intended to move along, and the collateral frame, whose unit basis vectors are $\boldsymbol{t} = \boldsymbol{t}(s)$, $\boldsymbol{n} = \boldsymbol{n}(s)$ and $\boldsymbol{b} = \boldsymbol{b}(s)$,
- the rails, which are symmetrically aligned along the path and
- the irregulartities that specify local deviations or disturbances of the idealized path and rail definition.

The 3D curve of the path is described by supporting points, which are interpolated by B-Splines in a sufficiently smooth manner. Together with the superelevation or roll angle $\phi(s)$, the supporting points are read from a file, to which a string parameter of the track component refers. The orientation of the collateral frame then results from the following definitions:

$$\begin{aligned} \boldsymbol{t}(s) &= \frac{\boldsymbol{r}_{,s}}{|\boldsymbol{r}_{,s}|} \quad \text{with} \quad (\;)_{,s} := \frac{\partial(\;)}{\partial s}\,, \\ \boldsymbol{n}(s) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi(s) & \sin\phi(s) \\ 0 & -\sin\phi(s) & \cos\phi(s) \end{pmatrix} (\boldsymbol{a} \times \boldsymbol{t})\,, \\ \boldsymbol{b}(s) &= \boldsymbol{t} \times \boldsymbol{n}\,, \end{aligned} \tag{1}$$

where the auxiliary unit vector $\boldsymbol{a}$ is a user defined parameter. Its introduction is needed to overcome the shortcomings of the Frenet frame definition, which refers to $\boldsymbol{t}_{,s}$ in order to specify $\boldsymbol{n}$ and results in a zero vector for straight line paths, cf. (Weber, 1990).

In the animation in Figure 4 the path progress is delineated as a transparent red band. The red frame presents the local triad at the instantaneous position. Whenever the terms longitudinal, lateral and vertical are used in the context of this library, they refer to this local coordinate system that moves along the predefined track path. The longitudinal direction hereby is specified by $\boldsymbol{t}$, i.e. tangential to the instantaneous track position, and the vertical axis referring to $\boldsymbol{b}$ is pointing downwards as usual in railway dynamics.

The position and orientation of the two rail reference frames are given relative to the path by three parameters: *gauge*, *gaugeOffset* and *inclination*, for which usual values are 1/20 or 1/40.

The irregularities implemented so far are random realizations of disturbances with specified spectral properties. The user may select four different geometrical types of irregularities (vertical, lateral, crosslevel and gauge) from a number of predefined spectra that are taken from railway vehicle textbooks or papers, e.g. (Frederich, 1984) or (Haigermoser et al., 2015).

3.2 Track Joint

Recall the *prismatic* joint from the MSL-Multibody subpackage that specifies one mechanical degree of freedom

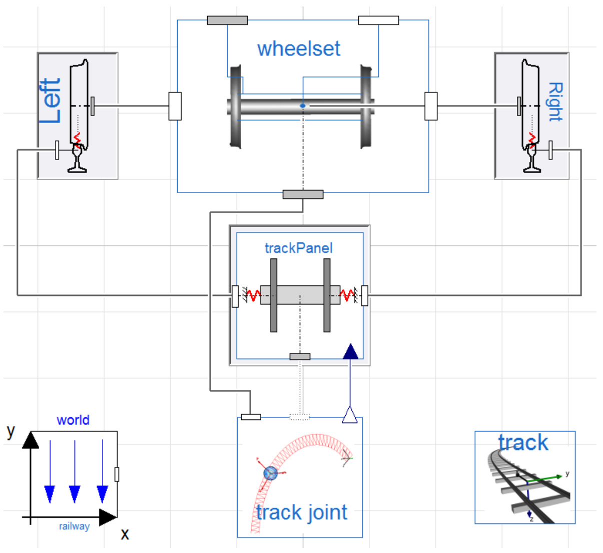


Figure 3. Diagram layer of a *wheelset on a curved track* model

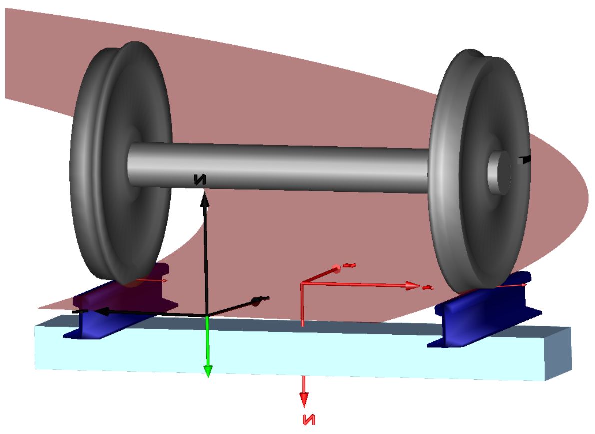
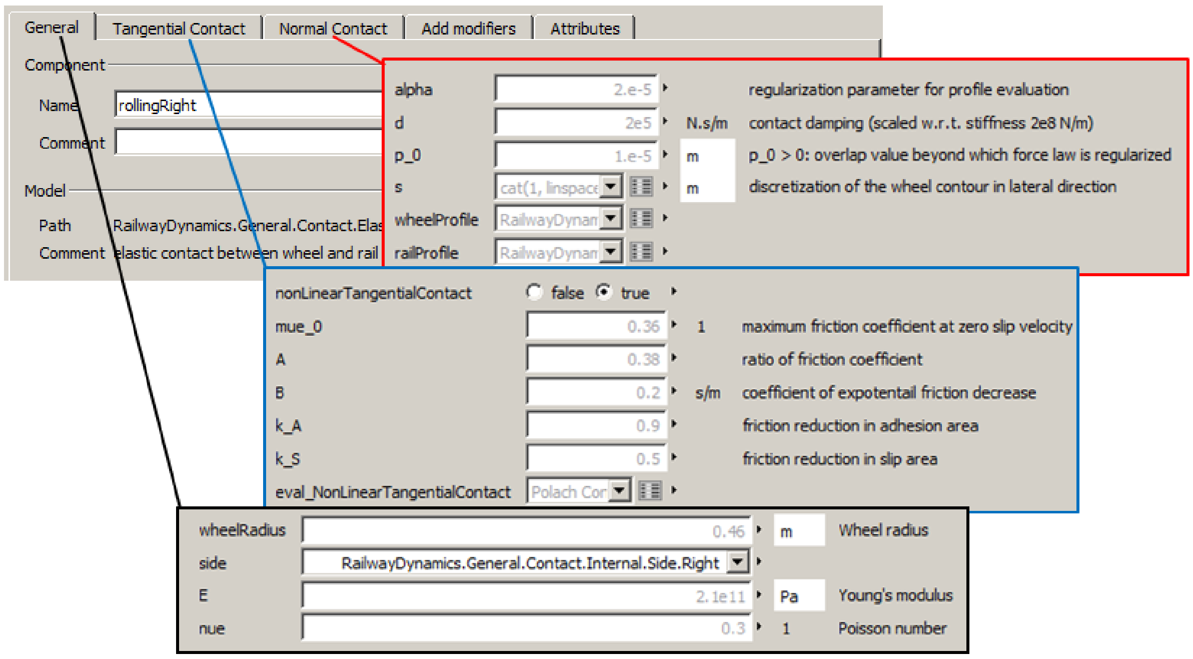


Figure 4. Animation of a *wheelset on a curved track* model



Component

Name	rollingRight
Comment	

Model

Path	RailwayDynamics.General.Contact.Elas
Comment	elastic contact between wheel and rail

alpha	2.e-5		regularization parameter for profile evaluation
d	2e5	N.s/m	contact damping (scaled w.r.t. stiffness 2e8 N/m)
p_0	1.e-5	m	p_0 > 0: overlap value beyond which force law is regularized
s	cat(1, linspace	m	discretization of the wheel contour in lateral direction
wheelProfile	RailwayDynan		
railProfile	RailwayDynan		

nonLinearTangentialContact	false [✓ true]		
mue_0	0.36	1	maximum friction coefficient at zero slip velocity
A	0.38		ratio of friction coefficient
B	0.2	s/m	coefficient of expotentail friction decrease
k_A	0.9		friction reduction in adhesion area
k_S	0.5		friction reduction in slip area
eval_NonLinearTangentialContact	Polach Cor		

wheelRadius	0.46	m	Wheel radius
side	RailwayDynamics.General.Contact.Internal.Side.Right		
E	2.1e11	Pa	Young's modulus
nue	0.3	1	Poisson number

Figure 5. Excerpts from the menu to specify parameters of the *elastic Contact* model.

or two states, which represent the capability to move along a straight line. In the same manner, the *track Joint* defines one mechanical degree of freedom, but now refers to the *track* instance and presents the capability to move along the 3D path $\boldsymbol{r}$. The two associated states are $s_j = s_j(t)$ and $v_j = v_j(t)$, i.e. the instantaneous position along the path and its time derivative.

The local frame specified by $\boldsymbol{t}$, $\boldsymbol{n}$ and $\boldsymbol{b}$ uniquely assigns an orientation to each instantaneous position on the 3D track path and a given translational track speed $v_j = v_j(t)$ corresponds to a specific angular velocity $\boldsymbol{\omega}_j = \boldsymbol{\omega}_j(v_j(t), \boldsymbol{r}(s_j))$.

The railroad base in Figure 2 contains five *track Joint* components to represent the longitudinal degrees of freedom of the four wheelsets and the carbody.

3.3 Track Panel

As visualized in the animation in Figure 4, the *track panel* presents two rail stubs and one sleeper that are assumed to move in longitudinal direction associated to the wheelset. This is a quite common model simplification in railway dynamics in order to avoid the representation of the rail and the subgrade structure as a distributed system with many degrees of freedom and high computational demands. The flexibility of the rail road may then be introduced by modeling the *track panel* as a discrete spring-damper-mass system, which is parametrized on the basis of shaker measurements (Chaar and Berg, 2006). Thus, the mutual influence of neighbouring *track panels* through the rails and the subgrade is neglected.

Since the vehicle model considers four wheelsets, the railroad base in Figure 2 contains four *track panel* components, which are connected using vectors of MSL Multibody *frames*.

3.4 Wheelset

The *wheelset* model reproduces interia properties and contains two prismatic joints to enable lateral and vertical motion and three revolute joints to allow for roll, yaw and revolute motion. These five degrees of freedom complement the longitudinal motion already covered by the attached *track Joint*.

3.5 Wheel-Rail Contact

The wheel-rail contact component, which here is *elastic Contact*, is to be connected to the *wheelset* multibody frame that is located at the axle bearing position and the rail profile reference frame of the *track panel*. Figure 5 shows the *General* dialog menu tab, where wheel radius, Young's modulus, Poisson number and a side flag have to be specified.

The *Normal Contact* tab in Figure 5 specifies

- the smoothing parameter α, which is associated to a proposal of *Arnold* et al. to even profile curvature jumps, see (Arnold and Netter, 1998) or (Heckmann et al., 2014a),
- a contact damping parameter d,
- a parameter p_0 that helps to regularize the Hertzian contact algorithm, see (Heckmann and Grether, 2017),
- a vector $\boldsymbol{s}$ of lateral positions that samples the wheel contour in a number of discrete points,
- a reference to the wheel and rail profile geometry in the following manner:

```
import RailwayDynamics.General.Contact;
```

```
    replaceable package wheelProfile=
        Contact.Profile.S1002
      constrainedby
          Contact.Profile.partialProfile;
    replaceable package railProfile=
        Contact.Profile.UIC60
      constrainedby
          Contact.Profile.partialProfile;
```

These two profile packages may be replaced by the user, so that other standard or even measured profiles may be introduced, as long as the following base class is inherited:

```
partial package partialProfile "Specifies
    base class to introduce the geometry of
     arbitrary wheel or rail profiles"
  replaceable function evalProfile =
      partialEvalProfile;
  partial function partialEvalProfile
    "Function to evaluate wheel or rail
       profile"
    import SI = Modelica.SIunits;
    input Real s[sSize]
      "Lateral positions, where profile
         height is to be returned";
    input SI.Radius r0 "Nominal wheel
       radius (=0 for rail)";
    input Integer sSize
      "Number of positions and dimension of
          input and output vectors";
    output Real F[3,sSize]
      "Profile height and its 1st and 2nd
         derivative";
  end partialEvalProfile;
end partialProfile;
```

The *Tangential Contact* tab allows for

- switching between linear and nonlinear tangential contact evaluation,
- refering to a replaceable function to evaluate the nonlinear contact and may be user-defined as well,
- specifying parameters associated to the predefined nonlinear contact formulation according to *Polach*, see (Polach, 2005) or (Heckmann et al., 2014a).

In order to replace the *ElasticContact* in Figure 3, a *Constraint* contact model as described in (Heckmann et al., 2014a) and a *Simplified* contact component, which considers the wheel to present a conical profile that runs along a sharp edge rail are available in the library. The *Normal Contact* tab for this alternative contact models slightly differs to what is described above.

In addition, contact model classes to represent the rolling contact of a wheel to a roller as given in roller rigs are available in die subpackage *RailwayDynamics.General.Contact*. All contact models implemented so far consider the wheel and rail profile to touch each other at just one single point and exploit the usual assumptions in multibody vehicle modeling collected in Table 1 of (Heckmann et al., 2014a).

4 Example Applications

4.1 Traction

The goal of traction analysis is to calculate the acceleration and resulting in-train forces as well as longitudinal oscillations of coupled vehicles during traction and braking maneuvers. These are important investigations regarding safety, longitudinal comfort, fatigue of components, train control, vehicle stability and energy considerations, cf. (Spiryagin et al., 2014).

In Figure 6 an industrial scaled train model consisting of a locomotive and 4 cars is pictured. The vehicles are connected by coupling elements. The model is used to simulate the acceleration and deceleration phase of the train driven by the locomotive and to estimate the resulting in-train forces. A simple control unit sets the torques of the wheelsets of the locomotive in order to reach the target velocity.

In order to minimize the computational effort for this kind of simulations the *Longitudinal* subpackage offers models with a reduced number of lateral and vertical degrees of freedom. The train model in Figure 6 consists of models from the *General* subpackage (car1, grey) and from the *Longitudinal* subpackage to be distinguished by their light green icon fill color (locomotive and car2,3,4). In the longitudinal vehicle models the relative motions between carbody and bogie as well as between bogie and wheelset, except for the rotation of the wheelset, are neglected and therefore no suspension elements between these bodies are applied. Furthermore, the excitation caused by track irregularities is ignored. However, the vehicle model moves along the 3-D path defined by the track model and is therefore affected by downhill-slope forces and resistance forces caused by curvature of the track. The calculated velocity of the locomotive and the resulting in-train forces acting in *coupler1* are shown in Figure 7 (*assembly1*). One can observe peaks of the coupler forces caused by the impact of the cars on each other at the beginning of the acceleration and brake phases.

In order to compare the simulation results and the computational effort using models from the different subpack-

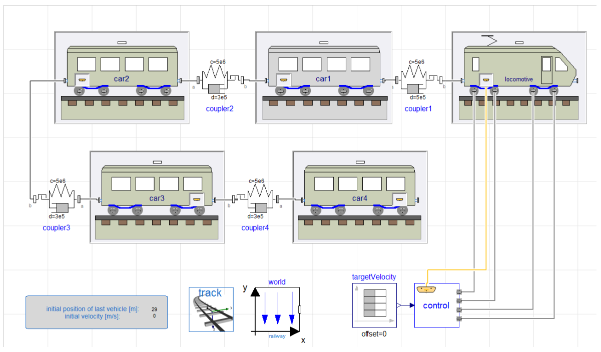


Figure 6. Mixed model built up of locomotive and cars from the *General* and the *Longitudinal* subpackage to simulate traction and braking maneuver (*assembly1*)

assembly	CPU-second / second	number of states
assembly1	6.65	157
assembly2	0.08	51
assembly3	69.50	605

Table 1. Table comparing computational effort and time states of three different model assemblies

ages, two more assemblies of the introduced train are built up and simulated. In *assembly2* all vehicle models of the train are from the *Longitudinal* subpackage and in *assembly3* all vehicle models are from the *General* subpackage. Figure 7 compares the simulation results of the three different model assemblies. It illustrates that the simulated vehicle speed and the peaks of the simulated coupler forces of all three model assemblies coincide.

A comparison of the computation time per simulated second and the number of continuous time states is shown in Table 1. The train model built up from the *Longitudinal* subpackage only (*assembly2*) leads to a simulation model with 51 states that computes the results within 8 s. The replacement of car1 by a model from the *General* subpackage, as shown in Figure 6 (assembly 1), leads to 157 states and a computation time of 665 s. The computation of *assembly2* using models from the *General* subpackage only takes 6950 s and uses 605 states.

This example demonstrates that the *Longitudinal* subpackage provides suitable models which are capable to analyze the longitudinal dynamics of trains by requiring only a fraction of the computational effort compared to a simulation of the train considering the entire vehicle dynamics. Furthermore, the models of this subpackage allow the analysis of very long trains, such as freight trains with up to 200 cars, with reasonable computational effort.

However, it is important to be aware of the limited scope of simulations using models of the *Longitudinal* subpackage. Due to the reduced number of degrees of freedom certain dynamics, such as pitch, yaw and roll of the bogie and carbody cannot be simulated. Thus, the simulation of scenarios in which this behavior might influence traction or braking of vehicles need to be carried out by using models that take these dynamics into account. A potential scenario is the reduction of traction or brake forces induced by pitching of the bogie which in turn decreases the normal contact forces between wheel and rail and in consequence the maximum transferable traction force.

The replacement strategy in Figure 6 also allows for mixed scenarios e.g. with one vehicle model from the *General* subpackage surrounded by several ones from the *Longitudinal* subpackage. The *General* model then provides a detailed insight, while the *Longitudinal* models are mainly intended to introduce the interaction with neighboring cars.

4.2 Comfort

As all technical components rails are non-ideal systems and exhibit irregularities, which induce vehicle vibrations. These track excitations are characterized by distance frequency components in power density spectra (PSD). In the RailwayDynamics Library the usual PSDs of ERRI and Frederich as well as own PSDs can be defined. Usual passenger trains therefore use two-level suspensions to reduce vibrations of the carbody to meet ride comfort targets. The lay-out of these suspensions is a significant engineering task, which can be done by simulation using full vehicle models from the *General* subpackage.

However in early design phases, it is a common assumption that the ride comfort is dominated only by vertical vibrations. This premise opens the opportunity to simplify vehicle models and reduce their computational needs in order to facilitate optimization studies. To this aim, the *Vertical* subpackage of the RailwayDynamics Library provides quarter and half vehicle models.

The example model of a quarter vehicle in Figure 8 consists of a track panel, a bogie and a car body model. The

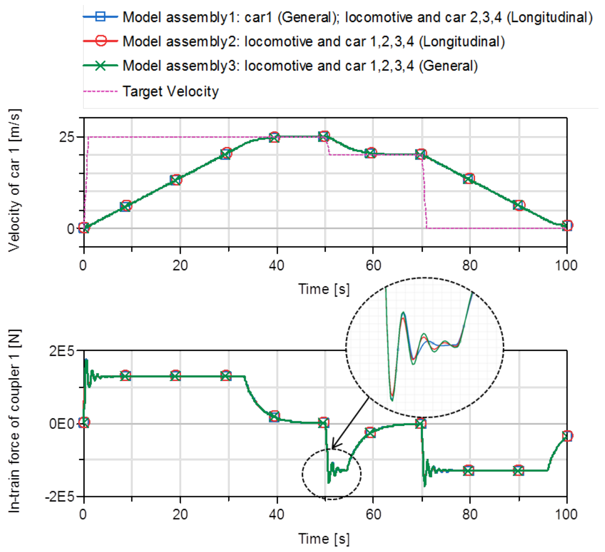


Figure 7. Comparison of simulation results of models from the *Longitudinal* subpackage and the *General* subpackage

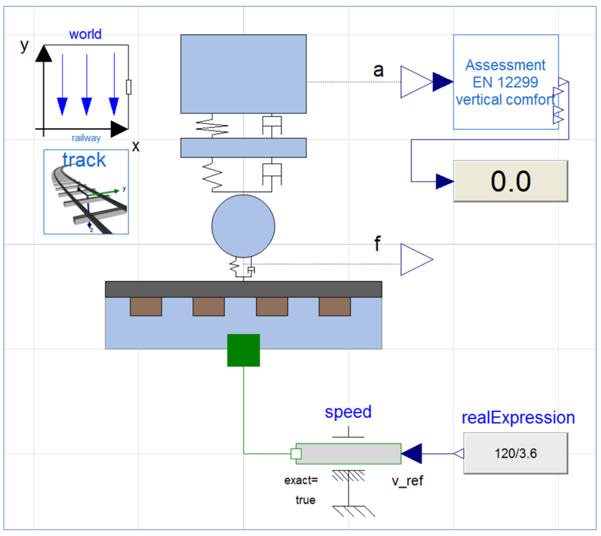


Figure 8. Vertical quarter vehicle model

position of the vehicle on the track is determined by an external input to the translational *flange* connector, since the longitudinal dynamics is neglected.

As mentionend in Section 3.3 the track panel is presented as a discrete sleeper mass, supported by a spring-damper system. The position of the rails follows the motion of the track panel, to which predefined irregularities are superimposed, cf. Section 3.1. The wheel-rail contact model here is simplified to a spring-damper system. The primary suspension that connects wheel and bogie mass and the secondary suspension between bogie and carbody mass are also spring-damper models with one vertical degree of freedom, each.

Several calculation methods can be used to express the human comfort perception of a rail vehicle in a performance index. The evaluation of the vibration comfort of a rail vehicle is defined in the (EN 12299: 2009) standard. The average comfort is expressed with the N_{MV} value, where the acceleration measurement is reshaped by various frequency filters, so that the N_{MV} number quantitatively expresses the human sensation of vibration comfort. To be more specific, values $N_{MV} < 1.5$ are characterized as very comfortable while values $N_{MV} \geq 4.5$ are assessed to be very uncomfortable.

In a first application example, an AVMZ-wagon, see (Iwnicki, 1998) runs at 120 km/h on a track with excitation. The vertical vibration comfort N_{MVz} is determined on the basis of a full-vehicle model from *General* subpackage and a quarter vehicle model shown in Figure 8.

The track irregularities are characterized by a customized polynomial with input parameters a_i and b_i:

$$S(\Omega) = \frac{b_0 + b_2\Omega^2}{a_0 + a_2\Omega^2 + a_4\Omega^4 + a_6\Omega^6}\,, \quad f = \frac{\Omega}{2\pi}\left[\frac{1}{m}\right]. \qquad (2)$$

Figure 9 shows the PSD $S(f)$ of the customized excitation as polynomial (blue) that is compared to two PSDs from literature called *ERRI high* and *ERRI low* (Bergander and Kunnes, 1993). For verification purposes, the customized irregularities were additionally measured from the simulation results and their PSD was reconstituted in cyan in Figure 9. As indicated by the signal drop at the

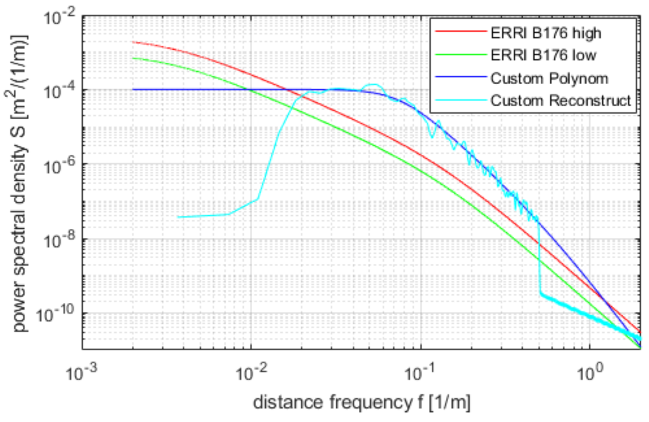


Figure 9. Comparison of track irregularities defined by power spectral densities (PSD)

edges of the reconstituted PSD, the customized excitation has been limited to $0.02 \leq f \leq 0.5$ $[1/m]$ by user input.

The vibration comfort of the full vehicle model is different along the carbody due to the contribution of its pitch motion to the local accelerations. While it is best in the middle, it is worst at the vehicle ends. The simulated vibration comfort of the quarter vehicle model shows an average comfort of the full-vehicle model. Reviewing the number of states in Table 2, it has to be taken into account that the various frequency filters according to (EN 12299: 2009) introduce 31 states into both model assemblies. The following table compares the simulation results of the two vehicle models:

Vehicle Model	states	CPU-s/s	N_{MVz}
full vehicle	171	9.08	0.43 ... 0.83
quarter vehicle	40	1.58	0.63

Table 2. Table comparing computational effort and time states of two different model assemblies

4.3 Roller Rig

The use of test rigs for railway research and development is widely spread in industry as well as at research institutes, see e.g. (Jaschinski et al., 1999). Even if test rigs cannot entirely replace track tests, in early design phases a test rig provides essential benefits, like cost effectiveness, repeatable testing conditions and an extended set of measurement equipment. To support this testing process the *General* subpackage contains all necessary components including specific contact models for the wheel-roller contact to build up a suitable simulation environment.

An animation of a model of a single wheelset on a roller rig is shown in Figure 11, to which Figure 10 presents the diagram layer. The so-called *uFrame* in green in Figure 11

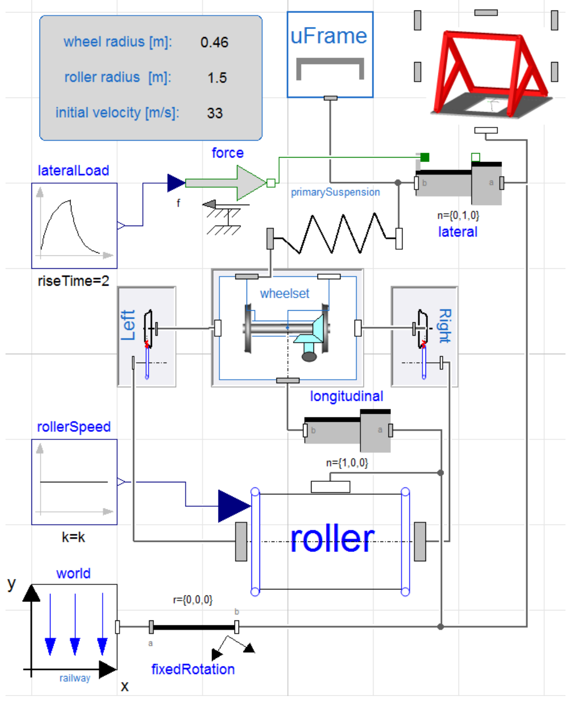


Figure 10. Diagram layer of a roller rig with one wheelset

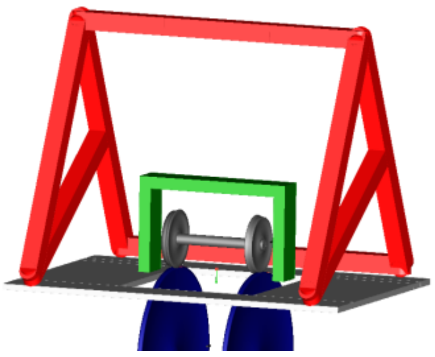


Figure 11. Animation of a roller rig with one wheelset

imitates the bogie while the *aFrame* in red represents the carbody in a simplified way.

If longitudinal investigations are of interest, the test rig can be extended by brake units and traction motors, respectively, to validate for example new wheel slide protection and anti-skid algorithms.

Besides the longitudinal analysis the roller rig environment can be used to evaluate the lateral dynamics, what is exemplarily presented in Figure 12. The two pictures result from a test rig simulation with a lateral force excitation (blue line in the upper plot) on the uFrame. The red line denotes the resulting force in the primary spring, which clearly shows the interdependence of the higher frequency wheel-roller contact forces.

In the lower plot the lateral displacements of the wheelset (green) and the uFrame (black) are illustrated and the typical hunting motion can be recognized especially by the wheelset behavior. The difference between these two signals is the relative deviation of the primary spring. In the end, this scenario allows to verify or even optimize the dynamic stability of the wheelset e.g. under a crosswind disturbance or an other lateral influence.

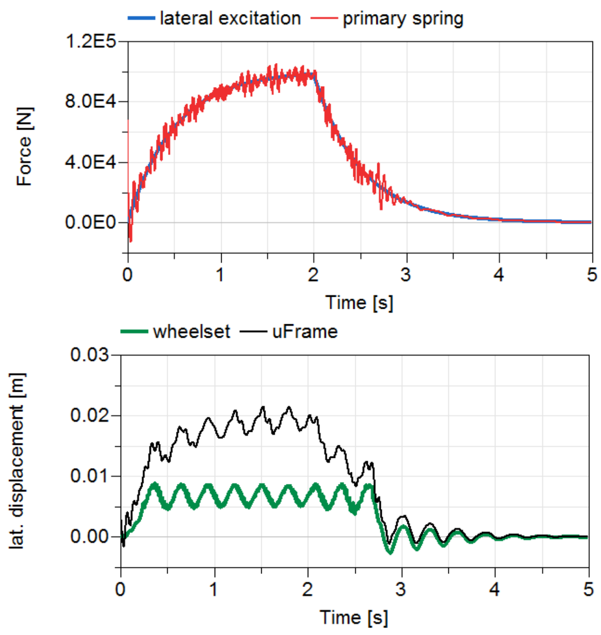


Figure 12. Simulation results of a laterally excited wheelset on the roller rig

4.4 Crosswind Stability

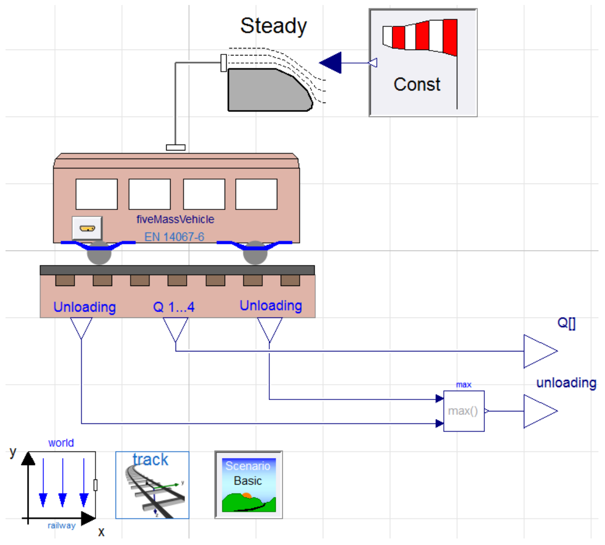


Figure 13. Diagram layer of the quasistatic crosswind model according to EN 14067-6

Crosswind stability addresses the risk that vehicles running on high speed are prone to overturning, if high crosswinds occur. The assessment of this risk is part of the vehicle acceptance procedure, regulated by (TSI HS RST 2008) and (EN 14067-6: 2010). One of several assessment scenarios defined in these references refers to a simplified five-mass model, which is therefore predefined in the RailwayDynamics Library. Figure 13 shows the diagram layer of this quasistatic model. The reader is referred to (Heckmann and Grether, 2017), where a detailed discussion on the crosswind stability issue and further modeling approaches are given.

5 Multi-domain Modeling

5.1 The VehicleInterfaces Library reloaded

As already mentioned in Section 1.2, the RailwayDynamics Library is intended to facilitate the multi-domain modeling of railway vehicles. Therefore, the present paper makes the attempt to initiate a discussion on the appropriate organization of multi-domain railway vehicle models. The multi-domain railway model in Figure 14 is inspired

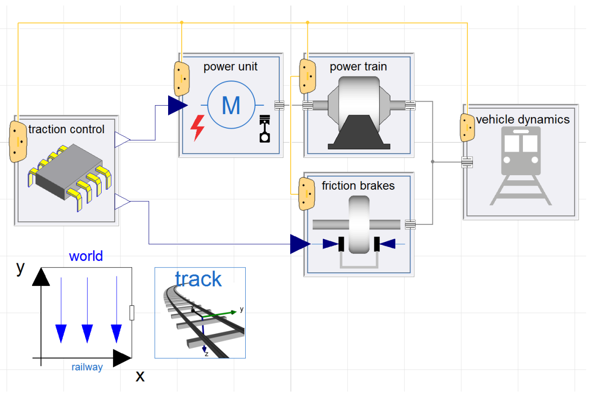


Figure 14. First option to organize multi-domain modeling

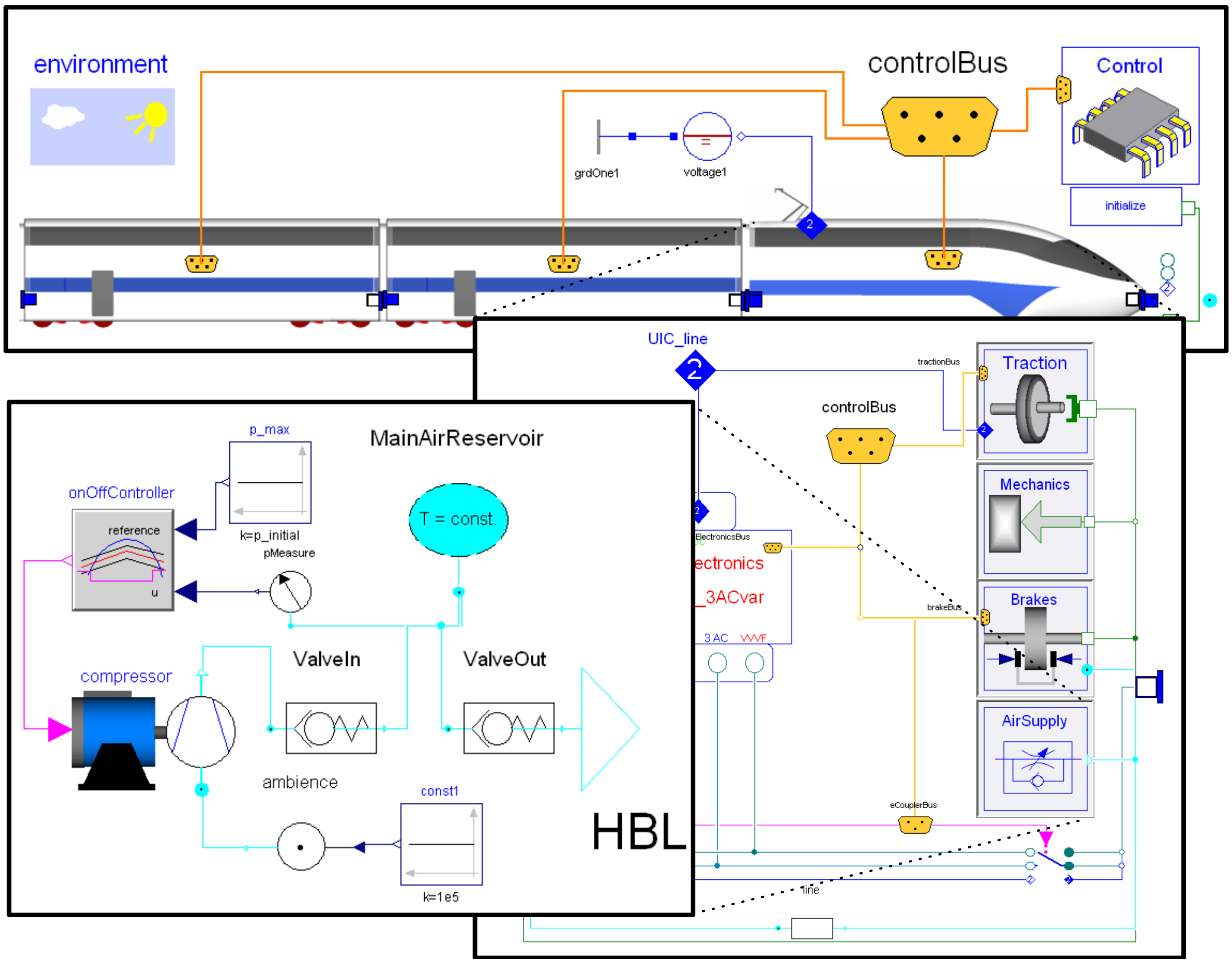


Figure 15. Alternative option to organize multi-domain modeling from (Carrarini et al., 2010)

by a corresponding activity in the automotive field that led to the definition of the VehicleInterfaces library (Dempsey et al., 2006).

There, the issues on railway vehicle dynamics are presented by one submodel depicted by one icon. Additional submodels cover propulsion systems, power train and brake modeling and introduce control algorithms. Following this scheme, it is easy to replace submodels e.g. in order to adapt their detail level. External supplier companies may provide submodels of their domain and the interconnection of the submodels may be organized on the model top level.

However, railway vehicles actually are train sets, where several cars are connected at buffers. Each single car is a multiphysical system on its own. Therefore, the scheme in Figure 14 actually presents an aggregation of single-domain train sets, one vehicle dynamics train set, one train set of propulsion systems, one for brakes, etc. A vector of *flange with bearing* connectors propagates traction or braking torques to the wheelsets, cf. Figure 2.

5.2 Alternative Approach

Figure 15 originates from a former project proposal for auxiliary systems in trains. There, each car is a multi-domain model with (electric) traction, air supply, mechanical and brake subsystem. The component view of the air supply subsystem is shown in Figure 15 as well. The list of optional subsystems may be further extended, e.g. to consider the energy supply of air conditioning subsystems or the control of door systems.

A newly specified multi-domain buffer connector was defined that connects single cars, e.g. by connecting the pneumatic line of the leading car with the pneumatic line of the trailing one and so on.

This approach is assumed to rely on a more elaborate specification of all stakeholders such as OEMs and suppliers on interfacing and multilevel modeling organization compared to the proposal in Section 5.1. The multi-domain overhead an engineer has to keep in view while working on his or her single-domain task might be larger. The authors of the present paper are curious where a discussion on this issue may lead to.

6 Summary and Outlook

This paper presents the newly released DLR RailwayDynamics Library, which is intended to provide a sound modeling framework dedicated to vehicle dynamics and running gear design. The consideration of vehicle dynamics issues in multi-domain engineering tasks is a specific focus of the RailwayDynamics Library.

Already initiated and future applications of the library concern the synthesis of advanced observer and control lay-outs, (Schwarz et al., 2018), (Heckmann et al., 2016), and multidisciplinary simulation tasks such as the interaction of running dynamics and drive train, the systems engineering of pneumatic brake systems (Ehret, 2018) and research on advanced system design and assessment scenarios in order to ensure the crosswind stability of railway vehicles, cf. (Heckmann et al., 2014b).

Acknowledgment

A very early version of the subpackage of the RailwayDynamics Library that is specialized to crosswind stability was implemented by Dr. Antonio Carrarini during his period of employment at DLR.

References

M. Arnold and H. Netter. Approximation of contact geometry in the dynamical simulation of wheel-rail. *Mathematical and Computer Modelling of Dynamical Systems*, 4(2):162–184, 1998.

B. Bergander and W. Kunnes. ERRI B176/DT 290: B176/3 Benchmark Problem, Results and Assessment. Technical report, European Rail Research Institute, 1993.

A. Carrarini, A. Heckmann, I. Kaiser, B. Kurzeck, J.L.R̃eyes Pérez, and L. Valente. Multidisciplinary applications of multibody simulation to railway vehicle engineering. In *IMSD 2010*, 2010. URL https://elib.dlr.de/64436/.

N. Chaar and M. Berg. Simulation of vehicle–track interaction with flexible wheelsets, moving track models and field tests. *Vehicle System Dynamics*, 44(sup1):921–931, 2006. doi:10.1080/00423110600907667.

M. Dempsey, M. Gäfvert, P. Harman, C. Kral, M. Otter, and P. Treffinger. Coordinated automotive libraries for vehicle system modelling. In *Proceedings of the 5th International Modelica Conference, Vienna*, pages 33–41, 2006.

M. Ehret. Modelca library for the systems engineering of railway brakes. In *Proceedings of the American Modelica Conference 2018*, 2018.

EN 12299: 2009. Railway Applications -Ride comfort for passengers - measurement and evaluation, 2009.

EN 14067-6: 2010. Railway Applications -Aerodynamics- Requirements and test procedures for crosswind assessment., 2010.

F. Frederich. Die Gleislage aus fahrzeugtechnischer Sicht. *ZEV–Glasers Annalen*, pages 108–1984, 1984.

A. Haigermoser, B. Luber, J. Rauh, and G. Gräfe. Road and track irregularities: measurement, assessment and simulation. *Vehicle System Dynamics*, 53(7):878–957, 2015. doi:10.1080/00423114.2015.1037312.

A. Heckmann and G. Grether. The DLR RailwayDynamics Library: the Crosswind Stability Problem. In *Proceedings of the 12th International Modelica Conference*, pages 623–631, 2017. doi:10.3384/ecp17132623.

A. Heckmann and S. Streit. The modeling of energy flows in railway networks using xml-infrastructure data. In *Proceedings of the 9th International Modelica Conference*, pages 125–132, 2012. doi:10.3384/ecp12076125.

A. Heckmann, A. Keck, I. Kaiser, and B. Kurzeck. The Foundation of the DLR RailwayDynamics Library: the Wheel-Rail-Contact. In *Proceedings of the 10th International Modelica Conference*, pages 465–475, 2014a. doi:10.3384/ECP14096465.

A. Heckmann, B. Kurzeck, T. Bünte, and S. Loose. Considerations on active control of crosswind stability of railway vehicles. *Vehicle System Dynamics*, 52(6):759–775, 2014b. doi:10.1080/00423114.2014.901539.

A. Heckmann, C. Schwarz, T. Bünte, A. Keck, and J. Brembeck. Control development for the scaled experimental railway running gear of DLR. In *24th Symposium of the International Association for Vehicle System Dynamics (IAVSD 2015)*. CRC Press, 2016.

S. Iwnicki. The Manchester Benchmarks for rail simulators - an introduction. *Vehicle System Dynamics*, 29(sup1):717–722, 1998. doi:10.1080/00423119808969598.

A. Jaschinski, H. Chollet, S. Iwnicki, A. Wickens, and J. Würzen. The application of roller rigs to railway vehicle dynamics. *Vehicle System Dynamics*, 31(5-6):345–392, 1999.

O. Polach. Creep forces in simulations of traction vehicles running on adhesion limit. *Wear*, 258(7):992–1000, 2005. doi:10.1016/j.wear.2004.03.046.

C. Schwarz, A. Heckmann, and A. Keck. Different models of a scaled experimental running gear for the DLR RailwayDynamics Library. In *Proceedings of the 11th International Modelica Conference*, pages 441–447, 2015. doi:10.3384/ecp15118441.

C. Schwarz, J. Brembeck, and B. Heckmann. Dynamics observer for the longitudinal behavior of a wheelset on a roller rig. *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, 2018. submitted for publication.

M. Spiryagin, C. Cole, Y.Q. Sun, M. McClanachan, V. Spiryagin, and T. McSweeney. *Design and simulation of rail vehicles*. CRC Press, 2014. doi:10.1201/b17029.

TSI HS RST 2008. 2008/232/EC: Commission Decision of 21 February 2008 concerning a technical specification for interoperability relating to the rolling stock sub-system of the trans-european high-speed rail system, 2008.

W. Weber. Die Gleisbogenachse als räumliches Kurvenstück. *ETR*, 39(1/2):79 – 81, 1990.

Using Baumgarte's Method for Index Reduction in Modelica

Scott A. Bortoff[1]

[1]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA, bortoff@merl.com

Abstract

We show by example how Baumgarte's method can be used in a Modelica model to reduce the differential algebraic equation index prior to compilation. This has advantages for some constrained mechanical systems especially those with closed-chain kinematics, including improved initialization and enabling model-based control system design. We derive models for a simple pendulum, a delta robot and for elevator cable sway as case studies. The models are used for simulation and also for dynamic analysis and to design and realize feedback controllers.

Keywords: DAE, index reduction, robotics, control

1 Introduction

Modeling and simulation of some types of constrained mechanical systems, such as closed kinematic chains, can be challenging in the Modelica language. One reason is because component-oriented modeling for such systems results in a set of high-index differential algebraic equations (DAEs). Modelica compliers, such as Dymola, use the method of "dummy derivatives" (Mattsson and Söderlind, 1993; Bachmann, 2006; Cellier, 2006) to reduce the index for very good and fundamental reasons. However, for closed chains it has some disadvantages, and there are other methods (Bauchau and Laulusa, 2007), which have advantages especially for consistent initialization and use cases beside simulation, such as control system design.

In this paper we show, by example, how Baumgarte's method of index reduction (Baumgarte, 1972, 1983) can be used in Modelica to reduce the index of a constrained mechanical system *prior* to compilation. Our primary example is a delta robot, for which we derive a singularity-free, index 1 DAE. No automatic index reduction is done at compile time, and no dynamic state selection is required at simulation time. We find that the method is amenable to Modelica's object oriented modeling paradigm, and results in simulation code that can be, at least anecdotally, faster. We construct several components of a feedback controller directly from the index-1 system model, and show how consistent initial conditions can be computed in this formulation. We provide a second example, elevator cable sway, in which the method is vital to simulation and feedback control system design. Interestingly, the method can model certain types of time-varying constraints such as loss-of contact or constraint breaking.

Baumgarte's method should be considered as a viable *alternative* - not a general replacement - to the automatic index reduction algorithms that are built into Modelica compilers. It is appropriate for certain situations in which these algorithms either fail to reduce the index, or result in complex and therefore slow, simulation code. The method has been criticized in the numerical analysis literature (Bauchau and Laulusa, 2007), primarily because selection of values for its parameters, described later, is problem-dependent, and because it results in a system of equations that is of dimension larger than the number of degrees of freedom in the problem. As a result, a simulation can "drift," meaning that the algebraic constraint is not enforced exactly during a simulation. For some use cases, this could be disastrous and the method should not be used. However, for our applications, we find these criticisms to be inconsequential. The method's two parameters are easy to tune, and the drift is on the order of the solver tolerance, so it can be reduced by reducing the solver tolerance. The drift vanishes when the mechanical system is at rest.

2 Toy Pendulum Example

Consider the equations of motion of a simple pendulum expressed in Cartesian coordinates,

$$\dot{x}_1 = v_1 \tag{1a}$$
$$\dot{x}_2 = v_2 \tag{1b}$$
$$M\dot{v}_1 = -2x_1\lambda \tag{1c}$$
$$M\dot{v}_2 = -2x_2\lambda - g \tag{1d}$$
$$0 = h(x) = x_1^2 + x_2^2 - L^2 \tag{1e}$$

where M is the pendulum bob mass, L is the rod length, g is the acceleration due to gravity, $x = [x_1\ x_2]^T$ is the position in Cartesian coordinates of the pendulum bob, $v = [v_1\ v_2]^T$ is the velocity, and λ is the Lagrange multiplier which corresponds to the tension in the rod required to maintain the constraint $h(x) = 0$ in (1e). System (1) is an index-3 DAE in variables x, v and λ. Modelica code for the pendulum is (Fritzon, 2015)

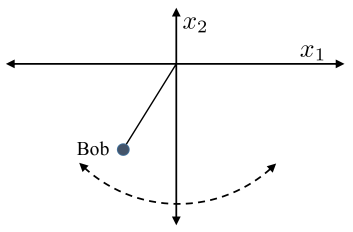
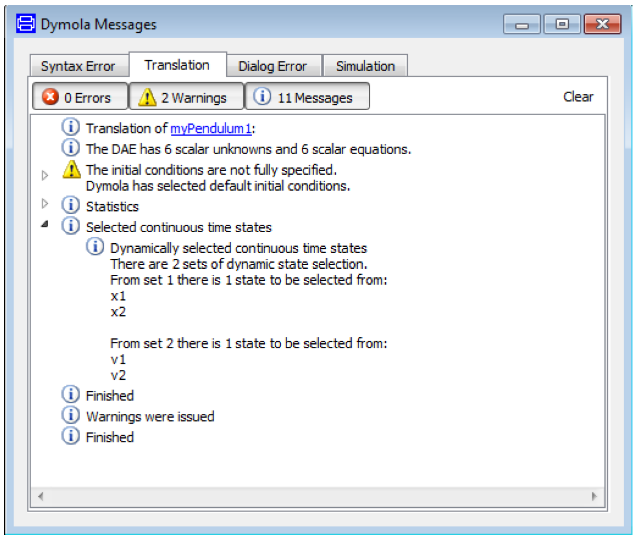


Figure 1. Pendulum.

```
der(x1) = v1;
der(x2) = v2;
M * der(v1) = -2.0 * x1 * lambda;
M * der(v2) = -2.0 * x2 * lambda - M * g;
h = x1^2 + x2^2 - L^2;
h = 0.0;
```

When this is compiled by Dymola, for example, the index is reduced using the method of "dummy derivatives," resulting in a system with two differential states and three algebraic states. However, for any single choice of differential states, there exists a kinematic configuration in which the solver Jacobian becomes singular. This means that at least two representations are required to cover the complete configuration space, and the solver switches between them. Figure 2 shows the Message window after compiling this model, indicating that two sets of two dynamics states were selected.



```
Translation of myPendulum1:
The DAE has 6 scalar unknowns and 6 scalar equations.
The initial conditions are not fully specified.
Dymola has selected default initial conditions.
Statistics
Selected continuous time states
    Dynamically selected continuous time states
    There are 2 sets of dynamic state selection.
    From set 1 there is 1 state to be selected from:
    x1
    x2

    From set 2 there is 1 state to be selected from:
    v1
    v2
Finished
Warnings were issued
Finished
```

Figure 2. Message Window showing two sets of two dynamic states for the method of "dummy derivatives."

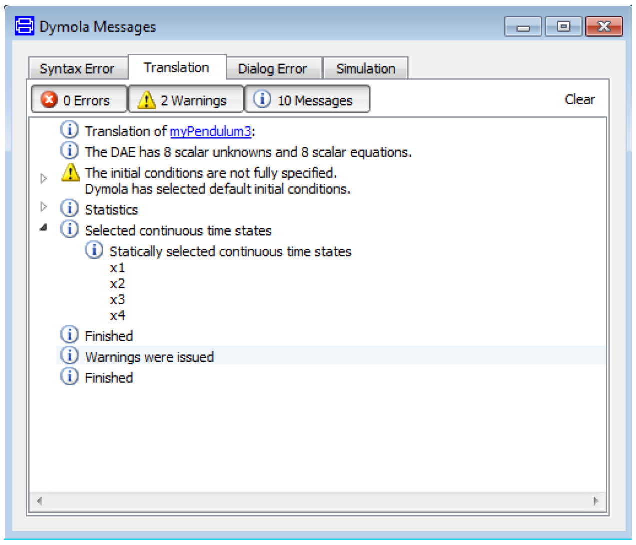
Baumgarte's method replaces (1e) with a linear combination of its first two derivatives,

$$h''(x,v,\lambda)+\alpha_1 h'(x,v)+\alpha_0 h(x)=0\,, \qquad (2)$$

where $\alpha_i > 0$ for $i = 0, 1$, and $s^2 + \alpha_1 s + \alpha_0$ is Hurwitz (all roots in the open left-half plane). Values for α_i are tuned depending on the specific problem. Large values have smaller drift, but result in a stiff system. We find that placing the roots at locations that are on the order of the system time constant is sufficient. The resulting system (1a)-(1d) and (2) is an index-1 DAE that has the same solution as (1), which is shown below, and can be numerically integrated with an index-1 solver such as DASSL. Modelica code for the pendulum reduced via Baumgarte's method is

```
der(x1) = v1;
der(x2) = v2;
M * der(v1) = -2.0 * x1 * lambda;
M * der(v2) = -2.0 * x2 * lambda - M * g;
h0 = x1^2 + x2^2 - L^2;
h1 = der(h0);
h2 = der(h1);
0.0 = h2 + alpha1 * h1 + alpha0 * h0;
```

where we take advantage of Modelica's automatic differentiation. Figure 3 shows the message window for Baumgarte's method. We see that four static states are selected.



```
Translation of myPendulum3:
The DAE has 8 scalar unknowns and 8 scalar equations.
The initial conditions are not fully specified.
Dymola has selected default initial conditions.
Statistics
Selected continuous time states
    Statically selected continuous time states
    x1
    x2
    x3
    x4
Finished
Warnings were issued
Finished
```

Figure 3. Message Window showing one set of four dynamic states for Baumgarte's method.

Simulation of this model is about 5x faster than the first system for the same simulation parameters, but it is less accurate. Figure 4 shows the constraint $h(x)$ for a portion of the simulation. Baumgarte's method drifts away from $h(x) = 0$ by an amount on the same order as the solver tolerance (1e-4). On the other hand, the method of dummy derivative implicitly enforces $h(x) = 0$ for all time, which is one reason why it is used in compilers.

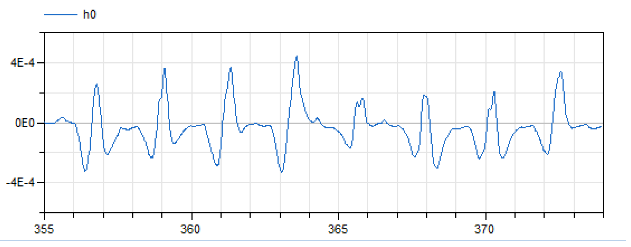


Figure 4. Constraint $h(x)$ for the pendulum example, Baumgarte's method.

It is useful to understand the geometric structure of the index-1 system (1a)-(1d) and (2). Define $z_0 = h(x)$ and $z_1 = h'(x,v)$. Following (Isidori, 1989), define $\xi = [z_0\; z_1]^T \in \mathbb{R}^2$ to be the "linear" part. Then there exist coordinates $\eta \in \mathbb{R}^2$ which are functions of x and v (after eliminating λ through algebraic manipulation) so that (1a)-(1d) and (2) can be written locally in so-called Zero Dynamics

Normal Form (Isidori, 1989),

$$\dot{\eta} = f(\eta, \xi) \tag{3a}$$
$$\dot{\xi} = A\xi, \tag{3b}$$

where the two eigenvalues of A are located at the roots of

$$s^2 + \alpha_1 s + \alpha_0 = 0, \tag{4}$$

and the two-dimensional zero dynamics

$$\dot{\eta} = f(\eta, 0) \tag{5}$$

are the dynamics of the pendulum. In other words, we simulate a four-dimensional system with state $[x\ v]^T \in \mathbb{R}^4$, but there is an attractive two-dimensional manifold in $\mathbb{R}^4$, defined by $\xi = 0$, on which the pendulum dynamics exist and evolve according to (5). The ξ-dynamics are exponentially stable, and once they converge to 0, do not affect x or v. This has two important implications. First, we may initialize the system at a state $[x_0\ v_0]^T \in \mathbb{R}^4$ nearby $\xi = 0$, and the state it will converge exponentially to the constraint manifold $\xi = 0$. This can be useful to compute consistent initial conditions by starting with an inconsistent initial condition and simulating the system until the exponentially stable part has converged. Second, if we linearize (1a)-(1b) and (2), we expect two pole-zero cancellations at the roots to (4). In a control design situation, these modes are exponentially stable, and are uncontrollable and unobservable, and may therefore be removed with a Hankel-norm model truncation (Skogestad and Postlethwaite, 2005) because their corresponding Hankel singular value is zero. The resulting reduced-order model is two-dimensional (because we started with a system of dimension four, and removed the two modes) and is equivalent to a linearization obtained otherwise, e.g., if we reduced the index using the method of dummy derivatives and then linearized it.

2.1 Breaking Pendulum

One advantage that Baumgarte's method offers is simulation of breaking constraints, which is an example of multimode modeling (Elmqvist et al., 2017). Consider the situation in which the pendulum rod will fracture if its tension exceeds a threshold. This situation is difficult to model using conventional methods, because the index changes from 3 to 0 when the rod breaks. It can be modeled with Baumgarte's method because the number of equations and variables remains constant before and after the break.

```
der(x1) = x3;
der(x2) = x4;
M * der(x3) = rhsX;
M * der(x4) = rhsY;
if lambda < lambdaMax then
 rhsX = -2.0 * x1 * lambda;
 rhsY = -2.0 * x2 * lambda - M * g;
 0.0 = h2 + alpha1 * h1 + alpha0 * h0;
else
 rhsX = 0.0;
 rhsY = -M * g;
 lambda = lambdaMax + epsilon;
end if;
h0 = x1^2 + x2^2 - L^2;
h1 = der(h0);
h2 = der(h1);
```

(Note that the value of `lambda` should be zero after the break, but we set it to an arbitrary `lambdaMax+epsilon` to avoid switching back after the break.) Figure 5 shows the result of a simulation.

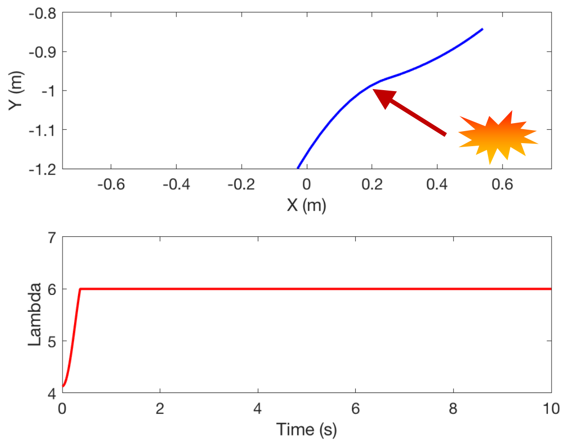


Figure 5. Simulation of breaking pendulum, in (x, y)-coordinates (top), and the Lagrange multiplier (bottom) .

3 Delta Robot Model

Next we use the same method to derive a model of a delta robot (Clavel, 1990) pictured in Figure 6, consisting of three symmetric arms constrained kinematically by universal joints at the end effector. Each arm consists of a proximal link, rigidly attached to a servomotor shaft at the proximal end, and a pair of parallel distal links that are attached to the proximal link via a pair of universal joints. The six distal links are attached to the end effector by universal joints such that the pair of arm distal links remain parallel, and the orientation of the end effector is invariant.

Delta robots are closed-chain mechanisms. Unlike the kinematics of serial chain robots (Spong and Vidyasagar, 2004), the forward kinematics of the delta robot (the function from actuated joint angles to the location of the end effector) cannot be expressed analytically (Merlet and Gosselin, 2008), making formulation of dynamic (and inverse dynamic) equations of motion more difficult (Guglielmetti, 1994; St. and C., 2003; Merlet and Gosselin, 2008; Brinker et al., 2015).

We derive the robot dynamics as in (Bortoff, 2018) first by defining the dynamics for each independent arm, assuming it is unconstrained, and then adding the holonomic coupling constraint representing the end effector. The resulting index-3 DAE is stabilized using Baumgarte's method, giving an index-1 DAE.

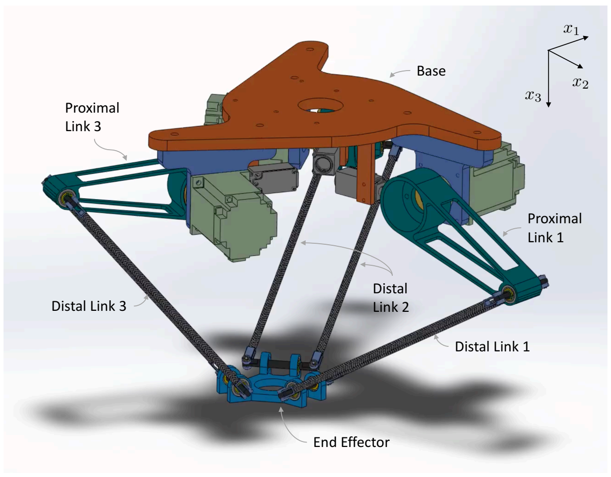


Figure 6. Delta robot.

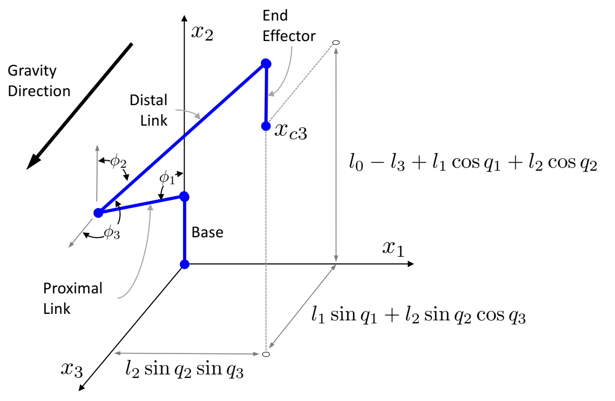


Figure 7. Delta robot arm coordinates with end effector location x_{c3} indicated.

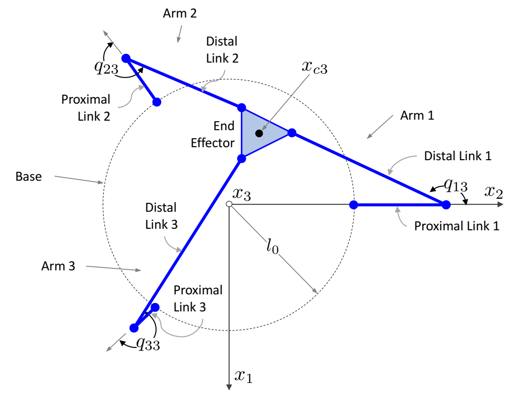


Figure 8. Delta robot coordinates, bottom view, looking up.

3.1 Arm Dynamics

In deriving the dynamics of each arm, we can lump together the two distal links into a single effective link. Referring to Figures 6-7 in which the fixed "world" frame has axes labeled $[x_1, x_2, x_3]$, let $\phi = [\phi_1, \phi_2, \phi_3]^T$ be the generalized angular position for the arm, defined as follows. The servomotor angle is ϕ_1, which is the rotation of the proximal link about the x_1-axis, measured with respect to the x_2-axis. The universal joint position is represented with ϕ_2 representing the rotation *about* the x_1-axis measured with respect to the x_2-axis, and ϕ_3 representing the rotation *about* the x_2-axis measured with respect to the $x_2 - x_3$ plane. Note that, in these coordinates, the universal joint has a singularity at $\phi_2 = 0$. However, this is outside the range of motion of the robot once the three arms are kinematically constrained by the end effector.

Assuming that the distal links are thin rods, i.e., neglecting the inertia of the distal link about its longitudinal axis, the kinetic energy of each arm, including 1/3 the mass of the end effector, is

$$\begin{aligned} T(\phi, \dot{\phi}) &= \frac{1}{2} m_1 \dot{x}_{c1}^T \dot{x}_{c1} + \frac{1}{2} m_2 \dot{x}_{c2}^T \dot{x}_{c2} + \frac{1}{6} m_3 \dot{x}_{c3}^T \dot{x}_{c3} \\ &+ \frac{1}{2} J_1 \dot{\phi}_1^2 + \frac{1}{2} J_2 \left(\sin(\phi_2)^2 \dot{\phi}_3^2 + \dot{\phi}_2^2 \right), \quad (6) \end{aligned}$$

where the position of the center of mass of the proximal link is

$$x_{c1} = \begin{bmatrix} 0.0 \\ l_{c1} \cos(\phi_1) \\ l_{c1} \sin(\phi_1) \end{bmatrix}, \quad (7)$$

the position of the center of mass of the distal link is

$$x_{c2} = \begin{bmatrix} l_{c2} \sin(\phi_2) \sin(\phi_3) \\ l_1 \cos(\phi_1) + l_{c2} \cos(\phi_2) \\ l_1 \sin(\phi_1) + l_{c2} \sin(\phi_2) \cos(\phi_3) \end{bmatrix}, \quad (8)$$

the position of the center of mass of the end effector is

$$x_{c3} = \psi(\phi) := \begin{bmatrix} l_2 \sin(\phi_2) \sin(\phi_3) \\ l_0 - l_3 + l_1 \cos(\phi_1) + l_2 \cos(\phi_2) \\ l_1 \sin(\phi_1) + l_2 \sin(\phi_2) \cos(\phi_3), \end{bmatrix}, \quad (9)$$

the velocities $\dot{x}_{c1}$, $\dot{x}_{c2}$ and $\dot{x}_{c3}$ are computed by the chain rule to be functions of ϕ, $\dot{\phi}$ and the parameters are listed in Table 1. Note that the forward kinematics of the arm are defined as $\psi(\phi)$ in (9). The gravitational potential energy of each arm is

$$\begin{aligned} V(\phi) &= -g((l_{c1} m_1 + l_1 (m_2 + m_3/3)) \sin(\phi_1) \\ &+ (l_{c2} m_2 + l_2 m_3/3) \sin(\phi_2) \cos(\phi_3), \quad (10) \end{aligned}$$

where gravity points along the positive x_3 axis and 1/3 of the mass of the end effector is included in each arm. The Lagrangian

$$L(\phi, \dot{\phi}) = T(\phi, \dot{\phi}) - V(\phi) \quad (11)$$

is used to define the arm equations of motion with Lagrange's equation,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\phi}} - \frac{\partial L}{\partial \phi} = bu, \quad (12)$$

Table 1. Delta robot parameter definitions.

Symbol	Description (Units)
l_0	Base radius (m)
l_1	Length of proximal link (m)
l_2	Length of distal link (m)
l_3	Width of end effector (m)
l_{c1}	Distance to proximal link center of mass (m)
l_{c2}	Distance to distal link center of mass (m)
m_1	Mass of proximal link (kg)
m_2	Mass of distal mass (kg)
m_3	Mass of end effector (kg)
J_1	Rotational inertia, proximal link ($\text{kg}\cdot\text{m}^2$)
J_2	Rotational inertia, distal link ($\text{kg}\cdot\text{m}^2$)

giving

$$m(\phi)\ddot{\phi}+c(\phi,\dot{\phi})+g(\phi)=bu, \tag{13}$$

where m is the 3×3 inertia matrix, c is the 3×1 vector of Coriolis and centripetal torques, g is the 3×1 vector of torques due to gravity, $b=[1,0,0]^T$ and u is the servomotor torque. Expressions for m, c and g are given in Appendix 1.

3.2 Robot Lagrangian Dynamics

Each of the three arms is identical except for a 120° rotation about the z-axis. To represent the dynamics of the full robot, we sum the unconstrained Lagrangians for each arm (11), and augment the result with the holonomic constraints that equate the x_{c3} positions of the end effectors of each arm (9) in the world coordinates. Lagrange's equation gives the constrained dynamical equations.

Referring to Figure 8, define $q_i\in\mathbb{R}^3$ for $1\le i\le 3$, to be the generalized angular position of each of the three arms, replacing the ϕ-notation used in Section 3.1. Define $q=[q_1,q_2,q_3]^T\in\mathbb{R}^9$ and the unconstrained Lagrangian as

$$L_u(q,\dot{q})=L(q_1,\dot{q}_1)+L(q_2,\dot{q}_2)+L(q_3,\dot{q}_3),$$

and form the augmented robot Lagrangian as

$$L_a(q,\dot{q})=L_u(q,\dot{q})+\lambda^T h(q), \tag{14}$$

where the constraint $h(q):\mathbb{R}^9\to\mathbb{R}^6$ is

$$h(q)=\left[\begin{array}{c}\psi(q_1)-R_z(2\pi/3)\cdot\psi(q_2)\\ \psi(q_1)-R_z(-2\pi/3)\cdot\psi(q_3)\end{array}\right], \tag{15}$$

the rotation matrix

$$R_z(\theta)=\left[\begin{array}{ccc}\cos(\theta)&-\sin(\theta)&0\\ \sin(\theta)&\cos(\theta)&0\\ 0&0&1\end{array}\right], \tag{16}$$

ψ is defined in (9), and $\lambda\in\mathbb{R}^6$ is a vector of Lagrange multipliers. Then the Lagrangian equations of motion for the robot are

$$\begin{aligned}\frac{d}{dt}\frac{\partial L_a}{\partial\dot{q}}-\frac{\partial L_a}{\partial q} &= \lambda^T H(q)+Bu &(17)\\ h(q) &= 0, &(18)\end{aligned}$$

where

$$H(q)=\frac{\partial h(q)}{\partial q}.$$

Defining $v=\dot{q}$, (17)-(18) can be written as a set of 24 first-order DAEs of Index 3 (Brenan et al., 1996; Kunkel and Mehrmann, 2006), in the variables $q\in\mathbb{R}^9$, $v\in\mathbb{R}^9$ and $\lambda\in\mathbb{R}^6$,

$$\begin{aligned}\dot{q} &= v &(19)\\ M(q)\dot{v}+C(q,v)+G(q) &= \lambda^T H(q)+Bu &(20)\\ h(q) &= 0, &(21)\end{aligned}$$

where

$$\begin{aligned}M(q) &= \operatorname{diag}(m(q_1),m(q_2),m(q_3))\in\mathbb{R}^{9\times 9},\\ C(q,v) &= \operatorname{diag}(c(q_1,v_1),c(q_2,v_2),c(q_3,v_3))\in\mathbb{R}^9,\\ G(q) &= \operatorname{diag}(g(q_1),g(q_2),g(q_3))\in\mathbb{R}^9,\\ B &= \operatorname{diag}(b,b,b)\in\mathbb{R}^{9\times 3}.\end{aligned}$$

Equations (19) - (21) are a complete dynamic model of the delta robot, but index reduction is necessary for simulation and application of modern control theory.

3.3 Robot Hamiltonian Dynamics

For some applications such as port-Hamiltonian analysis (van der Schaft, 2013) it is useful to have a Hamiltonian model of the robot. This is derived in similar fashion by defining the momentum vector $p\in\mathbb{R}^9$ and the Hamiltonian $H=T+V$ for each arm, augmenting the constraint (15) by the Lagrange multiplier λ and solving the Hamiltonian equations, resulting in

$$\begin{aligned}M(q)\dot{q} &= p &(22)\\ \dot{p} &= \frac{1}{2}v\frac{\partial M(q)}{\partial q}v-G(q)+Bu+H^T(q)\lambda &(23)\\ h(q) &= 0, &(24)\end{aligned}$$

where the partial derivatives of M need to be computed symbolically. This formulation has about the same computational complexity as (19)-(21), results in similar numerical solutions using the same type of solver, but could be used with a symplectic solver for speedup.

3.4 Index Reduction

Following the same approach from Section 2, the constraint (21) is replaced with a linear combination of its first two derivatives with respect to time. Define

$$\begin{aligned}z_0 &= h(q) &(25)\\ z_1 &= \dot{z}_0=\frac{\partial H(q)}{\partial q}\dot{q} &(26)\\ z_2 &= \dot{z}_1=\dot{H}(q)\dot{q}+H(q)M^{-1}(q)\left(\lambda^T H(q)\right.\\ &\quad \left.+Bu-C(q,\dot{q})-G(q)\right), &(27)\end{aligned}$$

and replace (21) or (24) with

$$z_2+\alpha_1 z_1+\alpha_0 z_0=0, \tag{28}$$

where $s^2 + \alpha_1 s + \alpha_0$ is a Hurwitz polynomial (all roots in the open left-half plane). The model (19)-(20) and (28) or (22)-(23) and (28), is an index 1 DAE with 18 differential equations, 6 algebraic equations and 24 states q, v and λ, or q, p and λ, respectively.

It is interesting to express the dynamics in Zero Dynamics Normal Form, as we did for the pendulum. Following (Isidori, 1989), we define $\xi = [z_0\, z_1]^T \in \mathbb{R}^{12}$ to be the "linear" part. Then there exist coordinates $\eta \in \mathbb{R}^6$ which are functions of q, v and u (after algebraically eliminating λ) so that (19)-(20) and (28) can be written locally in Zero Dynamics Normal Form (Isidori, 1989),

$$\dot{\eta} = f(\eta, \xi, u) \tag{29a}$$

$$\dot{\xi} = A\xi, \tag{29b}$$

where the 12 eigenvalues of A are located at the roots of (28), and the 6-dimensional zero dynamics

$$\dot{\eta} = f(\eta, 0, u) \tag{30}$$

are the dynamics of the robot. In other words, there is a 6-dimensional manifold defined by $\xi = 0$ on which the robot dynamics exist and evolve according to (30). The ξ-dynamics are exponentially stable, are not controllable from u, and once they converge to 0, do not affect q or v. This means that if we linearize (19)-(20) and (28), we expect to see 12 poles and zeros at the roots to (28), and these dynamics are neither controllable nor observable. They are easily removed using a Hankel-norm model truncation. The resulting reduced-order model is six dimensional and equivalent to a linearization obtained otherwise.

In practice, expressions for z_1 and z_2 in (26)-(27) are computed automatically using the `der(·)` operator. Also, because the model is an index 1 DAE (instead of an index 0 ODE), it is not necessary to compute the inverse of the inertia matrix for either the Lagrangian or Hamiltonian formulations. Further, it is not necessary to compute η or f in (29a)-(29b). Deriving these expressions is done to understand the geometric structure and properties.

The primary disadvantages of Baumgarte's method are that 24 equations in 24 variables are produced, instead of the minimal six (although λ can be removed by algebraic manipulation, leaving 18 implicit first-order differential equations in 18 differential variables), and that numerical solutions to (19)-(20) and (28) will drift off the constraint manifold $h = 0$ when the system is in motion. However, for this application we find the drift to be small, is computable for monitoring purposes, and controllable in the sense that it is reduced by reducing the solver tolerance. Moreover, simulation times for (19)-(20) and (28) are an order-of-magnitude faster than the model that results from index reduction by the dummy derivative method, despite the fact that we require three times more equations and dynamic states, due to the simplicity of the equations.

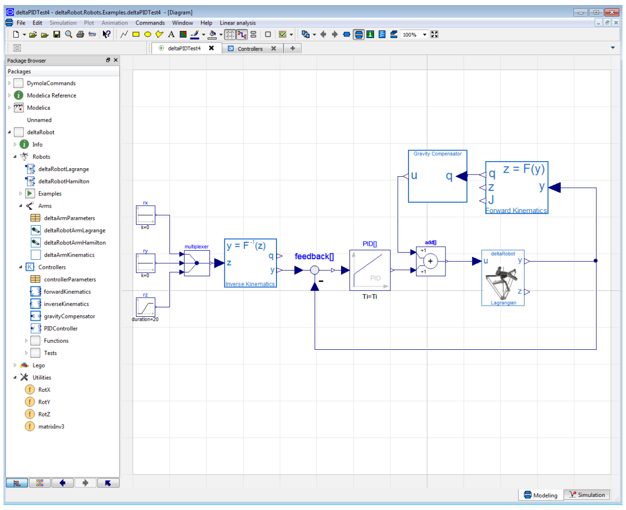


Figure 9. Screenshot of the Modelica deltaRobot library (left) and an a gravity-compensating PID feedback controller (right), showing the use of forward and inverse kinematics, gravity compensation and PID. The library contains models of the kinematics at the lowest level, arms, and robots at its highest level. We also have a package of controller components and a growing library of tasks, such as assembling Lego.

4 Modelica Library

We have created a Modelica library including models of the delta robot, various control algorithms that are derived from the model, and assembly tasks such as stacking blocks and assembling Lego bricks. A screen shot of the library is shown in Figure 9. For the delta robot models, the library is organized as a hierarchy, with partial models of the kinetics and parameters at the lowest level, extended into full models of the arms at the intermediate level, and models of the full robot at the highest level. We provide partial code listings of these components in the Appendix. At a higher level, multiple robots can be declared, and constraints among them defined in a manner analogous to what we have done for the arms. This allows for analysis of cooperative control using the same mathematics and approach. We remark that this is difficult using the Modelica standard library, because constraint forces acting on different parts of the end effector, for example, are difficult to introduce. The Lagrangian approach provides a natural way for additional constraint forces to be introduced, making this formulation more natural and effective when developing force and assembly control algorithms.

In the subsections that follow, we describe some of the control system blocks that we have constructed from the DAE model, each of which is realized as a functions using algorithm blocks.

4.1 Forward Kinematics

The forward kinematics function takes as input the three joint measurements at the servos and computes the other six joint angles (which are unactuated and unmeasured), and the location of the end effector in world coordinates. The robot Jacobian is also computed. The forward kinematics are one-to-one but not onto, and defined implicitly by (15), which needs to be solved numerically. Specifically, partition q into measured and unmeasured states by defining $y = [q_{11}, q_{21}, q_{31}]^T$ to represent the measured joint angles, and $x = [q_{12}, q_{13}, q_{22}, q_{23}, q_{32}, q_{33}]^T$ to represent the unmeasured joint angles, and rearrange the variables of h so that (15) can be written

$$h(x,y) = 0. \tag{31}$$

This is solved for x using Newton's method

$$\frac{\partial h}{\partial x}(x_k,y) \cdot (x_{k+1} - x_k) = -h(x_k,y), \tag{32}$$

which typically converges to 7 decimal places of accuracy in 2-3 iterations since it can be initialized close to its solution in a real-time application. Each iteration requires the solution to a 6-dimensional set of linear equations. With the solution (x,y), the end effector location is computed using ψ in (9), and the robot Jacobian is also computed.

4.2 Inverse Kinematics

The inverse kinematics takes as input a location of the end effector $w \in \mathbb{R}^3$ and computes values for the joint angles $q \in \mathbb{R}^9$. This is not one-to-one: there is not a unique solution for all values of w. The inverse kinematics defined implicitly by the nine equations

$$\psi(q_i) - w = 0 \tag{33}$$

for $i = 1, 2, 3$. This is solved using Newton's method with some logic for choosing the desirable solutions. Each Newton iteration involves computing the solution to three 3-dimensional linear systems of equations, making the complexity less than the forward kinematics.

4.3 Gravity Compensation

One popular control scheme is to cancel the effect of gravity on the manipulator with an inner loop, and then close an outer feedback loop with a PD or PID compensator. The gravity compensating feedback is computed as the solution to the 9-dimensional set of linear equations

$$\begin{bmatrix} u \\ \lambda \end{bmatrix} \cdot [B\, H^T(q)] = G(q), \tag{34}$$

where in any real-time application q is computed via the forward kinematics from the joint measurements y. A closed-loop model including a delta robot, gravity compensation and using forward and inverse kinematics is shown in Figure 9.

4.4 Feedback Linearization

A feedback linearizing control law can be defined as follows. Let

$$w_1 = \psi(q_1) \tag{35}$$

denote the location of the end effector. Symbolically differentiate this twice

$$\begin{aligned} w_2 &= \dot{w}_1 = d\psi(q_1)v_1 & (36)\\ \dot{w}_2 &= d\dot{\psi}(q_1)v_1 + d\psi(q_1)\dot{v}_1. & (37) \end{aligned}$$

Solving (20) for $\dot{v}$ and substituting the result into (37) gives

$$\dot{w}_2 = \alpha(q) + \beta(q) \cdot u$$

from which the control law

$$u = \frac{1}{\beta(q)}\left(-\alpha(q) - k_1 w_1 - k_2 w_2 + w_r\right)$$

renders the system linear from w_r to w_1. Expressions for α and β can be computed automatically. They require inversion of the 9×9 inertia matrix M, which is not difficult because it is block diagonal.

5 Linear Control Design and Analysis

The model (19)-(20), (28) and control functions described in the previous section, realized in the deltaRobot Modelica library, enable dynamic analysis and model-based design of new control algorithms for various tasks related to pick-and-place and robotic assembly. Here we show some results of an example dynamic analysis. We compute the linearization of the delta robot using values for parameters that are measured from a delta robot in our laboratory, at the equilibria $q_{i1} = 0$rad, meaning that the proximal links are all horizontal. A pole-zero plot is shown in Figure 10. First, notice that there are 12 pole-zero cancellations at $s = -5$ corresponding to the dynamics of (29a), as expected. These do not affect the input-output behavior and can be eliminated from the linear system by a Hankel norm truncation. Perhaps surprisingly, this configuration is open-loop unstable. Note that this configurations is well within the reachable workspace. (The unstable root crosses into the right-half plane at an angle of approximately $q_{i1} = 22°$, for our robot.) This kind of instability is a common characteristic of robotic manipulators, and has important consequences. For example, stabilizing feedback gains have lower limits (Skogestad and Postlethwaite, 2005). In some applications such as fine force control, it is common practice to reduce feedback gains to maintain stability during contact. But the lower bound means that this practice has has limits, which are not obvious without a model-based analysis.

6 Elevator Cable Sway

Modeling elevator cable sway is another example where we have applied Baumgarte's method. The system is diagrammed in Figure 11. The traveling cable, which supplies power and signals to the car, is attached to the bottom of the car at one end, and the inside of the elevator shaft at the other. The cable experiences horizontal motion ("sway") when the car moves or when the building

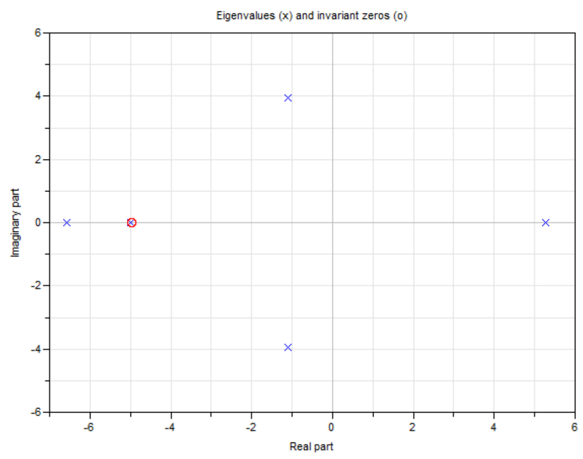


Figure 10. Pole-zero plot of the delta robot in equilibrium with $q_1 = 0$rad for the three proximal links. There are 12 pole-zero pairs at $s = -5$ corresponding to the dynamics of (28). The plot shows four poles at approximately $s = -2 \pm j$, one at $s = -6.5$, and an unstable pole at $s = 5.2$.

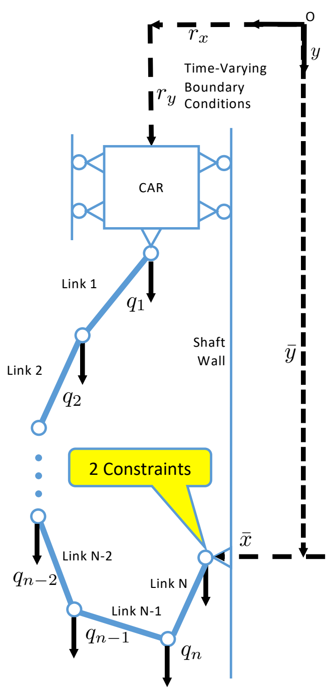


Figure 11. Elevator Cable Sway.

sways due to wind or earthquake. Because it can be damaged by striking the wall, we design a feedback controller attenuate the cable sway by moving the car.

The system can be modeled as a constrained chain of rigid links with springs and dampers between each link (Tomaszewski and Pieranski, 2005),

$$\dot{q} = v \tag{38a}$$
$$M(q)\dot{v} + C(q)v^2 + Dv + G(q) + Kq + a(q)\ddot{r}_x + b(q)\ddot{r}_y = \lambda H^T(q) \tag{38b}$$
$$h(q) = 0 \tag{38c}$$

where

$$h(q) = \left[\ \bar{x} - \sum_{k=1}^{N} l \sin(q_k) \quad \bar{y} - \sum_{k=1}^{N} \cos(q_k)\ \right]^T, \tag{39}$$

$q \in \mathbb{R}^N$ is the vector of link angles, $v \in \mathbb{R}^N$ is the vector of angular velocities, M, C, D, K and G are the inertia, centripetal, damping and gravity matrices, respectively, $\ddot{r}_x$ and $\ddot{r}_y$ are the x and y acceleration of the frame marked “O,” respectively, $\lambda \in \mathbb{R}^2$ is the Lagrange multiplier vector, $h = 0$ represents the constraint of the chain attached to the wall at locations $\bar{x}$ and $\bar{y}$, and N is the number of links, typically $N = 100$.

Equation (38) is a DAE of index 3, and we reduce the index exactly as we did earlier, replacing the constraint h with a linear combination of its first two derivatives. The resulting index-1 model is then used for simulation and feedback control design. The details are omitted for space reasons, and we present the results of one particular feedback controller which takes as input a single measurement of horizontal cable displacement, filters the measurement through a lead compensator which is designed using a frequency response computed from the model, and applies the output to the car motion controller. In Figure 12 we see the horizontal displacement of the car due to building sway that is caused by an earthquake, when the controller is off. This causes the cable to sway. In Figure 13, the feedback controller is engaged 50s after the earthquake begins, and moves the car up and down for a period of 300s, attenuating the cable sway by 75%.

We remark that a model of an *open* chain is elementary to construct from the Modelica Standard Library (MSL) and has been used for benchmarking (Casella, 2015). However, we have not been successful in modeling the *constrained* chain using the MSL, because the index reduction fails for large values of N. Even if it did compile, consistent initialization would be a challenge. On the other hand, using Baumgarte's method, we are able to compile models with $N > 200$ and can initialize the DAE using the procedure outlined in Section 2.

7 Conclusion

In this paper we show how Baumgarte's method of index reduction can be used in a Modelica model of a constrained mechanical systems. The method reduces the model index prior to compilation, so that the model does not undergo automatic index reduction by the compiler. Baumgarte's method has some advantages over the “dummy derivative” method that is integrated into Modelica compilers for some models. It may be easier to compute consistent initial conditions, the derived models can be used directly to derive model-based control algorithms, and simulations may run faster. On the other hand, the method does not enforce constraints exactly, and drift occurs during simulations. We find, however, that this drift is not consequential for our mechatronic applications, and in

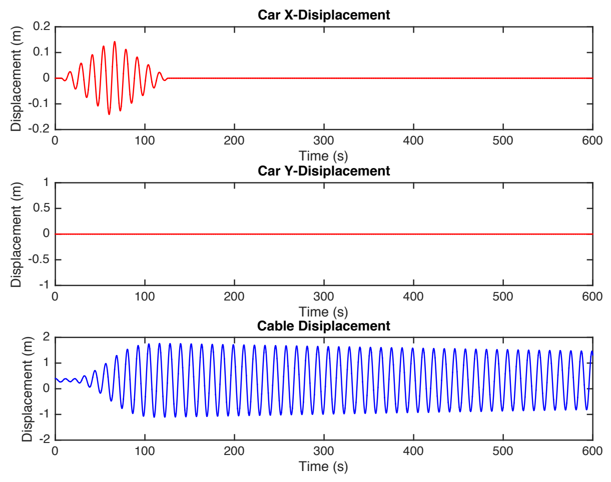


Figure 12. Car x (top) and y (middle) motion, and elevator cable sway (bottom) due to earthquake.

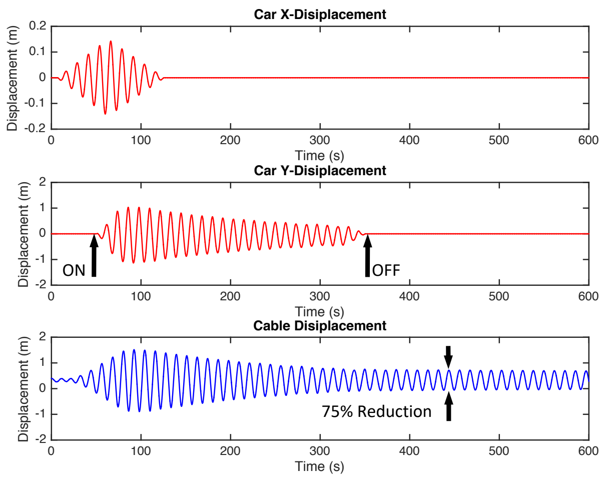


Figure 13. Car x (top) and y (middle) motion, under feedback control, and elevator cable sway (bottom) during earthquake.

fact the method allows for compilation and simulation of some models that otherwise cannot compile and initialize. We believe the method may find successful application in other domains, particularly for problems in which consistent initial conditions are difficult to compute.

A Delta Robot Modelica Model

The Delta robot arm kinematics are defined in the following partial Modelica model.

```
partial model deltaArmKinematics
 deltaArmParameters p;   // Parameters
 Real q[3],psi[3],dpsi[3,3];
equation
 psi[1]=p.L2*sin(q[2])*sin(q[3]);
 psi[2]=p.L3-p.L0+p.L1*cos(q[1])...
  +p.L2*cos(q[2]);
 psi[3]=p.L1*sin(q[1])+...
  p.L2*sin(q[2])*cos(q[3]);
 // Gradient of end effector location ...
 dpsi[1,1]=0.0;
 dpsi[1,2]=p.L2*cos(q[2])*sin(q[3]);
 dpsi[1,3]=p.L2*sin(q[2])*cos(q[3]);
 dpsi[2,1]=-p.L1*sin(q[1]);
 dpsi[2,2]=-p.L2*sin(q[2]);
 dpsi[2,3]=0.0;
 dpsi[3,1]=p.L1*cos(q[1]);
 dpsi[3,2]=p.L2*cos(q[2])*cos(q[3]);
 dpsi[3,3]=-p.L2*sin(q[2])*sin(q[3]);
end deltaArmKinematics;
```

Arm dynamics are defined extending the kinematics model. These expressions are computed in *Mathematica* and exported via scripts, automatically generating the Modelica code.

```
model deltaRobotArmLagrange
extends deltaArmKinematics;
Real v[3], tau[3];
protected
Real M[3,3],C[6,3],G[3];
equation
// Inertia Matrix...
m[1,1]=p.J1+p.LC1^2*M1+p.L1^2*(p.M2+p.M3);
m[1,2]=p.L1*(p.LC2*p.M2+p.L2*p.M3)...
 *(cos(q[1])*cos(q[2])*cos(q[3])...
 +sin(q[1])*sin(q[2]));
m[1,3]=-p.L1*(p.LC2*p.M2+p.L2*p.M3)...
 *cos(q[1])*sin(q[2])*sin(q[3]);
m[2,1]=m[1,2];
m[2,2]=p.J2+p.M2*p.LC2^2+p.M3*L2^2;
m[2,3]=0.0;
m[3,1]=m[1,3];
m[3,2]=0.0;
m[3,3]=(p.J2+p.M2*p.LC2^2+p.M3*p.L2^2)*sin(
    q[2])^2;
// Centripetal and Coriolis vectors...
c[1,1]=0.0;
c[1,2]=p.L1*(p.LC2*p.M2+p.L2*p.M3)...
 *(cos(q[1])*sin(q[2])-cos(q[2])*cos(q[3])*
    sin(q[1]));
c[1,3]=p.L1*(p.LC2*p.M2+p.L2*p.M3)...
 *sin(q[1])*sin(q[2])*sin(q[3]);
c[2,1]=p.L1*(p.LC2*p.M2+p.L2*p.M3)...
 *(cos(q[2])*sin(q[1])-cos(q[1])*cos(q[3])*
    sin(q[2]));
c[2,2]=0.0; c[2,3]=0.0;
c[3,1]=-(p.L1*(p.LC2*p.M2+p.L2*p.M3)...
 *cos(q[3])*cos(q[1])*sin(q[2]));
c[3,2]=-(p.J2+p.LC2^2*p.M2+p.L2^2*p.M3)....
 *cos(q[2])*sin(q[2]);
c[3,3]=0.0; c[4,1]=0.0;
c[4,2]=0.0; c[4,3]=0.0;
c[5,1]=-2.0*p.L1*(p.LC2*p.M2+p.L2*p.M3)...
 *cos(q[1])*cos(q[2])*sin(q[3]);
c[5,2]=0.0;
c[5,3]=(p.J2+p.LC2^2*p.M2+p.L2^2*p.M3)*sin
    (2*q[2]);
c[6,1]=0.0; c[6,2]=0.0; c[6,3]=0.0;
// Gravity vector...
G[1]=-p.g*(p.LC1*p.M1+p.L1*(p.M2+p.M3))...
 *cos(q[1]);
G[2]=-p.g*(p.LC2*p.M2+p.L2*p.M3)...
 *cos(q[2])*cos(q[3]);
G[3]= p.g*(p.LC2*p.M2+p.L2*p.M3)...
 *sin(q[2])*sin(q[3]);
// Arm Dynamics...
```

```
der(q) = v;
m*der(v)+c[1,:]*v[1]^2+c[2,:]*v[2]^2...
 +c[3,:]*v[3]^2+c[4,:]*v[1]*v[2]...
 +c[5,:]*v[2]*v[3]+c[6,:]*v[1]*v[3]...
 +G+p.DAMPING.*v = tau;
end deltaRobotArmLagrange;
```

Below is the Lagrangian robot model. The Hamiltonian version is similar. Note that the derivatives of h are computed automatically.

```
model deltaRobotLagrange
Arms.deltaRobotArmLagrange arm1,arm2,arm3;
Real lambda[6];
Real h0[6],h1[6],h2[6];
Input Real u[3];
parameter Real POLE=5.0;
constant Real Rot2[3,3] = Utilities.RotZ
    (2.0*PI/3.0);
constant Real Rot3[3,3] = Utilities.RotZ
    (-2.0*PI/3.0);
constant Real B[3] = {1, 0, 0};
equation
// tau = H^T(q) * lambda...
arm1.tau=transpose(arm1.dpsi)*lambda[1:3]
 +transpose(arm1.dpsi)*lambda[4:6]+B*u[1];
arm2.tau=-transpose(Rot2*arm2.dpsi)*...
 lambda[1:3]+B*u[2];
arm3.tau=-transpose(Rot3*arm3.dpsi)*...
 lambda[4:6]+B*u[3];
// Baumgarte's method of index reduction...
h0=cat(1,arm1.psi-Rot2*arm2.psi,...
 arm1.ps -Rot3*arm3.psi);
h1=der(h0);
h2=der(h1);
zeros(6)=h2+2.0*POLE*h1+POLE^2*h0;
end deltaRobotLagrange;
```

We remark that the index-3 model can be constructed by replacing the last line with

```
h0=zeros(6);
```

which will compile in Dymola using the "dummy derivative" method for index reduction. The result is two sets of DAEs with some switching logic.

References

Bernhard Bachmann. Mathematical aspects of object-oriented modeling and simulation. In *Proceedings of the 5th International Modelica Conference*, 2006.

Olivier A. Bauchau and André Laulusa. Review of contemporary approaches for constraint enforcement in multibody systems. *Journal of Computational and Nonlinear Dynamics*, 2007.

J. W. Baumgarte. Stabilization of constraints and integrals of motion in dynamic systems. *Computer Methods in Applied Mechanics and Engineering*, 1:1–16, 1972.

J. W. Baumgarte. A new method of stabilization for holonomic constraints. *ASME Journal of Applied Mechanics*, 50:869–870, 1983.

Scott A. Bortoff. Object-oriented modeling and control of delta robots. In *IEEE Conference on Control Technology and Applications*, pages 251–258, 2018.

K. E. Brenan, S. L. Cambell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, 1996.

J. Brinker, B. Corves, and M. Wahle. A comparative study of inverse dynamics based on clavel's delta robot. In *Proceedings of the 14th IFToMM World Congress*, Oct. 2015.

Francesco Casella. Simulation of large-scale models in modelica: State of the art and future perspectives. In *Proceedings of the 11th International Modelica Conference*, pages 459–468, 2015.

Francois E. Cellier. *Continuous System Simulation*. Springer, 2006.

Francois E. Cellier and Jurden Greifeneder. *Continuous System Modeling*. Springer, 1991.

R. Clavel. Device for the movement and positioning of an element in space. U.S. Patent 4, 976, 582, Dec. 11 1990.

Hilding Elmqvist, Toivo Henningsson, and Martin Otter. Innovations for future modelica. In *Proceedings of the 12th International Modelica Conference*, pages 693–702, 2017.

Peter Fritzon. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley, 2015.

Philippe Guglielmetti. *Model-Based Control of Fast Parallel Robots: A Global Approach in Operational Space*. PhD thesis, Ecole Polytechnique Federale de Lausanne, 1994.

Alberto Isidori. *Nonlinear Control Systems*. Springer-Verlag, 1989.

Peter Kunkel and Volker Mehrmann. *Differential-Algebraic Equations: Analysis and Numerical Solution*. European Mathematical Society, 2006.

Sven Erik Mattsson and Gustaf Söderlind. Index reduction in differential algebraic equations using dummy derivatives. *SIAM Journal on Scientific Computing*, 14(3), 1993.

Jean-Pierre Merlet and Clement Gosselin. *Springer Handbook of Robotics*, chapter Parallel Mechanisms and Robots. Springer, 2008.

Sigurd Skogestad and Ian Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. Wiley, 2005.

M. M. Spong and M. Vidyasagar. *Robot Dynamics and Control*. Wiley, 2004.

Staicu St. and Carp-Ciocardia D. C. Dynamic analysis of clavel's delta parallel robot. In *Proceedings of the 2003 International Conference on Robotics and Automation*, pages 4116–4121, 2003.

Waldemar Tomaszewski and Piotr Pieranski. Dynamics of ropes and chains: 1. the fall of the folded chain. *New Journal of Physics*, 7(45), 2005.

A.J. van der Schaft. *Surveys in Differential-Algebraic Equations I*, chapter Port-Hamiltonian Differential-Algebraic Systems, pages 173–226. Springer, 2013.

Modeling of Rotating Shaft with Partial Rubbing

Tatsuro Ishibashi[1] Tadao Kawai[2]

[1]Meidensha Corporation, Japan, ishibashi-tat@mb.meidensha.co.jp
[2]Department of Mechanical & Physical Engineering, Osaka City University, Japan, kawai@osaka-cu.ac.jp

Abstract

We have created the rotating machinery library to carry out analytical investigations for diagnosis by transfer matrix method in Modelica. In this paper, rubbing components for partial rub are implemented in our rotating machinery library. In this research, the model in which the rotor come into contact from a non-contact state by a pulsating external force is analyzed. The relationship between the contact configurations and the generation of various kinds of vibration is investigated. We validated the rubbing model in one side contact case with a rotor kit. By simulation, we reproduced the time history, the orbit and the full spectrum characteristics of the rotating shaft measured by the experiment precisely.

Keywords: Rotor Dynamics, Rubbing, Contact, Friction, Subharmonics

1 Introduction

To improve efficiency in rotating machinery, the clearance between rotors and casings has become smaller and smaller. However, it increases risk of rubbing i.e. contact between rotating and stationary elements of a machine. It is mainly resulted from the mass unbalance, turbine or compressor blade failure, defective bearing, or rotor misalignment. The rub-impacting vibration of a rotor system shows a very complicated phenomenon including not only the periodic motion but also the quasi-periodic and chaotic motions. When the rub-impact happens, the partial rub arises at first. During a whole period, the rub and impact interactions occur between rotors and stators (i.e. casings) once or fewer times. Gradual deterioration of the partial rub will lead to the full rub, and then the vibration will affect the normal operation of the machines negatively. Thus, the rubbing phenomenon is one of the main malfunctions in rotating machines and causes the breakdown of machines.

Because of serious damage of rubbing, many researchers have studied this problem from different aspects (Ehrich, 1966; Beatty, 1985; Choi and Noah, 1987). Much attention has been given to the nonlinear dynamics of the rub-impacting rotor system. A contact force of rubbing between a rotor and a casing has been modeled as a piecewise linear spring and damper model. The relationship between the contact configurations and the generation of various kinds of vibration, such as "collision type synchronous vibration", "sub harmonic vibration", etc. has been studied, both theoretically and experimentally (Watanabe *et al*, 2004; Watanabe *et al*, 2005).

We have created the rotating machinery library by transfer matrix method in Modelica (Ishibashi *et al*, 2017). By transfer matrix method, the rotating shaft is decomposed into rotors, shafts, journals, couplings, housings and supports. The 5 DOF rotor dynamics model components have common faults of rotating machinery systems such as static and dynamic unbalance, shaft bending, and faulty bearing. Basic components are reusable, and their parameters can be simply modified. Even if it is not a Jeffcott (i.e. symmetrical) rotor system, this library makes it easy to analyze dynamics of rotating machinery. The objective of creating this library is to carry out analytical investigations in order to gain some insight into the diagnostics of rotating machinery.

Many papers have been written regarding modeling contact phenomena in the Modelica language. The contact models can be roughly classified into two types, collision of multibody objects and contact of gears. The former is handled in the following papers. A solution based on a collision handling software called Solid was described in (Otter *et al*, 2005). The paper (Oestersötebier *et al*, 2014) introduced non-central contact blocks in which the contact surfaces were defined. (Hofmann *et al*, 2014) discussed the use of the Bullet Physics Library.

The latter is handled in the following papers. One work is (van der Linden, 2012) where the 3 DOF elastic gear contact model was implemented in the Planar Mechanics library. A much more detailed approach was taken by (Kosenko and Gusev, 2011) and further improved in (Kosenko and Gusev, 2012), where the forces between gears were modelled with high detail in a Modelica environment. (Dahl *et al*, 2017) integrated the gear contact model in the MultiBody library from the Modelica Standard library.

In this paper, rubbing components for partial rub are implemented in our rotating machinery library. Rubbing

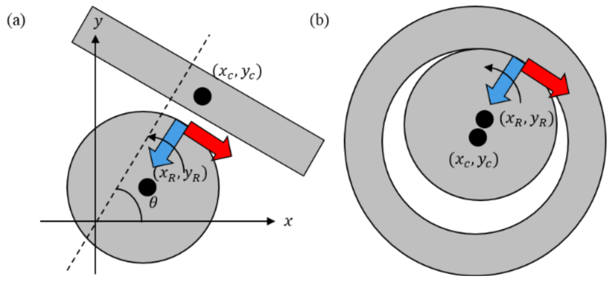


Figure 1. Type of Rubbing. (a) One side contact case. (b) Annular contact case.

components in the one side contact case and the annular contact case are created for analyzing the several contact configurations respectively.

In the analysis of rubbing, models in which the rotor come into contact with the casing due to unbalance or models in which rotor is already contacting with the casing at rest are usually analyzed. However, in this research, the model in which the rotor come into contact from a non-contact state by an external force due to earth quake or flow-induced vibration is analyzed. The relationship between the contact configurations and the generation of various kinds of vibration is investigated. We validated the rubbing model in one side contact case with a rotor kit. By simulation, we reproduced the time history, the orbit and the full spectrum characteristics of the rotating shaft measured by the experiment precisely.

2 Rubbing Forces and Equations

This section describes the modeling of the rubbing force between the rotor and the casing. In Figure 1, schematic overviews of the rotor and the casing in rubbing are shown. The two contact configurations, one side and annular contact cases for translational motion are treated. The rubbing force consists of the radial contact force (blue arrow in Figure 1) and the tangential friction force (red arrow in Figure 1).

Contact stress theory is used for the contact force model. The contact force between the rotor and the casing is modeled as a piecewise linear spring and damper model. The friction force is modeled by multiplying the contact spring force by the friction coefficient. Although there is a model for using the coefficient of restitution for contact, since the contact time becomes infinitely small, it is not suitable for handling the frictional force which is calculated by multiplying the contact spring force by the friction coefficient at the time of contact.

The contact spring force F_s and the damping force F_d and the friction force F_f are written as follows.

$$F_\mathrm{s}(r) = \begin{cases} K(r - C_R) & \text{if } r > C_R \\ 0 & \text{if } r \le C_R \end{cases} \tag{1}$$

$$F_\mathrm{d}(v_r) = \begin{cases} dv_r & \text{if } r > C_R \\ 0 & \text{if } r \le C_R \end{cases} \tag{2}$$

$$F_\mathrm{f} = \mu(v_\phi)F_\mathrm{s}(r) \tag{3}$$

Here,
C_R: Clearance,
K: Contact spring constant,
d: Contact damping constant,
μ: Friction coefficient.
Thus the rubbing force in one side contact at the angle θ of the rectangular components Fx and Fy are written as follows.

$$\begin{aligned} F_x &= \left(F_\mathrm{s}(r) + F_d(v_r)\right)\cos\theta - \mu F_\mathrm{s}(r)\sin\theta \\ F_y &= \left(F_\mathrm{s}(r) + F_d(v_r)\right)\sin\theta + \mu F_\mathrm{s}(r)\cos\theta \end{aligned} \tag{4}$$

Here, (r, ϕ) is the relative rotor position against the casing in the polar coordinates and (v_r, v_ϕ) is the relative velocity.

$$r = \sqrt{(x_R - x_C + \delta_x)^2 + \left(y_R - y_C + \delta_y\right)^2}\cos(\phi - \theta) \tag{5}$$

$$\phi = \tan^{-1}\frac{y_R - y_C}{x_R - x_C} \tag{6}$$

$$v_r = \dot{r} \tag{7}$$

$$v_\phi = -(\dot{x_R} - \dot{x_C})\sin\theta + (\dot{y_R} - \dot{y_C})\cos\theta + R\omega \tag{8}$$

Here,
(δ_x, δ_y): The rotor offset against the casing,
$(x_R,\ y_R)$: Center of the rotor,
$(x_C,\ y_C)$: Center of the casing,
R: Rotor radius,
ω: Rotating speed.
In the annular contact case, $\phi = \theta$ holds in Equation 4, 5 and 8.

To estimate the contact spring constant, the Hertzian Contact Theory between two cylinders with parallel axes is used (Inagaki *et al*, 2005). The indentation depth $r - C_R$ is related to the contact force F_c as follows.

$$(r - C_R) = \frac{2F_\mathrm{c}(k_0 + k_1)}{\pi L}\left(1.8864 + \log\left(\frac{L}{2b}\right)\right) \tag{9}$$

$$k_i = \frac{1 - v_i^2}{E_i}(i = 0{,}1) \tag{10}$$

$$b = \sqrt{\frac{2F(k_0 + k_1)R}{\pi l}} \tag{11}$$

$$\frac{1}{R} = \frac{1}{R_0} + \frac{1}{R_1} \tag{12}$$

Here,
v_i: Poisson ratio of the cylinder,
E_i: Young's modulus of the cylinder,
R_i: Radius of the cylinder,
L: Length of the cylinder.
The contact spring constant is estimated by linearizing the contact force F_c against the indentation depth $r - C_R$.

The contact damping constant is estimated so that the loss for one contact is equivalent to that calculated from the coefficient of restitution. The coefficient of restitution e is defined by the following equation,

$$e = -\frac{v_1}{v_0} \tag{13}$$

Here, v_1 is the vertical velocity of the rotor immediately after contact with the casing surface, v_0 is the velocity immediately before contact.
Assuming that the rotor motion in contact follows the damped harmonic oscillator, the following relationship holds.

$$v_1 = v_0 exp\left(-\frac{\pi\zeta}{\sqrt{1-\zeta^2}}\right) \tag{14}$$

$$\zeta = \frac{d}{2\pi\sqrt{mK}} \tag{15}$$

Here,
ζ: Damping ratio,
m: Rotor mass.
From Equation 13, 14 and 15, the contact damping constant d is given by the function of the coefficient of restitution e as follows

$$d = \sqrt{\frac{2(\log e)^2 mK}{\pi^2 + (\log e)^2}} \tag{16}$$

It is possible to determine the coefficient of restitution by an experiment or the other more detailed analysis (Jackson *et al*, 2009).

3 Modelica Implementation

The presented rubbing force models must be supplied by constraints in the transverse direction x, y and rotating angle direction. Our Rotating Machinery library is used to supply these constraints (Ishibashi *et al*, 2017). The presented rubbing models are implemented in our rotating machinery library. The basic flange of this library has 5 DOF (degree of freedom), consisting of 4 DOF (two dimensional deflections and slopes) for transverse vibration of the rotor system and 1 DOF (rotating angle) for torsional vibration, neglecting axial vibration. Features like unbalanced rotors, flexible beams (shaft), supports, springs and dampers are all represented. The library is used to create the total rotating machinery system.

The rubbing force components in the one side contact case and the annular contact case are implemented respectively. The rubbing force components are implemented with two connectors, each with 5 DOF. Since the above rubbing force models has the only 3 DOF, the moments are set as zero. These connectors are the connections to the rotor and the casing.

The contact spring and damping force are implemented by the same method as that of `Modelica.Mechanics.Translational.Components.ElastoGap`. In order to calculate the friction force without further discontinuous events, the continuously differentiable friction model which decreases the simulation speed (Makkar *et al*, 2005) is used. The implementation is done by the same method as the Idealized Contact library (Oestersötebier *et al*, 2014). The following function of the relative velocity v_ϕ to approximate the friction coefficient of the characteristic Stribeck curve is implemented.

$$\mu(v_\phi) = \gamma_1\left(\tanh(\gamma_2 v_\phi) - \tanh(\gamma_3 v_\phi)\right) + \gamma_4 \tanh(\gamma_5 v_\phi) + \gamma_6 v_\phi \tag{17}$$

Here,
$\gamma_i (i = 1, 2, \cdots, 6)$: Non-physical constants.

In Figure 2 the icons of the rubbing force components in the one side contact case and the annular contact case are shown. No inertias or constraints are included in the model.

Using our Rotating Machinery library, it is possible to create rotating machinery systems. A simple rotating machinery system with casing is easily generated. Here, we treat a Jeffcott rotor system in partial rubbing with the casing as a test case in Figure 3. The model parameters are set to simulate the rotor kit shown in Figure 14. The lowest eigen frequency ε_L of the shaft bending mode in the model is 30 Hz. The casing mass, stiffness and damping are the same as the rub screw in Figure 14. Also, the contact spring and damping constant and friction coefficient in the rubbing component are the same as the rub screw shown in Figure 14. Only the casing position of the model in the direction of rotating shaft axis is different from the model shown in Figure 18.

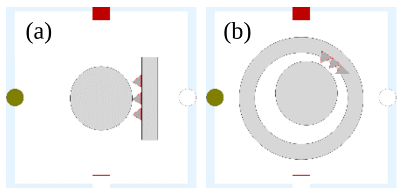


Figure 2. Modelica Icons for rubbing components. (a) One side contact case. (b) Annular contact case.

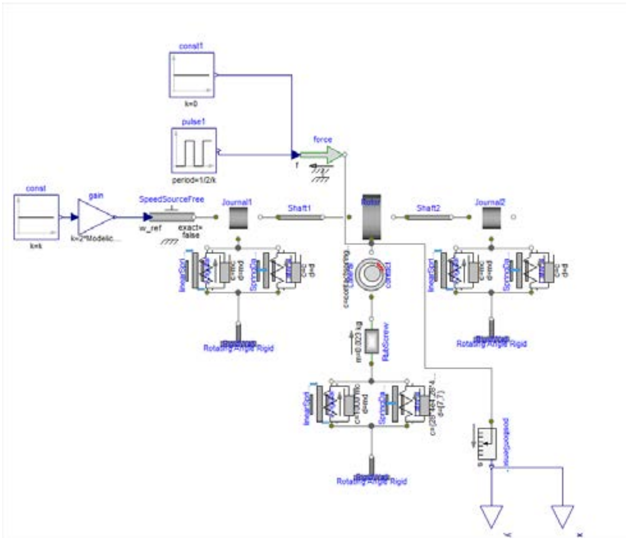


Figure 3. Modelica model of a Jeffcott rotor system having partial rubbing with the casing.

In this model (Figure 3), the whole rotor system including the casing position is symmetrical. In the model, the rubbing component (Figure 2) is defined as described in this paper, all other components are from our rotating machinery library.

4 Simulation Results

In this section, using the model (Figure 3) and rubbing components (Figure 2), the generation of various kinds of vibration at high speed rotating speed due to partial rubbing is investigated by simulations.

4.1 Rubbing vibration

Dymola is used for the simulations. Since the model contains many events, single-step solver "Radau" is used for simulation.

Simulation is done at the constant rotating speed ω over 1.5 times the speed of the critical speed ε_L. The simulation that the rotating shaft is whirling with smaller amplitude than the clearance is done. By applying a pulsating external force, the partial rubbing vibration is induced (Figure 4). A pulsating external force is applied when the gap between the rotor and the casing become the smallest.

As a result, although the rotor whirling of unbalance is smaller than the clearance, the rotor keeps in contact with the case after the contact due to a pulsating external force under some conditions despite the same pulsating external force amplitude (Figure 4). However, the vibration converges and returns to a non-contact state under another conditions (Figure 5). To investigate this kind of vibration, batch simulation of sweeping rotating speed and the eccentricity of static unbalance in Rotor is done by python interface.

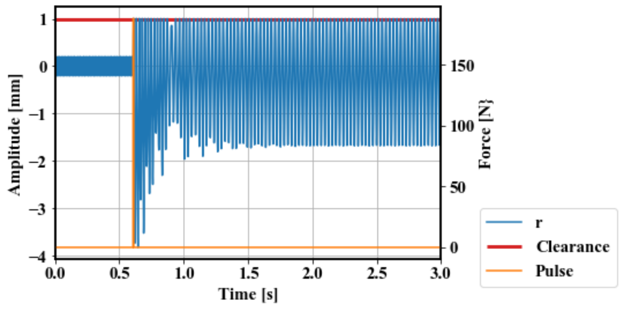


Figure 4. Rubbing vibration occurs and continues.

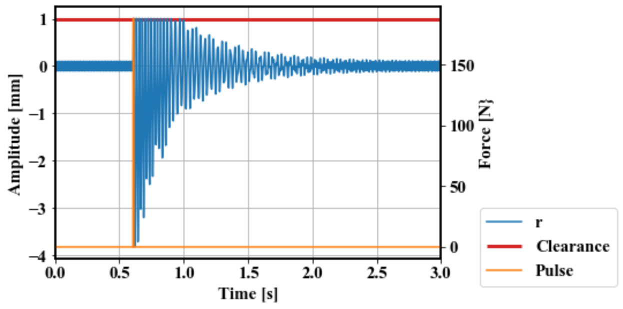


Figure 5. Rubbing vibration does not occur.

4.2 One side contact case

Figure 6 shows the domain of the rubbing vibration occurrence in the one side contact case. The model (Figure 3) replacing the rubbing component in the annular contact case with that in the one side contact case is simulated. The contact angle $\theta = \pi/2$ and the clearance $C_R = 1$ mm are set in the rubbing component. The rubbing (i.e. contact) vibration occurs and continues in the region with a plot in Figure 6. The vibration converges immediately after the contact, and returns to a non-contact state in the region without a plot in Figure 6. Figure 7 shows the rubbing vibration behavior in the one side contact case. From the left figure, the Rotor displacement in Y direction, orbit and full spectrum are shown. The full spectrum is obtained from the half spectrums of each X and Y displacement by the procedure written in the paper (Goldman and Muszynska, 1999). The unbalance amplitude before contact in Figure 7 is set as around 0.1 against the clearance. Due to the translation mode of the rotating shaft, Rotor unbalance amplitude is larger in the low rotating speed range.

In the high rotating speed range over the first critical speed ε_L, the 1/n (n: integer) sub harmonic rubbing vibration continues in the region shown in Figure 6. These vibration occurs in higher speed of n integer multiple of the eigen frequency ε_L. As the unbalance increases, the 1/n sub harmonic vibration occurs in higher rotating speed. As the integer n becomes larger, the 1/n sub harmonic rubbing vibration occurs from smaller unbalance region. The domain of the rubbing vibration occurrence shows a gap. In the gap where rubbing vibration is unlikely to occur, the casing is more than clearance away and moving away from the rotor.

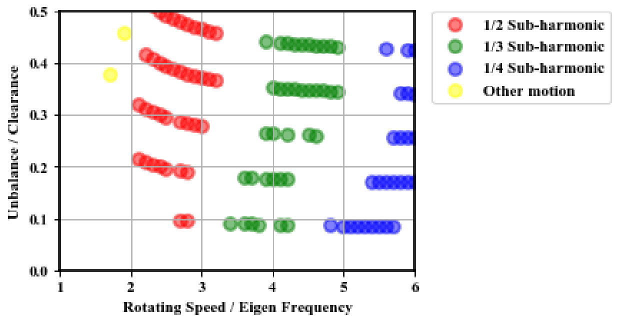
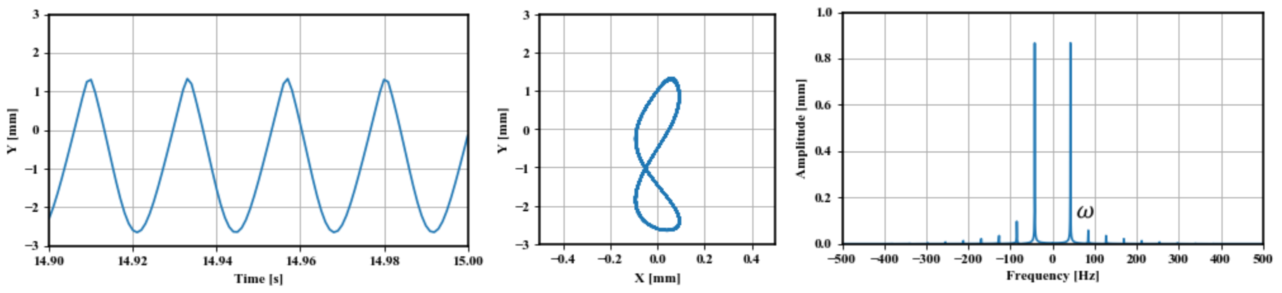
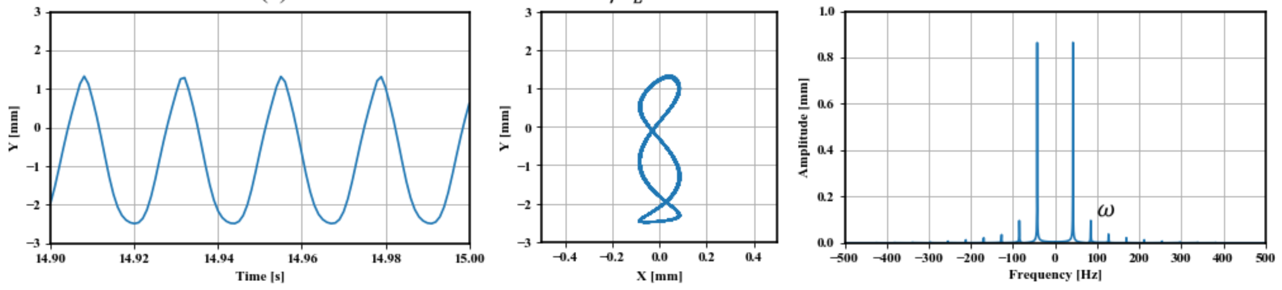


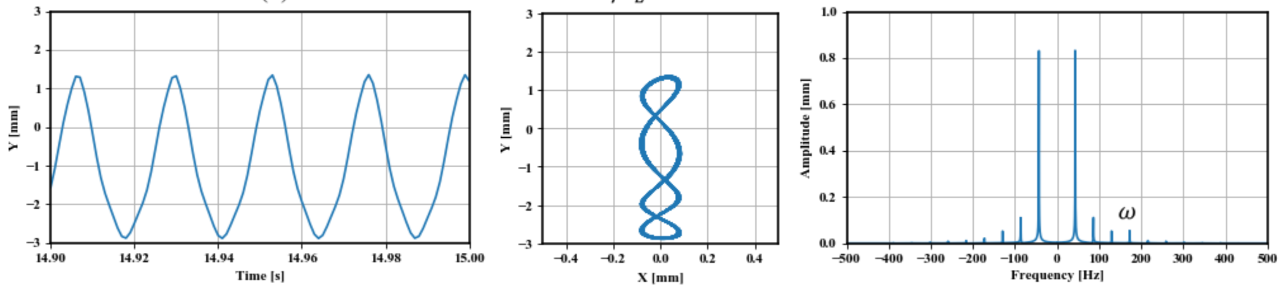
Figure 6. The domain of the rubbing vibration occurrence in the one side contact case.



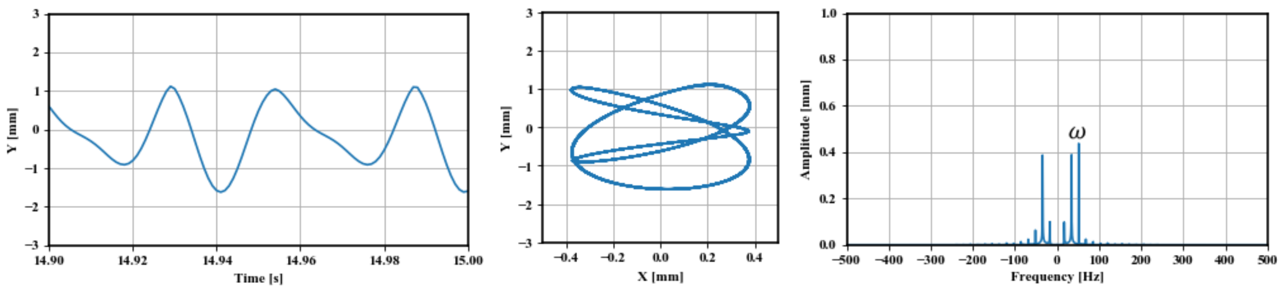
(a) 1/2 Sub-harmonic Vibration $\omega/\varepsilon_L = 2.8$



(b) 1/3 Sub-harmonic Vibration $\omega/\varepsilon_L = 4.2$



(c) 1/4 Sub-harmonic Vibration $\omega/\varepsilon_L = 5.7$



(d) Other motion $\omega/\varepsilon_L = 1.7$, Unbalance/Clearance $\approx$ 0.4

Figure 7. Vibration behavior in the one side contact case.

4.3 Both side contact case

Figure 9 shows the domain of the rubbing vibration occurrence in the both side contact case. Figure 8 shows the rubbing vibration behavior. The model adding the one side rubbing component and the casing on the other side is simulated. The two casings have the same mass, spring and damping constant as each other. Also, the two one side rubbing components have the same parameters values such as the contact spring and damping constant and friction coefficient as each other. The contact angle $\theta = -\pi/2$ and the clearance $C_R = 1$ mm are set in the other rubbing component.

In this case, the synchronous with rotating speed and 1/n sub harmonic vibration occurs. The sub harmonic rubbing vibration just occurs in the region of moderately small unbalance. As the unbalance gets larger, synchronous vibration occurs in the region of wide rotating speed range. This violent vibration is regarded as a kind of collision type self-excited vibration. In the both side contact case, only the odd number sub harmonic vibration occurs. Since the rotor orbit of the even number sub harmonic vibration is basically asymmetrical with respect to the origin, there is no solution where there are two casings located on both sides of the rotor under the same condition (see Figure 7 and Figure 8). However, if there is a difference between the two casings, the rotor sometimes contacts with the casings only at one side.

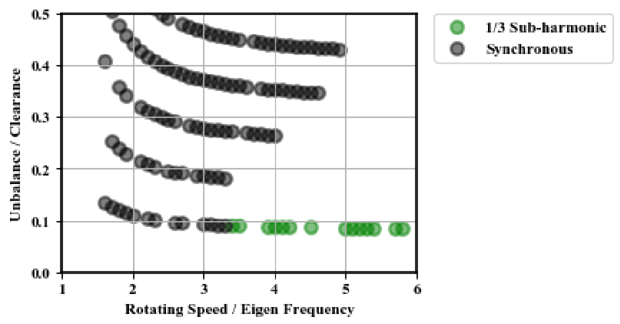
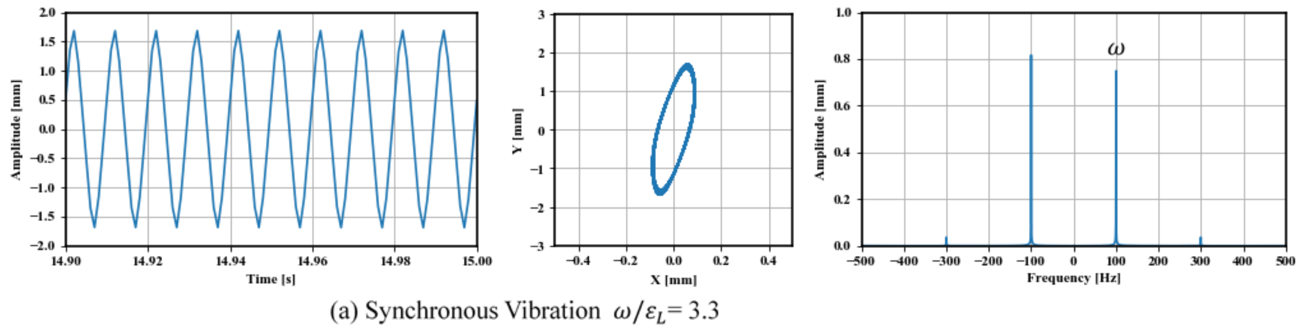
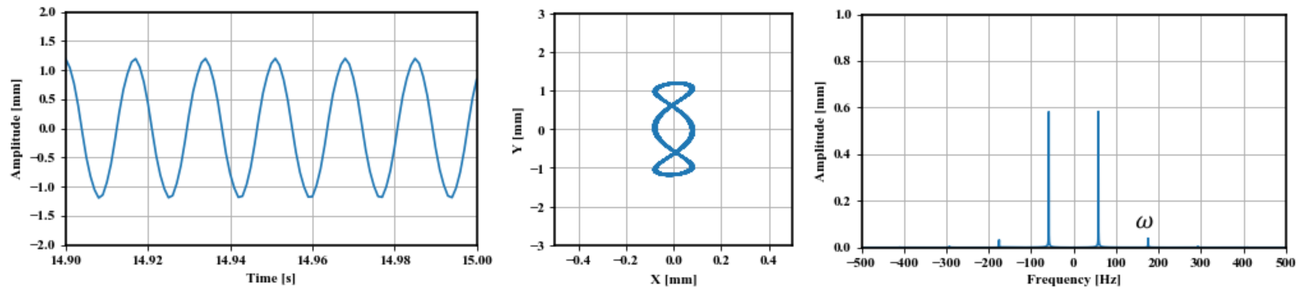


Figure 9. The domain of the rubbing vibration occurrence in the both side contact case.



(a) Synchronous Vibration $\omega/\varepsilon_L = 3.3$



(b) 1/3 Sub-harmonic Vibration $\omega/\varepsilon_L = 5.8$

Figure 8. Vibration behavior in the both side contact case.

4.4 Annular contact case

Figure 11 shows the domain of the rubbing vibration occurrence in the annular contact without offset. The model (Figure 3) is simulated. Figure 10 shows the rubbing vibration behavior. The contact spring constant in the rubbing component is calculated by considering the difference in the curvature. In this case, two kinds of collision type self-excited synchronous vibration with rotating speed occurs. One is the circle rotor orbit and the other is the oval rotor orbit shown in Figure 10. In the low rotating speed range, synchronous circle rotor orbit vibration occurs. In the high rotating speed range, synchronous oval rotor orbit vibration occurs from the small unbalance region. From the rotor orbit in Figure 10, the rotor collides with the casing several times per one whirling period.

Figure 12 shows the domain of contact vibration in the annular contact case with offset. Figure 13 shows the rubbing vibration behavior. The model (Figure 3) with the rotor offset (δ_x, δ_y)=(0, $C_R/4$) against the casing in the rubbing component is simulated. In comparison with the domain without the offset, the synchronous circle rotor orbit vibration occurs in the wider rotating speed range toward high rotating speed. In addition to the synchronous vibration in the annular contact case without the offset, the sub harmonic and the other kind of the synchronous circle rotor orbit vibration occur. The sub harmonic vibration occurs in the slightly higher rotating speed of the n integer multiple of the eigen frequency ε_L.

Although there was a small difference between our simulation results and the previous papers (Watanabe *et al*, 2004; Watanabe *et al*, 2005) due to the differences of the parameters amplitudes in the rubbing component, the relationship between the contact configurations and the generation of various kinds of vibration obtained in this paper showed the same trend as those papers.

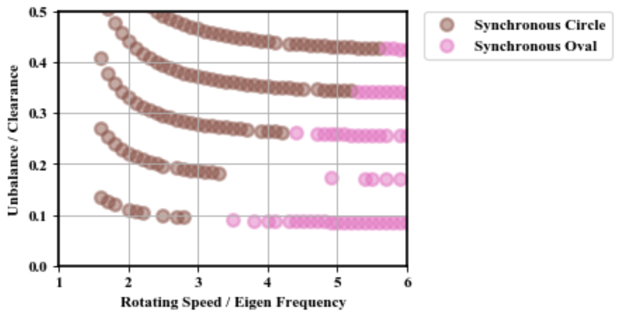


Figure 11. The domain of the rubbing vibration occurrence in the annular contact case without the offset.

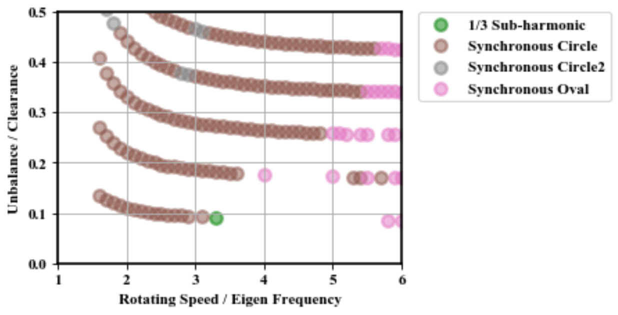
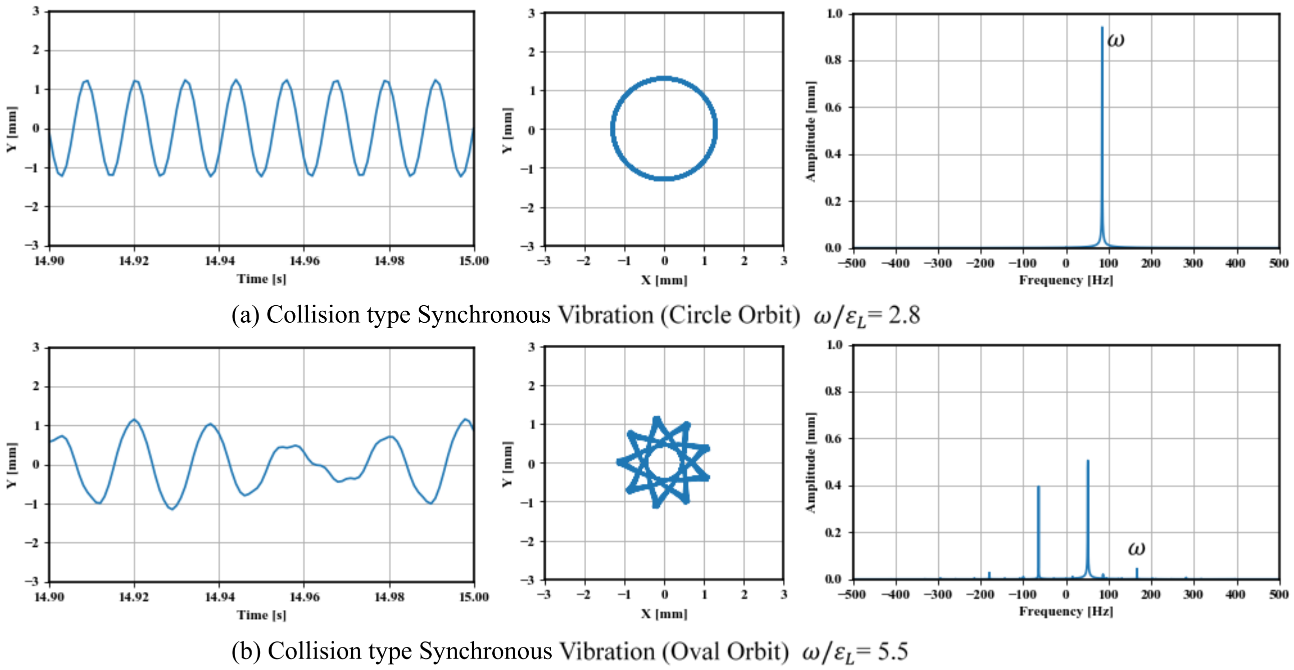


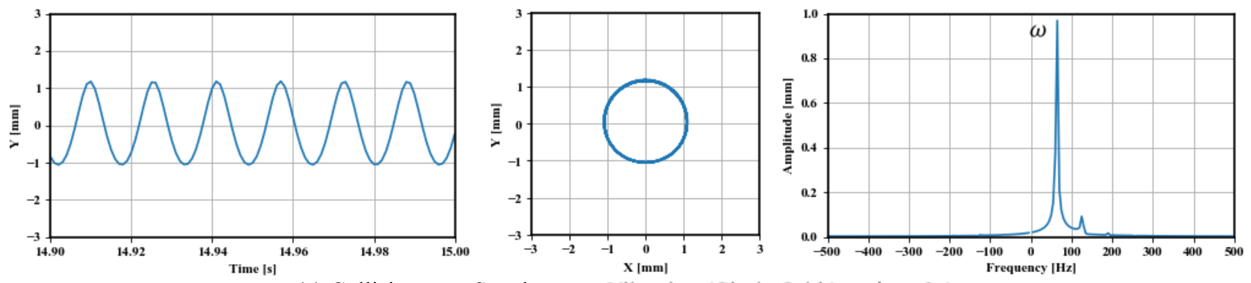
Figure 12. The domain of the rubbing vibration occurrence in the annular contact case with the offset.



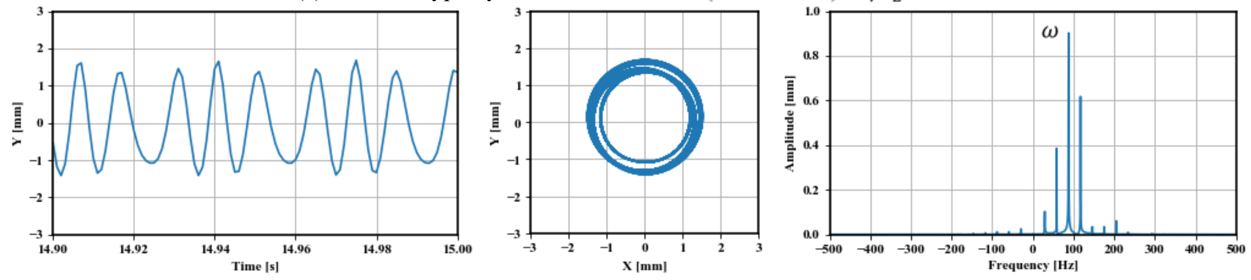
(a) Collision type Synchronous Vibration (Circle Orbit) ω/ε_L= 2.8

(b) Collision type Synchronous Vibration (Oval Orbit) ω/ε_L= 5.5

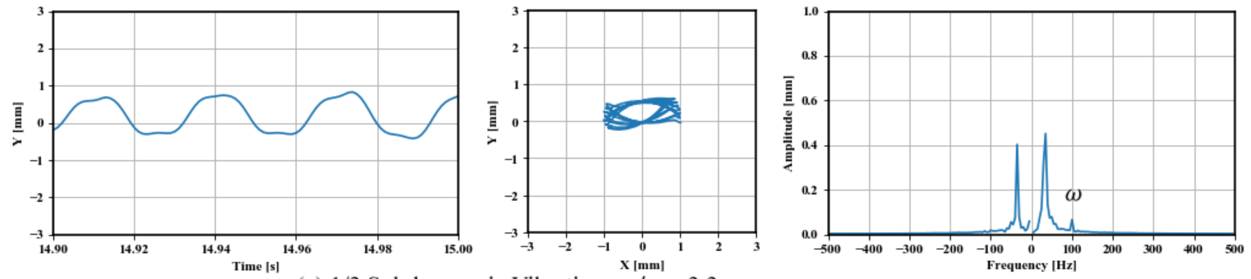
Figure 10. Vibration behavior in the annular contact case without the offset.



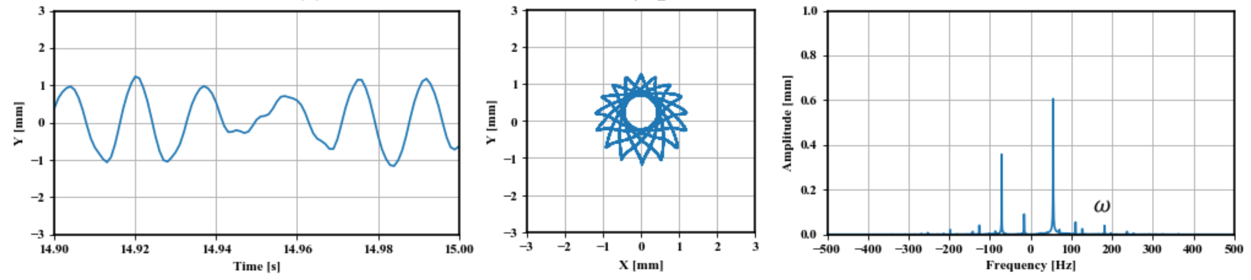
(a) Collision type Synchronous Vibration (Circle Orbit) $\omega/\varepsilon_L = 2.1$



(b) Collision type Synchronous Vibration (Circle Orbit 2) $\omega/\varepsilon_L = 2.9$, Unbalance/Clearance $\approx$ 0.4



(c) 1/3 Sub-harmonic Vibration $\omega/\varepsilon_L = 3.3$



(d) Collision type Synchronous Vibration (Oval Orbit) $\omega/\varepsilon_L = 6.0$

Figure 13. Vibration behavior in the annular contact case with the offset.

5 Model Validation

5.1 Experiment

To validate our models, we used the rotor kit (RK 4 Rotor Kit GE Bently Nevada) shown in Figure 14. The rotating shaft was supported by the solid lubricated bearings covered by the rubber. The rotating shaft was measured by two proximitors arranged in orthogonal directions. The data sampling time of the proximitors for recording was 1 ms.

To make it easier for analyzing the experimental result, the Jeffcott rotor system was investigated for the experiment. One side contact condition was established by adjusting the rub screw made of brass and applying a pulsating external force to the rotating shaft.

From the preliminary impulse and rotating speed ramp response experiment, the first critical speed ε_L (the lowest eigen frequency of the shaft bending mode) of this rotating shaft system was around 30 rps. By rotating at 70 rps (over twice the first critical speed) and applying a pulsating external force, the rubbing condition was established.

Figure 15 shows the time history of the rotating shaft in the both horizontal and vertical direction under rubbing condition. Figure 16 shows the orbit of the rotating shaft. Figure 17 shows the full spectrum analysis of the rotating shaft time history. The procedure for obtaining the full spectrum from the half spectrums of each proximity probe was followed by the paper (Goldman and Muszynska, 1999). The 1/2 sub harmonic rubbing vibration was observed.

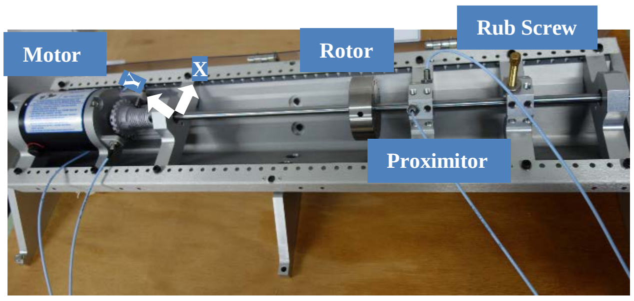


Figure 14. The rotor kit.

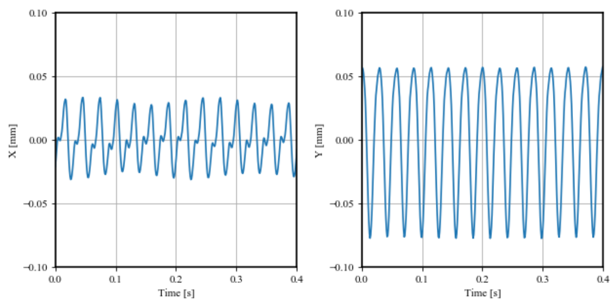


Figure 15. The time history of the experiment under rubbing condition at the rotating speed of 70 rps.

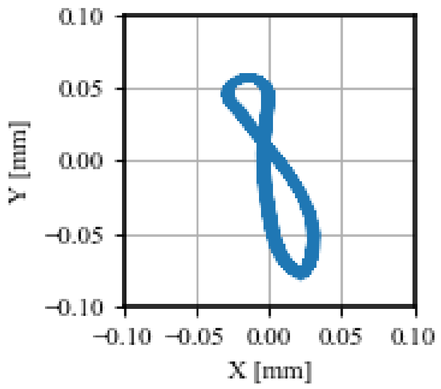


Figure 16. The orbit of the experiment under rubbing condition at the rotating speed of 70 rps.

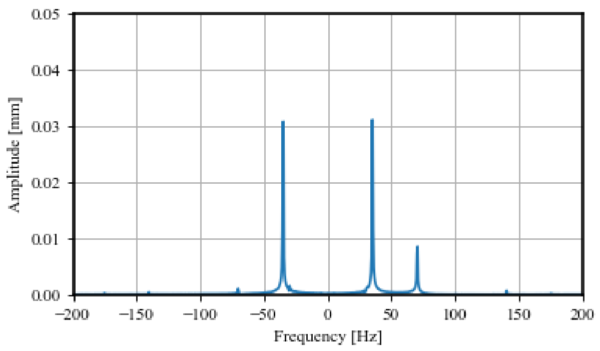


Figure 17. The full spectrum analysis of the experiment under rubbing condition at the rotating speed of 70 rps.

5.2 Simulation

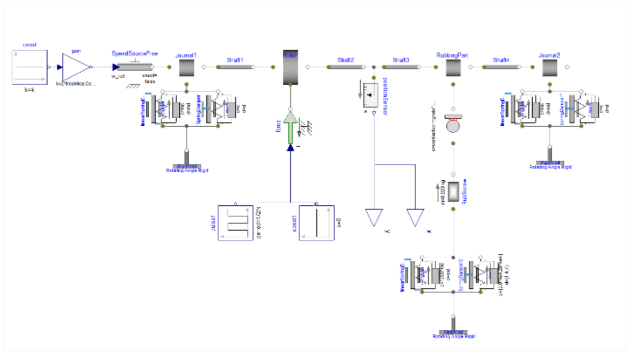


Figure 18. Modelica model for the rotor kit.

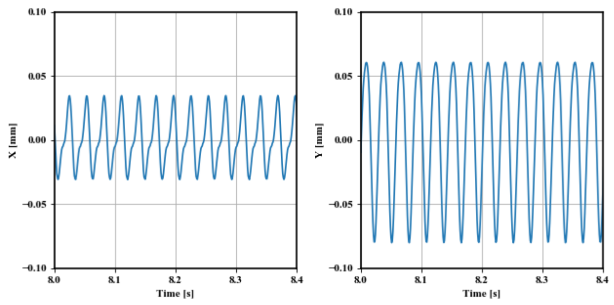


Figure 19. The time history of the simulation under rubbing condition at the rotating speed of 70 rps.

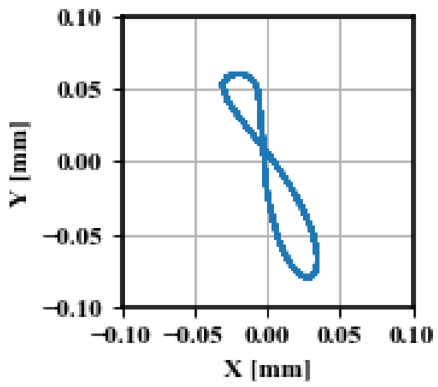


Figure 20. The orbit of the simulation under rubbing condition at the rotating speed of 70 rps.

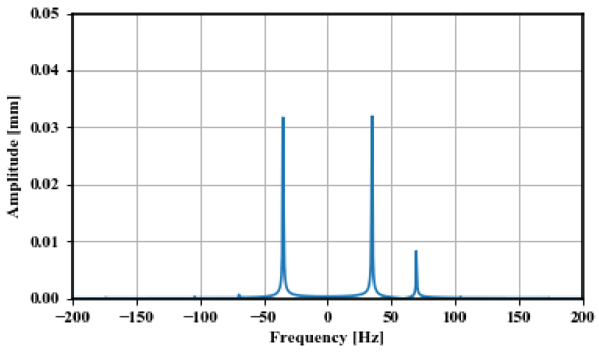


Figure 21. The full spectrum analysis of the simulation under rubbing condition at the rotating speed of 70 rps.

To carry out the analytical investigation of the experimental result, we built the Modelica model corresponding to the rotor kit based on our library shown in Figure 18. The parameters such as the bearing stiffness and damping, the residual bow (the bend of shaft) and the rotor static unbalance of the model were calibrated to match the preliminary experiments by the same method of the previous paper (Ishibashi *et al*, 2017). The rub screw stiffness and damping were estimated by the preliminary impulse test. The contact spring constant in the rubbing component were estimated by Equation 9. The contact damping constant were estimated by Equation 16 using the approximate value of the coefficient of the restitution (Jackson *et al*, 2009). The friction coefficient was estimated from the value reported in the paper (Watanabe *et al*, 2004).

Figure 19 - Figure 21 show the results under rubbing condition. By simulation, we reproduced the time history, the orbit and the full spectrum characteristics of the rotating shaft measured by the experiment precisely.

6 Conclusions

In this paper, the rubbing component models in the one side contact case and the annular contact case are presented to describe the partial rub. Using our rotating machinery library, it is possible to model a rotating machinery system with partial rubbing.

Examples of a Jeffcott rotor system with the different contact configuration were investigated by simulation. The relationship between the contact configurations and the generation of various kinds of vibration such as "collision type synchronous vibration", "sub harmonic vibration", etc. obtained by simulation showed the same trend as the previous papers (Watanabe *et al*, 2004; Watanabe *et al*, 2005).

We validated our model with partial rubbing in the one side contact case with a rotor kit. By simulation, we reproduced the time history, the orbit and the full spectrum characteristics of the rotating shaft measured by the experiment precisely.

The presented models make it possible to carry out analytical investigations of the partial rub in order to gain some insight into the diagnostics of rotating machinery.

References

R. F. Beatty. Differentiating rotor response due to radial rubbing. *Journal of Vibration and Acoustics*. 107(2): 151-160, 1985. doi:10.1115/1.3269238.

Yeon-Sun Choi and Sherif T. Noah. Nonlinear Steady-State Response of a Rotor-Support System. *Journal of Vibration, Acoustics, Stress and Reliability in Design*. 109 (4), 225-261, 1987. doi:10.1115/1.3269429.

Markus Dahl, Håkan Wettergren and Henrik Tidefelt. Modelica Spur Gears with Hertzian Contact Forces. *Proceedings of the 12th International Modelica Conference*, 2017. doi:10.3384/ecp17132755.

Fredric F. Ehrich. Subharmonic vibrations of rotors in bearing clearance, ASME Paper 66-MS-1, 1966.

Paul Goldman and Agnes Muszynska. Application of full spectrum to rotating machinery diagnostics. *Orbit First Quarter*, pp. 17-21, 1999.

Andreas Hofmann, Lars Mikelsons, Ines Gubsch, and Christian Schubert. Simulating Collisions within the Modelica MultiBody Library. *Proceedings of the 10th International Modelica Conference*, 2014. doi: 10.3384/ecp14096949.

Mizuho Inagaki, Yukio Ishida and Akimasa Hayashi. Nonlinear Resonances and Self-Excited Oscillations of a Rotor due to Radial Clearance and Impact in Bearings. *Trans. JSME*, C 71, pp. 2113-2118, 2005 *(in Japanese)*. doi: 10.1299/kikaic.71.2113.

Tatsuro Ishibashi, Han Bing and Tadao Kawai. Rotating Machinery Library for Diagnosis. *Proceedings of the 12th International Modelica Conference*, 2017. doi:10.3384/ecp17132381.

Robert L. Jackson, Itzhak Green and Dan B. Marghitu. Predicting the coefficient of restitution of impacting elastic-perfectly plastic spheres. *Nonlinear Dyn*, 60: 217-229, 2010. doi: 10.1007/s11071-009-9591-z.

Ivan Kosenko and Il'ya Gusev. Implementation of the spur involute gear model on modelica. *Proceedings of the 8th International Modelica Conference*, 2011. URL https://www.modelica.org/events/modelica2011/Proceedings/pages/papers/13_3_ID_117_a_fv.pdf.

Ivan Kosenko and Ilya Gusev. Revised and improved implementation of the spur involute gear dynamical model. *Proceedings of the 9th International Modelica Conference*, 2012. doi:10.3384/ecp12076311.

C. Makkar, W. E. Dixon, W. G. Sawyer, and G. Hu. A New Continuously Differentiable Friction Model for Control Systems Design. *Proceedings of the 2005 IEEE/ASME International Conference on Advanced IntelligentMechatronics*, Monterey CA, July, 2005.

Felix Oestersötebier, Peng Wang and Ansgar Trächtler. A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces. *Proceedings of the 10th International Modelica Conference*, 2014. doi: 10.3384/ecp14096929.

Martin Otter, Hilding Elmqvist, and José Díaz López. Collision Handling for the Modelica MultiBody Library. *Proceedings of the 4th International Modelica Conference*, 2005. URL http://elib.dlr.de/12299/1/otter2005-modelica-collision.pdf

F.L.J. van der Linden. Modelling of elastic gearboxes using a generalized gear contact model. *Proceedings of the 9th International Modelica Conference*, 2012. doi:10.3384/ecp12076303.

Yusuke Watanabe, Takuzo Iwatsubo and Keizo Awa. Study of Rotor Vibration due to the Rubbing against Casing. *Trans. JSME*, C 70, pp. 2181–2187, 2004 *(in Japanese)*. doi: 10.1299/kikaic.70.2181.

Yusuke Watanabe, Takuzo Iwatsubo and Keizo Awa. Study of Rotor Vibration due to the Rubbing against Casing. *Trans. JSME*, C 71, pp. 1421–1428, 2005 *(in Japanese)*. doi: 10.1299/kikaic.71.1421.

Aspects of Train Systems Simulation

Martin R. Kuhn[1] Yang Ji[2] Bo Wang[2] Xiang Li[2] Bohui Liu[2] Feng Sha[2] Dunwen Gan[3] Feng Gao[3]
[1]Yuanda SimTek GmbH, Germany, martin.kuhn@yuandasimtek.de
[2]SimTek CO, China, {yang.ji ,bo.wang, li.xiang, bohui.liu, feng.sha} @cnydsimtek.com
[3]BEIJING ZONGHENG ELECTRO-MECHANICAL TECHNOLOGY DEVELOPMENT CO, {gandunwen,gaofeng} @zemt.cn

Abstract

This paper present needs and implementations for system modeling of high speed trains with focus on the Beijing-Zhangjiakou Intercity Railway. Different scenarios are proposed which are relevant in systems design. The implementation with Modelica is discussed and demonstrated for the rail-wheel contact and mechanical, logical, electrical and thermal systems.

Keywords: High speed train, systems, electrical, mechanical, thermal, rail-wheel contact, single phase, traction

1 Motivation

1.1 The Chinese high speed trains system

Chinese Railway High-speed (CRH) was first introduced in 2007 and was developed further with operating speeds of 250-300 km/h. The train system was able to significantly reduce the further growth of air traffic. Even for long distances as Beijing-Shanghai with 1077 km the travelers are attracted by scheduled travel times of 288 minutes in relation to 135 minutes by airplane. Assuming 60 minutes additional journey time for the train respectively 150 minutes to the outlying airports the ratio shrinks to 348 minutes to 285. In addition the train service is operated roughly every 5 minutes with prices around 85% of the air ticket.

The Chinese high speed train program follows the principle of Electric Multiple Units (EMU), where the train consists of self-propelled carriages with no separate locomotive and traction motors incorporated within a number of the carriages. The same concept applies for example for the French AGV, Italian Pendolino, Japanese Shinkansen or German ICE 3. Beijing-Zhangjiakou Intercity railway, as a supporting project of the Beijing 2022 Olympic and Paralympic Winter Games, is the first high-speed train with an operational speed of 350 km/h and automatic driving. It will operate in an alpine and windy region where it will reduce the commuting time between Beijing and Zhangjiakou to an hour. Assembly of the EMUs is planned by end of 2018 and the adaption and test verification shall be completed in the first half of 2019. ZEMT is one of the core partners in the new development. The company identified a further need for model-based system design, optimization, fault analysis and diagnosis. Train systems simulation is a well known application of Modelica (Belmon and Liu, 2011; Dumont and Maurer, 2012; Franke and Wiesmann, 2014; Heckmann et al., 2014; Frilli et al., 2016). Therefore, SimTek has been commissioned to develop a Modelica-based train systems library for further virtual testing and verification of the Beijing-Zhangjiakou high-speed train. This paper presents typical scenarios which are relevant in systems design. The implementation of the library with Modelica is discussed and demonstrated for the rail-wheel contact and mechanical, logical, electrical and thermal systems, with special attention to the rail-wheel contact and electrical power off-take. The models are partly generic and do not necessarily reflect the real system layout.

1.2 Applications for simulation

The following list provides typical applications for simulation based studies:

- Vehicle energy consumption estimation for systems and supply network optimization
  - Estimation of the energy consumption of all subsystems
  - Energy consumption of air-conditioning and auxiliary system
  - Energy consumption in different climate and weather environments and roads
- Electric grid harmonic estimation for topology and filter selection
  - Estimate the harmonics according to industrial standards
- Traction system thermal capacity estimation for cooling system layout and control of power reduction
  - Cooling models
  - Simplification and parameter estimation of fluid models involving heat exchange
  - Simplification and parameter estimation of models of the detailed electric power switching with thermal effect

    - Simplification and parameter estimation of models of the coupled power electrics and dynamic thermal model
- Driven cars stability estimation
    - Lateral dynamics analysis
    - Vertical dynamics analysis

2 Modeling

2.1 Overview on libraries

The modeling for the ZEMT Project of the virtual EMU is structured into four libraries, which will be commercialized. Their layout is shown in Figure 1.

This paper is focused on the unique feature of train modeling with the introduction of the mechanical modeling and thermalfluid modeling in the TrainDynamics Library, and the control logics modeling and electrical traction modeling in TractionSystemLibrary.

The TrainDynamicsLibrary supports the following models：

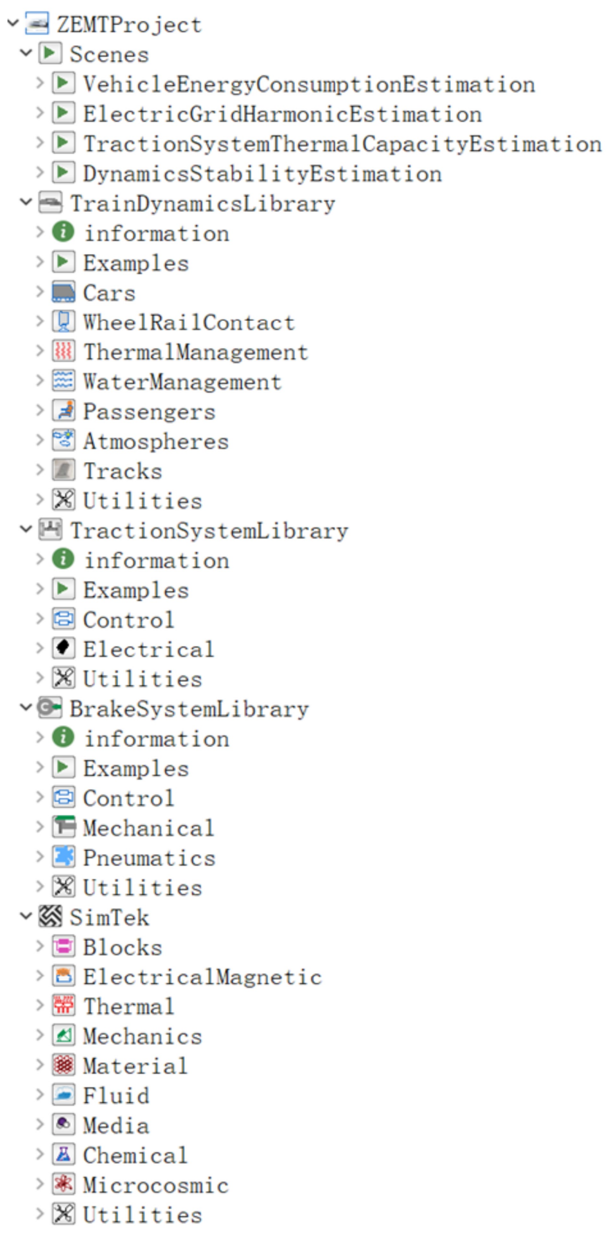
- Dynamic multi-body models of coach body and bogie
- Dynamic wheel/rail contact model.
- Thermal management, simple air conditioning and cooling unit of traction system, including motor cooling unit, converter cooling unit and transformer cooling unit.
- Water management, i.e., water supply and drainage system
- Passenger model, including weight, temperature, O_2 consumption, water consumption
- Environment model, including temperature, pressure, humidity and wind resistance
- Path model, including 2 kinds of x-y-z coordinates and x-z-R（bend road）.

The usage and contents of the TrainDynamicsLibrary:

- Analyze the stability of the train during hunting movement,
- Analyze the stability of the train while passing the winding railway line,
- Estimation and analysis of the energy consumption of the train.

ElectricTractionLibrary provide the dynamic models of electric traction system and control system. Subsystems include:

- Communication models,
- Signal system models,
- networks models,
- System integration and model in loop.



```
ZEMTProject
  Scenes
    VehicleEnergyConsumptionEstimation
    ElectricGridHarmonicEstimation
    TractionSystemThermalCapacityEstimation
    DynamicsStabilityEstimation
  TrainDynamicsLibrary
    information
    Examples
    Cars
    WheelRailContact
    ThermalManagement
    WaterManagement
    Passengers
    Atmospheres
    Tracks
    Utilities
  TractionSystemLibrary
    information
    Examples
    Control
    Electrical
    Utilities
  BrakeSystemLibrary
    information
    Examples
    Control
    Mechanical
    Pneumatics
    Utilities
  SimTek
    Blocks
    ElectricalMagnetic
    Thermal
    Mechanics
    Material
    Fluid
    Media
    Chemical
    Microcosmic
    Utilities
```

Figure 1. ZEMT project and SimTek commercial libraries.

2.2 Train Dynamics Modeling

2.2.1 Implementation

The mechanical structure of the EMU can be modeled by the simplified five elements model whose topology is shown in Figure 2. It includes:

- Coach body
- Bogie
- Wheel-sets
- The secondary suspension
    - air spring model
    - anti-roll var model
    - vertical damper
    - lateral damper
    - yaw damper
- The primary suspension

- spring damper.

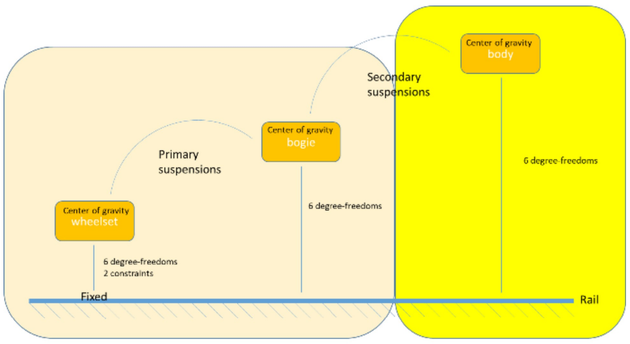


Figure 2. Topology structure about mechanical model.

All the design parameters and the characteristic curve of the suspension are based on identified test data. For example, the lateral damper characteristic is shown in Figure 3.

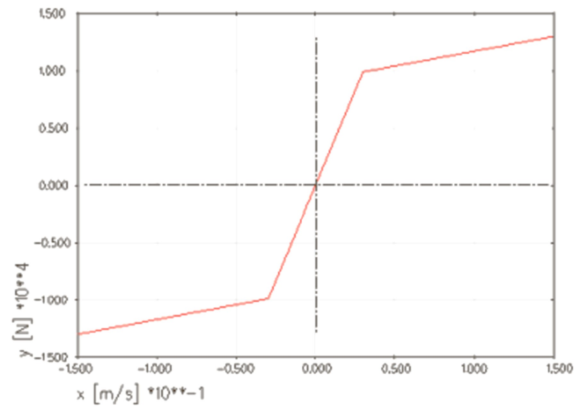


Figure 3. Lateral damper characteristic map.

The contact model is the most difficult part of the vehicle dynamics analysis. For automobiles, it's the adhesion forces to ensure the continuous motion. For aircraft, it's the air forces due to the contact between wing and air to guarantee the flight. The forces along motion direction and the perpendicular direction should be considered in the contact model. In a small range of the contact angle, the friction forces F in lateral direction is linear to the contact angle α, and its

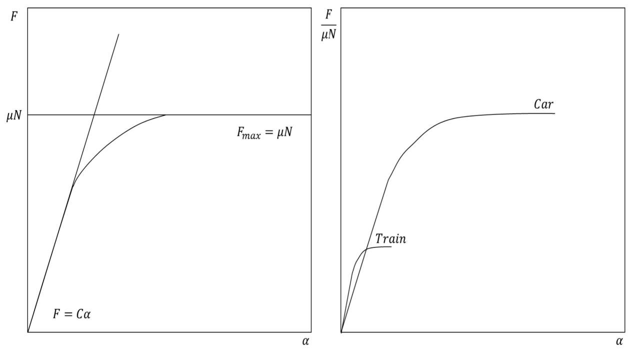


Figure 4. Left: Friction force in relation to contact angle. Right: Difference between trains and automobiles contact model.

maximum is linear to the vertical load (Masao, 2008). This is shown in the left of Figure 4.

As for the trains, the linearity range of the contact angle is smaller and the curvature is more complex than for cars, as shown in the right of Figure 4.

For the EMU study, the integrated train energy consumption needs to be analyzed and controlled in real-time, where the simulation speed is more critical than the accuracy. In (Heckmann et al., 2014) an algorithm for fast calculation of wheel-rail forces with Modelica was proposed. Here, an alternative very fast approach is demonstrated. The local coordinates of the model are depicted in Figure 5 (Masao, 2008). In the six degrees of freedom of the train wheel set, the motion along z axis and rotation around x axis are constrained by the rail. Only 2D planar motion is allowed, which is the forward/backward movement in x direction, left/right movement in y direction, rotation around z direction (ψ) and 1D rotation around y axis (θ). Considering the suspension of the train, the forces applied on the wheels by the train body are assumed to be one-dimensional in direction of z axis. This means a combination of 2D and 1D model was built, in use of the PlanarMechanics library (Zimmer, 2012).

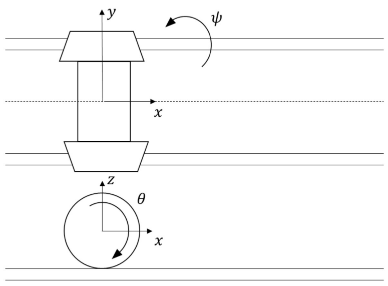


Figure 5. The moving coordinate system of the contact model.

The transformation between the local (x, y, ψ) and global reference system (X, Y, ψ) is performed by equations (1) and (2).

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \ddot{x} - \dot{y}\,\dot{\psi} \\ \ddot{y} + \dot{x}\,\dot{\psi} \\ \ddot{\psi} \end{bmatrix} \tag{2}$$

From the view of contact geometry, the wheel-rail shape is simplified to a cone to avoid massive nonlinear computation in which case both the slip forces and side forces are still considered in the model, as shown in Figure 6 (Masao, 2008).

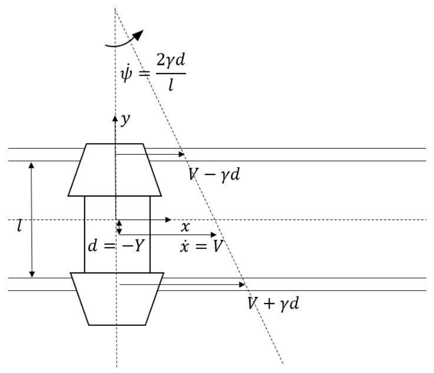


Figure 6. Wheels model.

The velocity in the moving coordinate system is given by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} V \\ 0 \\ \frac{2\gamma d}{l} \end{bmatrix} \tag{3}$$

Where γ is the angle of the cone wheel, d is the distance between the center of the wheel and the center of the rail ($d = -Y$), l is the distance between wheels and V is the speed in x-direction in local coordinates. When considering only the kinematics, as shown in equation (2) and (3):

$$\ddot{Y} = V\cos\psi\,\dot{\psi} \approx V\dot{\psi} = \frac{-2V\gamma}{l}Y \tag{4}$$

This indicates that the assumption is able to model the hunting behavior of the lateral vibration of the wheel-rail with the frequency of $\sqrt{2V\gamma/l}$.

Considering the adhesion and curve influence in the running direction, according to (Masao, 2008) the equation of the train dynamics in the moving coordinate system can be unified by equation (5):

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\psi} \end{bmatrix} + \frac{1}{V} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2c_{22} & 0 \\ 0 & 0 & \frac{1}{2}c_{11}l^2 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -2c_{22} \\ 0 & 2c_{11}\frac{l\gamma}{d} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \psi \end{bmatrix} = \begin{bmatrix} F_x - f \\ F_y + m\left(\frac{V^2}{R} - \frac{gh}{l}\right) \\ M_z \end{bmatrix} \tag{5}$$

Where c_{11} and c_{22} are the constants of Kaller contact theory, h is the height difference of outer side and inner side of the curves. m is the equivalent mass obtained through the gravity calculation in the interface. F_x, F_y, M_z are the friction forces and torques due to the adhesion which can be calculated by the slip rate. Assuming the slip rate is s and its component in x and y direction is s_x, s_y, with the self rotation ψ:

$$\begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix} = f \cdot \begin{bmatrix} s_x \\ s_y \\ \psi \end{bmatrix} \tag{6}$$

With the radius of the wheel R and inertia I, the relationship between slip rate s and the traction/braking forces is given by equation (7) and (8).

$$s = \begin{cases} \frac{R\dot{\theta} - V}{R\dot{\theta}} \ \forall \ \text{Traction} \\ \frac{R\dot{\theta} - V}{V} \ \forall \ \text{Brake} \end{cases} \tag{7}$$

$$RI\ddot{\theta} = F_{traction} + F_{brake} \tag{8}$$

The relationship between vehicle dynamics of the train and traction/braking forces can be obtained based on the equation (6). For even faster calculation, the library implements the simplification of (Polach, 2000) of equation (6). An approach assuming an ellipsoidal contact area is used, with semi-axes a, b and normal stress distribution according to Hertz to calculate the rail-wheel forces, as shown in Figure 7 (Polach, 2000).

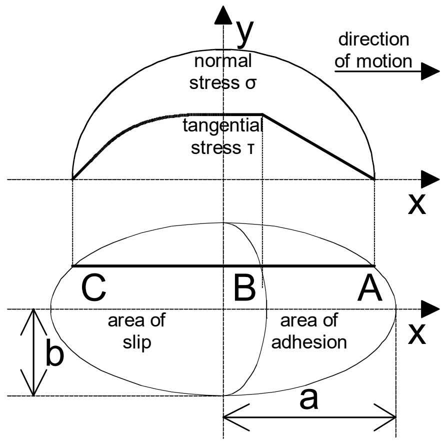


Figure 7. Assumption of the rail-wheel contact.

As a further simplification, a constant ellipse area is assumed. The maximum tangential stress τ is:

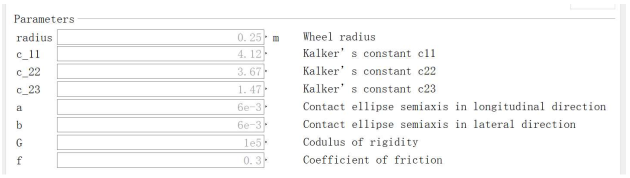
$$\tau = f \cdot \sigma \tag{9}$$

where f is the coefficient of friction and σ is the normal stress. f is assumed constant in the contact area. The method assumes a linear displacement between the leading point A to the trailing point C. At first, the contact area sticks firmly and the displacement is caused by the material creepage (area of adhesion). The tangential stress τ acts against the creep and grows linearly with the distance from the leading edge. If τ reaches the maximum value in the adhesion area, the relative movement of the contact area appears. This part is called area of slip.

Based on the theory, the relationship of the slip s and adhesion force is gotten.

2.2.2 Validation

The wheel-rail contact model based on Modelica is built according to the theory mentioned in section 2.2.1. The parameters and coefficient of friction f is assumed constant in the model as shown in Figure 8.

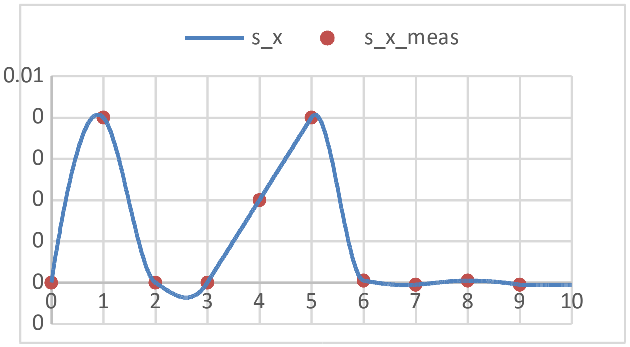


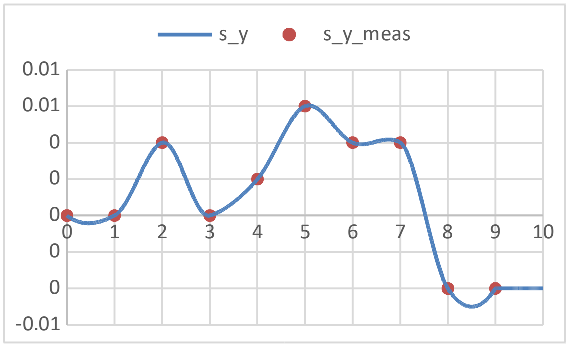
Parameters

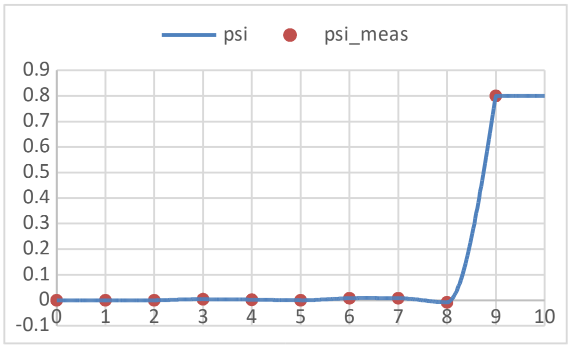
radius	0.25	m	Wheel radius
c_11	4.12		Kalker' s constant c11
c_22	3.67		Kalker' s constant c22
c_23	1.47		Kalker' s constant c23
a	6e-3		Contact ellipse semiaxis in longitudinal direction
b	6e-3		Contact ellipse semiaxis in lateral direction
G	1e5		Codulus of rigidity
f	0.3		Coefficient of friction

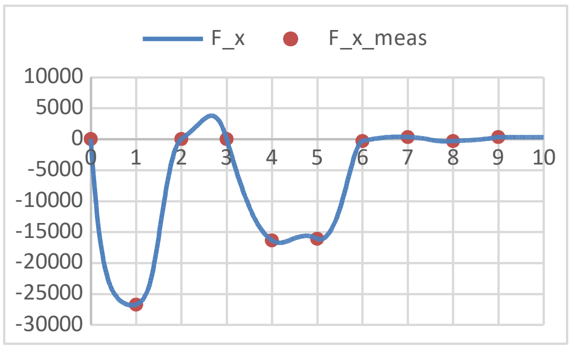
Figure 8. The parameters in the rail-wheel contact model.

The model was validated against the data in (Polach, 2000). The model proposed in this paper uses the same theory but adds the framework to multi-body simulation. The bluish continuous trajectories of the creep displacements s_x, x_y and spins ψ in the first three sub-figures of Figure 9 were applied to the model. The set of red dots 0..9 indicate the measurement values of (Polach, 2000). The other sub-figures show the resulting creep forces. The plots demonstrate that the Modelica model is in congruency with the paper results and proofs that the model can be used for the contact calculation.









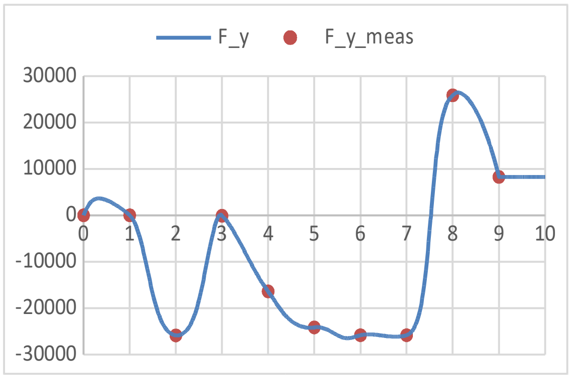


Figure 9: Validation of creep displacement and creep forces (x-axis: data points (discrete)/time (continuous)).

2.2.3 Outcome:

The following tasks can be implemented based on the mechanical models:

- Simulate the threshold of the lateral hunting fluctuation under the irregular rail or external forces, and the stability when passing through the curves.
- Simulate the damping effect in vertical direction of the train for human comfort evaluation.
- Simulate the energy consumption of the integrated train along the rail.

2.3 Thermal Fluid Modeling

When estimating energy consumption of the train, besides the auxiliary power consumption such as lighting, two critical factors of thermal and water management (i.e., A/C, water supply and drainage system) should be considered.

The purpose of thermal management is to control the cabin temperature in a proper range for the passengers' comfort and control the temperature of traction system to ensure the normal operation. Water management is to ensure the water supply and drainage in the cabin and washroom.

Based on the purpose to evaluate the energy consumption of the integrated train and the model level the behavioral model of the thermal and fluid models were built without considering the complex shapes of the flow machinery and pipelines. Only the dynamic equivalent model was used to analyze the model behavior.

2.3.1 Implementation

The main cooling types of the thermal management of the trains are air cooling and liquid plus air cooling. Compared to the thermal management system of automobiles, the biggest difference is that with the increase of the running speed, the demanded pressure difference increases, the flow decreases which may lead to the equipment being broken due to overheat in severe conditions. So it is critical to control the air conditioning system and cooling system according to the running speed.

The traction converter is liquid cooled, where the model consists of 4 cold plates, a pump and a cooling tank. The cooling medium is ethanol plus water.

The traction transformer is cooled through forced air cooling mixed with oil cooling. The cooling unit circulates the oil from the transformer to the oil cooling device by a pump where the heat is scattered by a fan.

The traction motors are cooled by forced air with one air inlet and two air exhausts whose air volume should coincide as good as possible.

The HVAC system of the train is mainly based on ventilation, which does not involve two phase flow. It provides cold air in summer and warm air in winter, as shown in Figure 10.

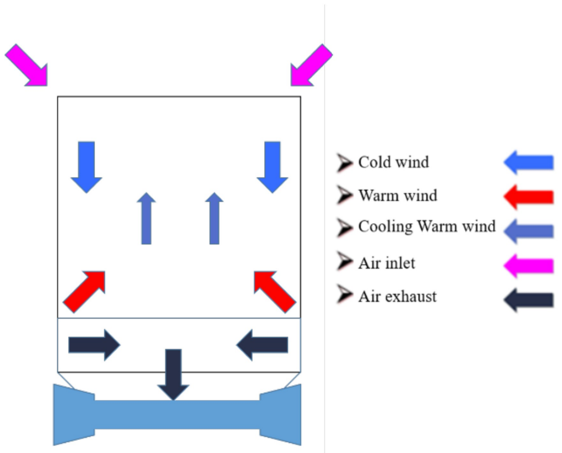


Figure 10. HVAC System in the train.

Water supply is added by further models where the dynamic behavior of the car may be influenced by the mass distribution.

2.3.2 Demonstration

CoolingLib was used to design cooling units, such as converter cooling unit, transformer cooling unit, and motor cooling unit. Figures 11 to 13 show the models and validation results (textual): the medium of the traction converters is water-glycol mixture solution, the medium of the transformer is 45# transformer oil.

Passenger comfort was evaluated by the HumanComfortLib by XRG Simulation GmbH. The

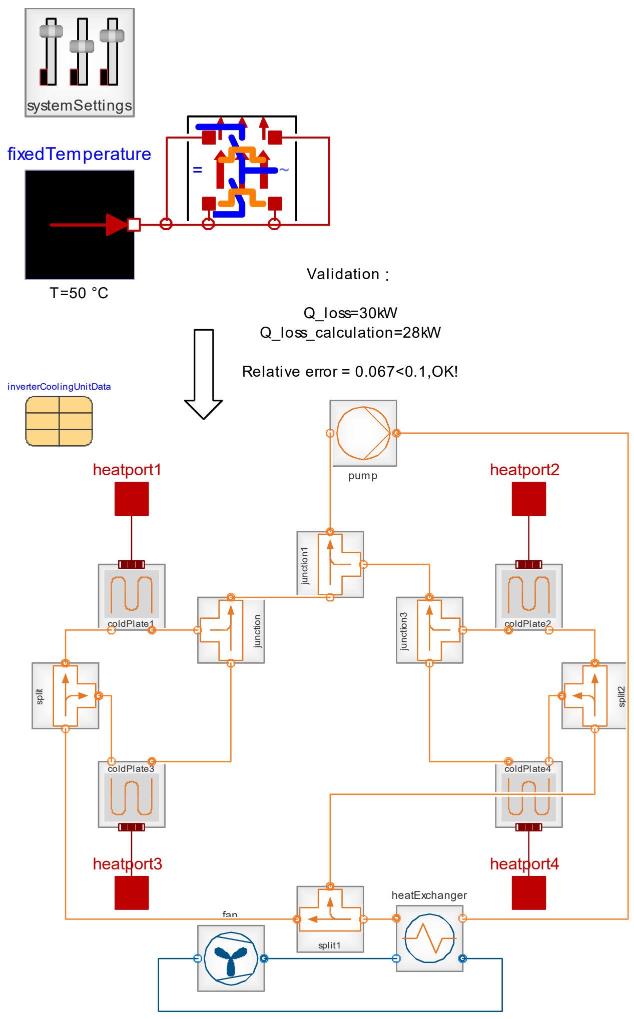


Figure 11. Traction converter cooling unit and Validation.

air conditioning and water supply system models were validated according to the passenger flow.

2.3.3 Outcome:

The thermal fluid model can be used to implement the following work:

- Simulate the cooling effect of the traction system and calculate the thermal capacity of the traction system.
- Simulate the air conditioning effect of the train for the human comfort evaluation
- Simulate the energy consumption of the auxiliary system in air conditioning system and water supply and drainage system to evaluate the energy consumption of the integrated train.

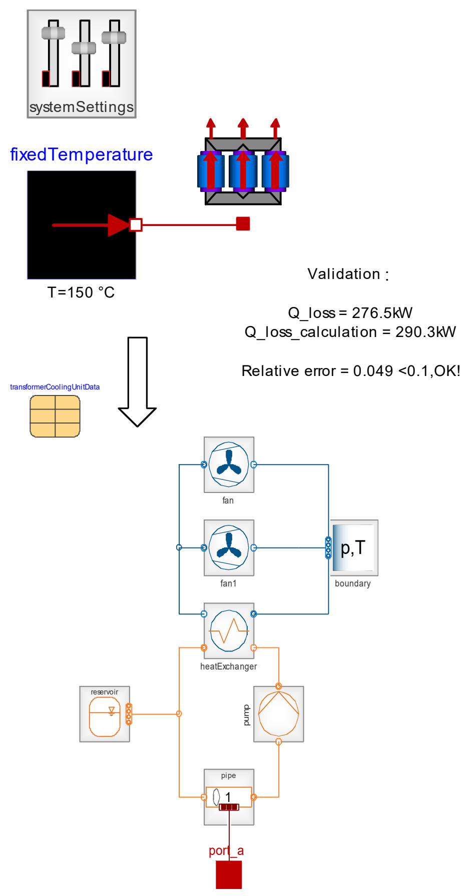


Figure 12. Transformer cooling unit and Validation.

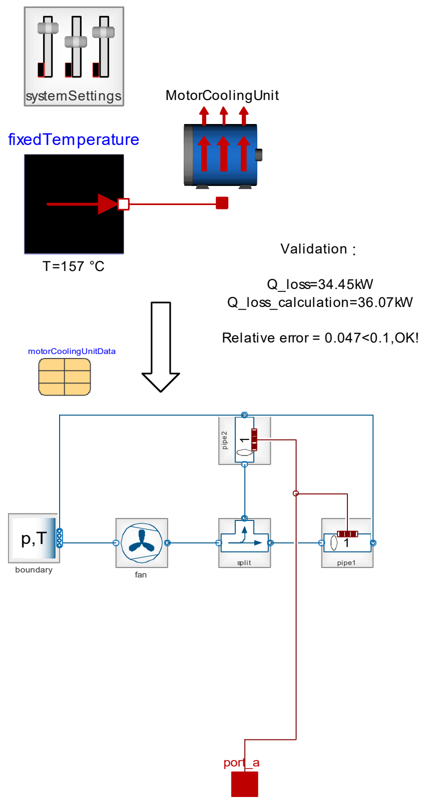


Figure 13. Motor cooling unit and Validation.

2.4 Traction System and Control Logics

The traction control and the control logics are closely linked and can be found in the figures presented in this chapter. The train control logic is of enormous relevance for industrial application and at the same time of limited scientific interest. For the sake of completeness, however, it should also be mentioned here.

2.4.1 Overview

Figure 14 shows an overview of the EMU structure. Each train consists of 8 cars where the rear 4 cars are mirror-inverted identical to the front part. Two trains can be coupled to a total number of 16 cars.

In nominal operation each half of a train is electrically independent and draws the power via the pantograph (T03/T06) from the power supply line. The circuit is grounded via the wheels and the railway. The power is transformed in the car with the pantograph to a lower voltage which is fed to the neighboring cars (M02/M04, M05/M07). Only those are driven cars with motors where the power is converted in these cars by traction current converters. The cars with the train conductor's cabins contain the auxiliary current converters and batteries and battery chargers.

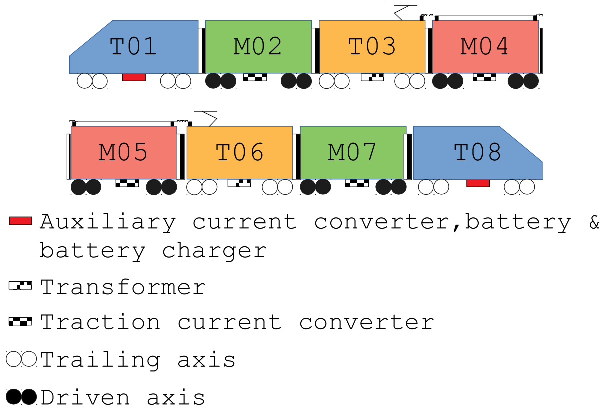

T01
M02
T03
M04
M05
T06
M07
T08
Auxiliary current converter,battery & battery charger
Transformer
Traction current converter
Trailing axis
Driven axis

Figure 14. Schematic diagram of traction system.

2.4.2 Implementation

The electrical system is shown in Figure 18 (some elements are hidden for better clarity). The model shows the electrical system of the train with cars 1-8. Mechanics are simulated by the element (9). The train is fed by the catenary. The impedance of the line and the railway (1) has substantial effects on the power quality to the line and maximum power off-take from the line. Considerations on the impedances can be found in (Hill et al., 1989; Allenbach, 2014). For the Chinese High Speed Railway the following parameters were assumed from literature:

Source voltage:	$25\,kV$
Net frequency:	$50\,Hz$
Series Resistance :	$0.09\,\Omega/m$

Series Inductance:	$0.669\,\mu H/m$
Parallel Capacitance	$7\,nF/m$
Parallel Conductance	$18\,\mu S/m$
Typical distance to power station	$14.7\,km$

Modelling does not take into account a dynamic change of the length but is implemented by Modelica.Electrical.Analog.Lines.Oline.

While the pantograph (2) is one of the most complex mechanical elements due to the high-speed operation, the Modelica model is limited to a resistance and sensors for net voltage and transferred current.

The voltage is transformed to a lower voltage level of around 2000 Volts by the transformers (4). The correct estimation of losses and harmonics demands a simulation with magnetic hysteresis effect. The Modelica.Magnetic library (Ziske and Bödrich, 2012) offers accurate and fast methods while the choice of magnetic core material is very limited. The transformer parameterization is crucial for the correct estimation of harmonics. Parameters may come from detailed design or hardware tests.

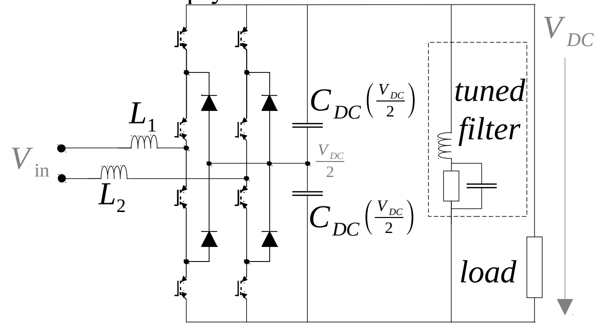
Number (3) is the power conversion system including AC to DC bi-directional converters, intermediate voltage circuit, current converter and traction motors.

(5) to (9) are control logics: (5) communication models , e.g. passing neutral phase information, (6) the driver's commands, (7) the Central Control Unit of the train, (8) the Traction Control Unit with one TCU per driven car.

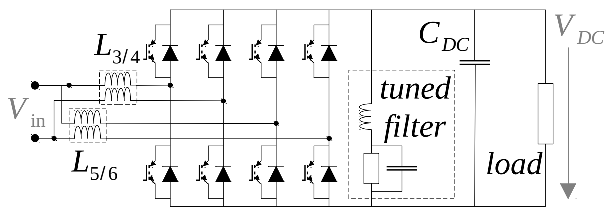
The traction system is shown in detail in Figure 19. For this train, the AC input voltage (left) is converted into an DC intermediate voltage and inverted to AC voltage by (20) to feed the parallel traction motors. The input converter should be able to convert the AC voltage passively into a lower DC voltage if no traction but only auxiliary power is needed, boost the AC voltage to a higher intermediate DC level or feed regenerated power back to the network when braking. This four-quadrant converter needs to be Power Factor Correcting (PFC) for good power quality in the supply line without interaction of trains, losses by reactive power or radiated harmonic noise.

Figure 15 shows two typical converter topologies. The converter with Neutral Point Clamped (NPC) has the benefit that all transistors only have to resist to half the DC voltage. In the last years new IGBTs were developed which are rated for the higher voltage level which enables the second type of converter: the interleaved boost converter. For both converters, transformer leakage inductance can be used as the boost inductances L_1/L_2 respectively L_3 to L_6. Due to the redundancy and other benefits, the interleaved converter was implemented in the library (Figure 19: 11,12, with precharge 10 and control 13). Considerations on the efficiency and rating of both topologies can be found in (Bellini et al., 2002) and (Musavi et al., 2010).

Typically the switching frequency is selected low to prevent switching losses. If the switching frequency is below 11 times the natural time constant of the input impedance, then simple current controls may fail (Freyberger, 2002). The library offers an advanced control concept, including a reliable phase detection and control in dq system.


V_{DC}
L_1
L_2
V_{in}
$C_{DC}\left(\frac{V_{DC}}{2}\right)$
$\frac{V_{DC}}{2}$
$C_{DC}\left(\frac{V_{DC}}{2}\right)$
tuned filter
load

Bi-directional PFC boost converter with NPC inverter



Bi-directional PFC interleaved boost converter

Figure 15. Boost converter topologies.

For both types of boost converters, the control aims to preserve power quality on the AC side V_{in} by drawing power from V_{DC} at double net frequency. The ripple can be attenuated by a tuned filter (14) of first or second order. The intermediate voltage is stabilized by capacitors (15). Power is drawn from the auxiliary system (17) or the traction inverter (19). Each traction converter feeds four induction motors connected in parallel. (16) and (18) are overvoltage protection and braking resistor.

2.4.3 Demonstration

The electrical model is demonstrated by a study of the net side power quality. The input current of one transformer feeding two conversion/traction units shall be investigated at nominal load (e.g. car 2-4). For better efficiency, the traction system is simulated by a more simple controlled power load.

Results are shown in Figures 16 and 17. The former one shows the currents on the transformer's secondary windings which are disturbed by pulse-width-modulation (PWM), where the currents combine to a smooth current in the primary windings. Figure 17 shows the load profile and transient of the actively

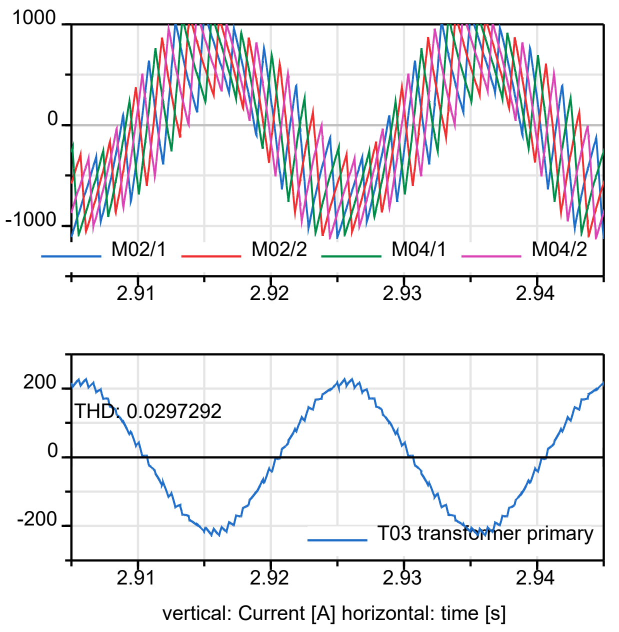


Figure 16. Harmonic test of current drawn by one transformer. Top: Currents of each interleaved converter. Bottom: transformer input current and THD.

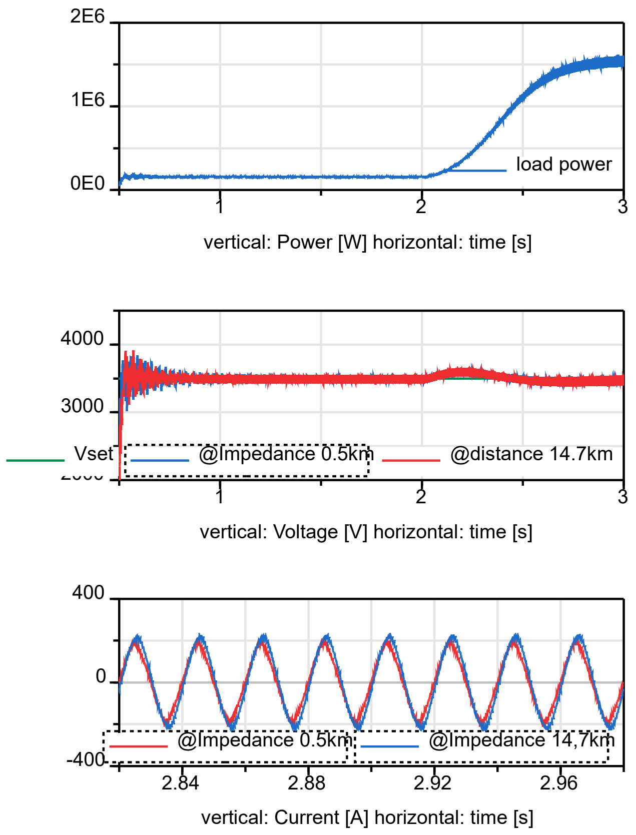


Figure 17. Harmonic test of current drawn by one transformer. Top: Load profile. Middle: Trajectory of controlled DC intermediate voltage @ distance 0.5km and 14.7km to power source. Bottom: Current shape for distance 0.5km and 14.7km.

controlled DC intermediate voltage and AC input currents at two impedance values reflecting 14.7km and 0.5km to the power station. The THD is well below the limit of the industrial standard.

2.4.4 Outcome:

All tasks which are listed above rely on the electrical models and control models. Detailed electrical models are utilized for

- sizing of electrical components for stability, power quality and rating, like inverter bridges, transformers, passive components,
- detailed sizing of heat dissipation in connection with simplified cooling system,
- validation of simplified models.

Simplified non-fast-switching models are used for

- Vehicle energy consumption estimation in conjunction with simple vehicle dynamics and power systems
- Traction system thermal capacity estimation for cooling system layout and control of power reduction

3 Conclusion and future work

In this paper, needs and implementations for system modeling of high speed trains with focus on the Beijing-Zhangjiakou Intercity Railway were shown. The models cover all relevant domains and tests and optimization for the real train system will be performed with it in the near future.

Acknowledgment

The authors gratefully acknowledge ZEMT for their help of providing the system information and test data. SIMTEK also cooperated with ZEMT on the control and health monitoring of the braking system.

Bibliography

Allenbach, J.-M. (2014). **Eisenbahntechnik (German)**. Available https://documents.epfl.ch/groups/t/tr/traction/www/documents/ZusammET.pdf.

Bellini, A., Bifaretti, S., and Constantini, S. (2002). **High power factor converters for single-phase AC traction drives**. In *WIT Transactions on The Built Environment* 61.

Belmon, L., and Liu, C. (2011). **High-speed train pneumatic braking system with wheel-slide protection device: A modelling application from system design to HIL testing**. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*, p. 549-556.

Dumont, E., and Maurer, W. (2012). **DyMoRail: A Modelica Library for modelling railway buffers**. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, p. 691-696.

Franke, R., and Wiesmann, H. (2014). **Flexible modeling of electrical power systems--the Modelica PowerSystems library**. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, p. 515-522.

Freyberger, F. (2002). **Leittechnik: Grundlagen, Komponenten, Systeme; Projektierung, Steuerung und Regelung, Signal-, Bussysteme, Aktoren, Sensoren (German)**. Pflaum.

Frilli, A., Meli, E., Nocciolini, D., Pugi, L., and Rindi, A. (2016). **Energetic optimization of regenerative braking for high speed railway systems**. In *Energy Conversion and Management* 129, p. 200-215.

Heckmann, A., Keck, A., Kaiser, I., and Kurzeck, B. (2014). **The foundation of the dlr railwaydynamics library: the wheel-rail-contact**. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, p. 465-475.

Hill, R., Carpenter, D., and Tasar, T. (1989). **Railway track admittance, earth-leakage effects and track circuit operation**. In *Railroad Conference, 1989. Proceedings., Technical Papers Presented at the 1989 IEEE/ASME Joint*, p. 55-62.

Masao, N. (2008). **Vehicle system dynamics and control (orig: Japanese, ISBN 978-4-8425-9901-4)**. The Japan Society of Mechanical Engineers.

Musavi, F., Eberle, W., and Dunford, W.G. (2010). **Efficiency evaluation of single-phase solutions for AC-DC PFC boost converters for plug-in-hybrid electric vehicle battery chargers**. In *Vehicle Power and Propulsion Conference (VPPC), 2010 IEEE*, p. 1-6.

Polach, O. (2000). **A fast wheel-rail forces calculation computer code**. In *Vehicle System Dynamics* 33, p. 728-739.

Zimmer, D. (2012). **A planar mechanical library for teaching modelica**. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, p. 681-690.

Ziske, J., and Bödrich, T. (2012). **Magnetic hysteresis models for modelica**. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, p. 151-158.

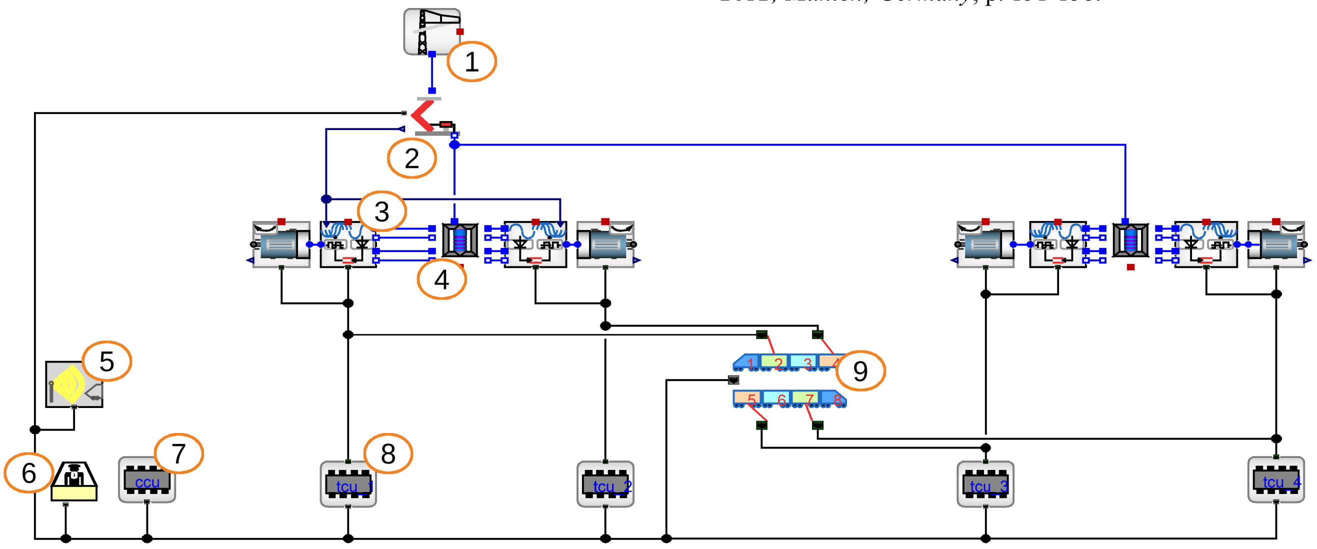


Figure 18. Modelica diagram of test model of the traction system (simplified).

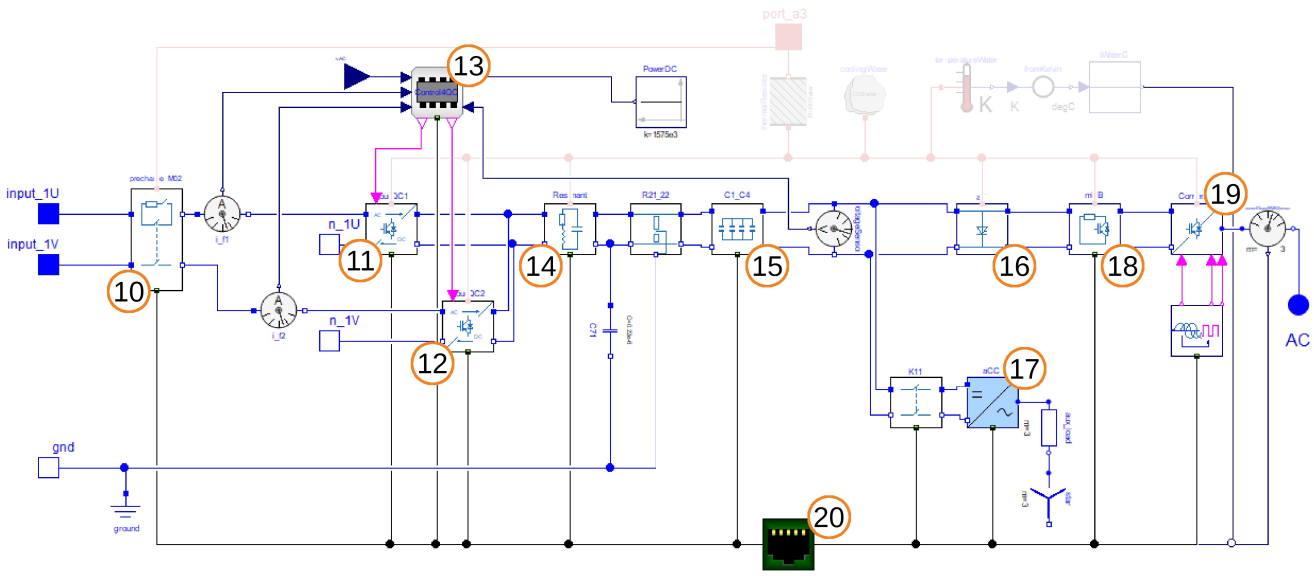


Figure 19. Schematic diagram of the power conversion system.

Session 3D: New Applications

Modeling Supply and Demand in Modelica

Michael M. Tiller[1]

[1]michael.tiller@gmail.com

Abstract

This paper demonstrates using component oriented modeling and acausal semantics to create a basic library of behavioral components to model supply and demand. The models presented are each steady state models. While some examples include shifting economic conditions that cause the equilibrium points to change during the simulation, none of the models feature dynamic states. The main purpose of this paper is to demonstrate to people unfamiliar with Modelica (Modelica Association 2017) how Modelica can be used to model non-engineering systems and how it makes such modeling faster, easier and less error-prone compared to other approaches (*e.g.,* using spreadsheets).

Keywords: Modelica, economics, supply and demand

1 Introduction

Modelica was designed from the outset to be domain neutral. The hope was that the foundations of Modelica were sufficiently complete that it could not only be used to model the wide range of engineering related systems that the designers were familiar with but that it was universal enough to model nearly any system from any domain, even those unfamiliar to the language designers. The wide range of domains that Modelica has been applied to over the last 20 years is a testament to the success of these design goals.

As I hope to demonstrate in this paper, the acausal semantics in Modelica are not only useful for describing the familiar conservation laws present across engineering domains. These semantics can be utilized whenever there is a need to ensure a proper accounting of many different quantities. In this particular case, we will focus on the movement of goods passing from producers to consumers.

In this paper, we will introduce the economic concepts of supply and demand. These concepts are often discussed only in qualitative terms. But if you characterize supply and demand quantitatively, you can use the features of Modelica to create a library of components models that can model not just sources of supply (production) and demand (consumers) but also model other economic effects such as taxation, complementary goods, exchange rates, *etc.*

The goal of this paper is not to provide a comprehensive collection of quantitative models of economic actors and effects. Instead, this paper attempts to achieve two primary goals. First, to demonstrate the applicability of Modelica to yet another domain. In this case, the non-engineering related subset of economics that involves modeling of supply and demand. The other goal of the paper is to describe the models in such a way that someone familiar with economics but unfamiliar with Modelica will appreciate how Modelica works and how it could be useful in the field of economics to create trusted and reusable libraries of components that are capable of performing all the necessary bookkeeping required for supply and demand systems and solve the underlying non-linear systems of equations better than other approaches (*e.g.,* using spreadsheets).

There are a number of online books (Taylor 2018; Hutchinson 2016; Posner and Tayari 2018) that cover the topics in this paper in much greater depth and I would encourage the reader to seek out these books to learn more about these topics from experts. Furthermore, there have been previous articles that used Modelica to model economic effects (Zimmer and Schlabe 2012; Casella, Miragliotta, and Uglietti 2005). However, these papers focused on specific types of markets and used slightly different approaches.

2 Mathematics of Supply and Demand

Before we discuss the details of the Modelica implementation, it will be useful to provide a basic discussion of the topic of supply and demand curves, how they are characterized and how we can use them to arrive at a supplied price and supplied volume.

Both supply and demand curves are expressed with price as the dependent variable and sales volume as the independent variable. This choice is unintuitive because price is the thing that is most controllable here and volume is simply a consequence of the chosen price. Nevertheless, this is the way supply and demand curves are typically represented and so this paper follows that convention as well.

2.1 Supply

As mentioned previously, the supply curve shows price as a function of sales volume. The curve visualizes what the per unit price would be for a given sales volume. A very simple supply curve is shown by the blue line in Figure 4. One important characteristic of a supply curve is that *it generally has a positive derivative*. At first, this seems counter-intuitive because most producers actually discount their products if customers are willing to buy more of the product. But this is related to pricing strategy and is contingent on being able to scale up production.

But supply curves are generally based on availability of limited resources (*e.g.,* Uber's "surge pricing"). As such, as demand for a limited resource goes up, the price that a producer can command will go up. In the *long run*, the market may compensate by increasing production which, in turn, increases overall supply and lowers prices. But this is an example of how the supply curve itself adapts to market conditions over time.

Another reason for the supply curve to have a positive derivative is related to accessibility of raw materials. Even if the amount of the raw material doesn't have a finite limit, it may be the case that there are multiple sources of the raw material and that some are more expensive than others. In such a case, the shape of the supply curve is a reflection of the fact that the initial sales will rely on easily accessible (*i.e.,* cheaper) sources while larger sales will require less accessible (*i.e.,* more expensive) sources.

2.2 Demand

While supply curves represent the availability of a given good, the demand curve represents how much consumers are willing to pay for a good. A very simple demand curve is shown by the red line in Figure 4. Unlike the supply curve, the demand curve generally has *a negative first derivative*. The simplest way to understand this is to think about a demand curve as a *histogram*. Imagine the consumer who values this good the most. They define the maximum possible price (*i.e.,* the y-intercept on demand curve). If producers offer that product at that price, they can expect to sell only to the wealthiest or most enthusiastic consumers. However, if producers reduce their price, they can expect to attract even more buyers. As they continue to lower the price, they can reasonably expect to continue to attract more buyers. In this sense, the demand curve is a histogram showing how much consumers are willing to pay.

3 Interfaces

3.1 Connector

The cornerstone of any Modelica library is the `connector` definitions. This is because the `connectors` define the way in which components interact. So it is necessary to carefully design the `connectors` so they can represent all potential interactions.

In the case of modeling supply and demand, there are two fundamental quantities we are concerned with. The first is the *price* of goods. We will talk about how price impacts the behavior of both producers and consumers shortly. But for now, all we need to recognize is that price motivates transactions to occur.

The other fundamental quantity is *volume* of sales. This represents the number of goods either produced or consumed (by producers and consumers, respectively). Our systems will be formulated such that all goods must be accounted for. This means that all goods produced have to go *somewhere*. It might be into a warehouse, it might be purchased by a consumer, it might be transported to a geographically remote market. But it must be accounted for.

As such, the volume of sales will be the `flow` variable in our system. In this way, the acausal semantics of Modelica, normally used to account for conserved quantities in engineering domains, will ensure our constraint that all goods are accounted for. Since the volume of sales is the `flow` variable, we will adopt the price as our across/potential variable.

Before we define the `connector`, let us first introduce two types:

```
type Price = Real(min=0, quantity="Price");
type SalesVolume = Real(quantity="Units");
```

With these two types defined, we can now define our connector as follows:

```
connector Market "Market interaction"
  Types.Price price(start=10);
  flow Types.SalesVolume volume;
end Market;
```

We establish a `min` and `start` value on the `Price` type to assist solvers in finding solutions for non-linear systems. The `min` attribute informs the solvers that negative values are not viable solutions. The `start` attribute provides an initial guess which helps the solver locate a solution and/or choose between multiple solutions. In the `Market` connector, we chose the rather arbitary value of 10 as an initial guess just to provide a positive initial guess. In specific models, thise `start` attribute can be overriden to provide a better problem specific initial guess. The non-linear solvers will also need good initial guesses for `volume`, but we cannot provide `min` and `start` values here because the sign will depend on the nature of the component so we will instead add those attributes on variables whose sign is known.

3.2 Partial Models

Our `connector` is defined in the `Interfaces` subpackage along with a few useful partial models.

3.2.1 Producer

The first of these partial models is a `Producer` model. The idea behind the `Producer` model is to define some protected variables associated with and employing the *sign convention* of a producer. Specifically, the normal Modelica sign convention is that flow of a conserved quantity (in this case, goods) is positive when flowing into a component. In the case of a producer, goods are always flowing *out*. As such, the `volume` field on the connector is always negative. However, the supply curve volume is always positive. For this reason, within the `Producer` model we define a local variable, `volume`, which represents the independent variable on the supply curve (*i.e.,*

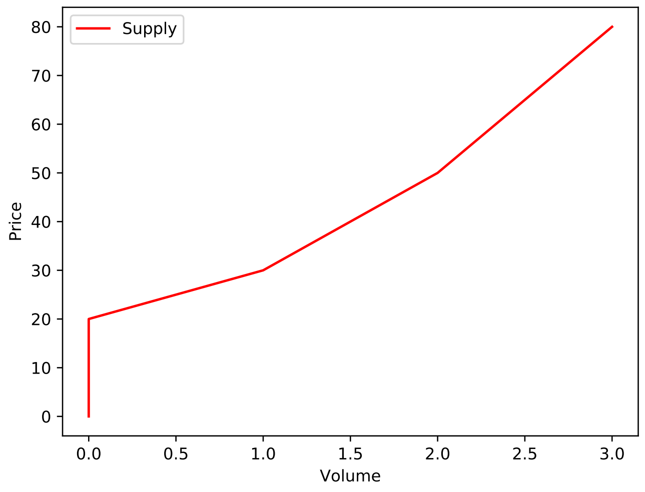


Figure 1. Sample S-parameterized Supply Curve

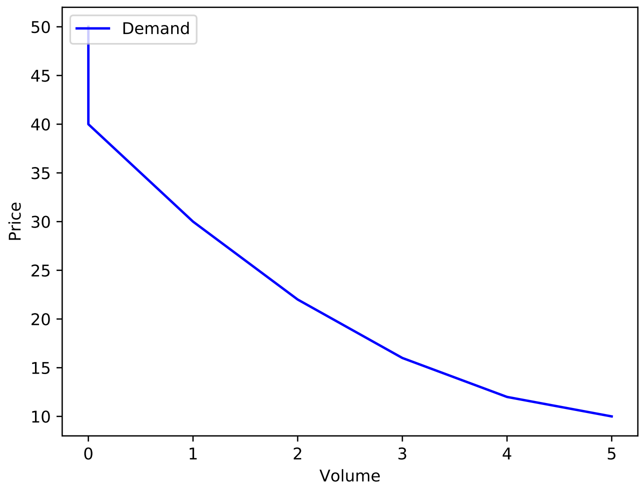


Figure 2. Sample S-parameterized Demand Curve

normally positive) and map that to the `volume` field on the `connector`, *i.e.,*

```
partial model Producer "Goods producer"
  Types.SalesVolume volume;
  Types.Price price;
  Interfaces.Market market(volume(start
      =-10));
protected
  Real s(start=-1);
equation
    if (s<0) then
      price = market.price;
      volume = -s;
    else
      price = market.price-s;
      volume = 0;
    end if;
  market.volume = -volume;
end Producer;
```

The first thing to notice in this model is the fact that it doesn't just define variables for `price` and `volume` but also a variable named `s`. Internally, the supply curve is not strictly represented as `price` as a function of `volume`. Instead, both `price` and `volume` are represented in terms of `s`. The resulting supply curve (parameterized in terms of `s`) is shown in Figure 1.

Doing the parameterization in this way allows us to define multiple potential prices for a given volume. This allows us to handle the case where the lowest possible production price is still above the highest price that consumers are willing to pay. Using this parameterization, we extend the supply curve to indicate that no goods will be produced (`volume=0`) for all prices below the lowest possible production costs. This allows us to solve for a supplied price and supplied volume in the case where a producer (or consumer) is priced out of the market.

3.2.2 Consumer

The `Consumer` model is very similar to the `Producer` model. It doesn't actually need to perform the sign change on `volume` but it does implement a similar s-parameterization of the demand curve except that in the case of the demand curve the `s` parameter extends the price *upward* rather than downward as shown in Figure 2.

```
partial model Consumer "Goods consumer"
  extends Curve;
  Types.SalesVolume volume;
  Types.Price price;
  Interfaces.Market market;
protected
  Real s(start=1) "Volume or price gap";
equation
  if (s<0) then
    price = market.price + s;
    volume = 0;
  else
    market.price = price;
    volume = s;
  end if;
  market.volume = volume;
end Consumer;
```

4 Supply and Demand

With the connectors and `partial` models defined, we can start defining various models for both supply and demand.

4.1 Linear Models

Many explanations of supply and demand use linear supply and demand curves to describe how to arrive at the supplied price and volume. So we'll start with such models and then transition into more realistic models of supply and demand shortly.

Consider a market where the highest price a consumer is willing to pay would be \$12. But for every \$1 that we reduce the cost of the good, we find 20 more customer. Let us further assume that the producer of these goods must charge at least \$10 and for every \$1 increase in price, 5 more goods can be supplied.

The supply curve would then be represented by:

$$p^s(v) = p_0^s + \beta * v = 10 + 0.2 * v$$

where β represents how much the production price would increase with each additional unit of goods produced. In the same way, the demand curve would be represented by:

$$p^d(v) = p_0^d - \alpha * v = 12 - 0.05 * v$$

where α represents how much the price would have to be reduced in order to sell each additional unit of goods.

These two curves are shown as the blue and red lines, respectively, in Figure 4. We want to find a combination of price and volume that are consistent with both the supply curve and the demand curve. In fact, what we are looking for is the *intersection* of these two curves. This is a price/volume point that satisfies both the consumers and the producer. As we can see in Figure 4, the price for goods at this point is called the *supplied price* and the volume of goods sold in that scenario is the *supplied volume*. The supplied volume is the volume, v_s, at which the price on the supply curve matched the price on the demand curve. In other words,

$$p^s(v_s) = p^d(v_s)$$

Note that because of the connection semantics of Modelica, this equation is automatically generated whenver we connect the `Market` connector of the consumer and the producers. This equation combined with the "conservation equation" generator by the connector which, in the case of a system containing only a consumer an producer as the effect of setting the `volume` values used by both to be equal, means that for this use case we can trivially determine that the supplied volume must be:

$$\frac{p_0^d - p_0^s}{\alpha + \beta} = \frac{12 - 10}{0.2 + 0.05} = \frac{2}{0.25} = 8$$

Plugging this supplied value in the supply (or demand) curve tells us that the supplied price must, therefore, be $10 + 0.2 * 8$ or \$11.6. To model this in Modelica, we can create the following two models to represent the supply and demand curves respectively:

```
model LinearProducer
  "Production with a minimum price and
     linear price increase"
  extends Interfaces.Producer;
  parameter Types.Price min_price "Minimum
     price to produce";
  parameter Types.PriceSensitivity beta "
     Price increase as a function of
     volume";
equation
  price = min_price + beta*volume;
end LinearProducer;

model LinearConsumer "Linear distribution
    of consumers"
  extends Interfaces.Consumer;
```

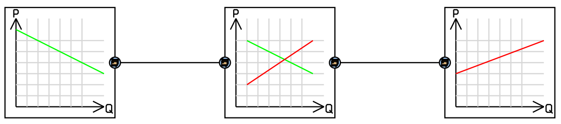


Figure 3. `LinearMarket` model

```
  parameter Types.Price max_price "The
     largest amount any consumer is
     willing to pay for this good";
  parameter Types.PriceSensitivity alpha "
     Rate of price drop as volume
     increases";
equation
  price = max_price - alpha*volume;
end LinearConsumer;
```

With these two models in hand, we can create a system market model in Modelica as follows:

```
model LinearMarket
  "Market where consumer and producer have
     linear relationships"

  Components.LinearConsumer consumer(
     max_price=12, alpha=0.05);
  Components.LinearProducer producer(
     min_price=10, beta=0.2);
  Components.MarketAnalysis market;
equation
  connect(market.producers, producer.market
     );
  connect(market.consumers, consumer.market
     );
end LinearMarket;
```

A diagram of our system model is shown in Figure 3. Note that to solve for the supplied price and supplied demand all we need to do is connect the `Market` connector of the consumer and the producer. But for this model we have introduced a special "intermediary" called the `MarketAnalysis` model in the center between the consumer and producer. This `MarketAnalysis` model enforces a market equilibrium condition (just as if we had directly connected the consumer to the producer) but only *at the start of the simulation*. This means that the price that the consumer is willing to pay has to match the price that the producer is willing to charge. Furthermore, the volume of goods that the producer produces must be equal to the volume of goods that consumers consume.

This solution is found at the start of the simulation. From that point (and over the following 1 second of simulation time), the `MarketAnalysis` model perturbs the system into non-equilibrium states. As a result of this process, it is possible to visualize the supply and demand curves parametrically. The results of the Modelica simulation are shown in Figure 4.

4.2 Exponential Models

Linear models work well to explain the concept of supply and demand as well as the idea of supplied price and supplied volume because it is straightforward to find a closed

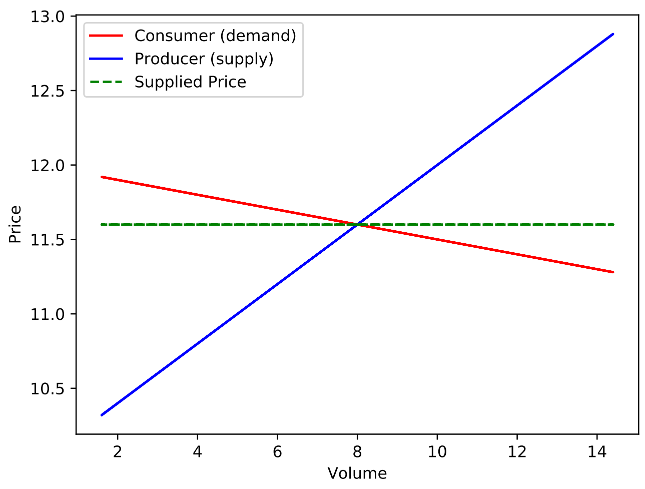


Figure 4. Simulation results for `LinearMarket`

form solution for two intersecting lines. But linear models are problematic because they don't make much sense. Very quickly the lines cross the axes and leave the first quadrant. For example, once the demand line crosses the x axis, the price goes negative. This reflects a situation where producers would have to pay consumers to take their products. While there are markets where this effect could be seen, it isn't a normal situation and you would generally have to have supplied many, many consumers at positive prices before this is likely to happen (something not usually reflected by a linear demand curve). Similar problems occur when the supply curve leaves the first quadrant.

A more realistic model is an exponential model. As with the linear models, this model also defines a price point on both the supply and demand curve associated with a sales volume of zero (*i.e.,* p_0^s and p_0^d). But instead of assuming a linear relationship, we introduce parameters representing an exponential price decay or growth.

Let's start with the demand curve. The equation for an exponential demand curve would be:

$$p^d(v) = p_0^d e^{-k_d v}$$

As before, p_0^d represents the maximum that consumers would be willing to pay. But with this model of demand, that price falls off exponentially with volume. An important characteristic of such a demand curve is that it never drops below zero. In other words, as the price approaches zero, the potential volume of sales approaches infinity.

The Modelica code for this model is:

```
model ExponentialConsumer "Exponential
    distribution of consumers"
  extends Interfaces.Consumer;
  parameter Types.Price max_price "The
      largest amount any consumer is
      willing to pay for this good";
  parameter Real decay(min=0) "Exponential
      decay rate as a function of volume";
equation
```

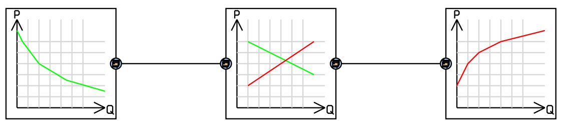


Figure 5. Exponential curves for consumer and producer

```
  price = max_price*exp(-decay*volume);
end ExponentialConsumer;
```

The supply curve is slightly different. It is characterized by the following equation:

$$p^s(v) = p_0^d e^{k_s v}$$

Again we see p_0^d, the minimum price that goods can be produced for. But now instead of a linear increase in price with respect to sales volume, we see an exponential curve. Whereas the demand curve tapers off with volume, the supply curve grows exponentially because as more and more of a finite resource is consumed, the cost of the resource sores.

In Modelica, this model can be expressed as:

```
model ExponentialProducer "Exponential
    pricing curve"
  extends Interfaces.Producer;
  parameter Types.Price min_price "Price at
      zero volume";
  parameter Real growth(min=0) "Exponential
      growth rate as a function of volume"
      ;
equation
  price = min_price*exp(growth*volume);
end ExponentialProducer;
```

We can combine an exponential models of supply and demand to create a simple system model as shown in the following Modelica model:

```
model ExponentialMarket
  "Market where consumer and producer have
      exponential relationships"

  Components.MarketAnalysis market(minScale
      =1, maxScale=2.5);
  Components.ExponentialConsumer consumer(
      max_price=12, decay=0.04);
  Components.ExponentialProducer producer(
      min_price=10, growth=0.06);
equation
  connect(consumer.market, market.consumers
      );
  connect(market.producers, producer.market
      );
end ExponentialMarket;
```

The diagram for this model is shown in Figure 5. Simulating this model we get the results shown by the thick lines in Figure 6. Note the slight curvature of the lines vs. the linear model. The curvature would be more pronounced if we considered a wider range of sales volumes.

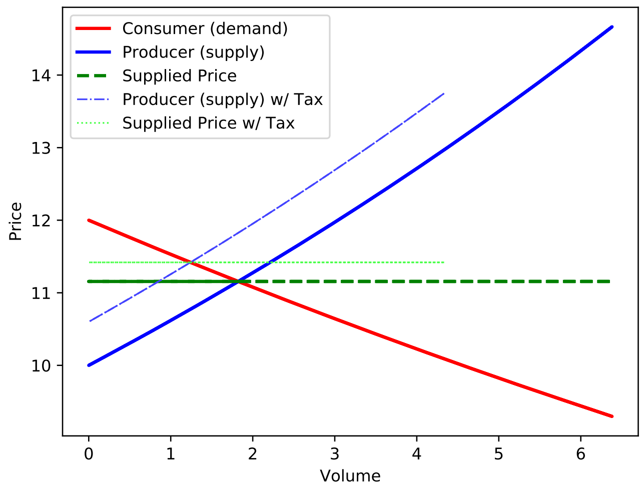


Figure 6. Simulation results for `ExponentialMarket`

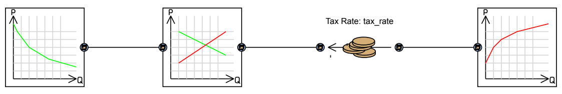


Figure 7. Adding taxation to `ExponentialMarket`

5 Scenarios

With these exponential models in place, a variety of interesting scenarios open up because we have the building blocks necessary to start modeling real markets. Because we have graphical component models we can now compose these scenarios simply by dragging and dropping these consumer and producer models down into a diagram and combining them with various other economic factors.

5.1 Taxation

A very simple adjustment we can make to the market is to introduce a tax and see how that impacts the supply and demand. In our previous example, `ExponentialMarket`, the supplied price was \$11.16 and the supplied volume was 1.82. Now let us revise the model to include a model of taxation. The taxation model itself can be implemented as follows:

```
model Tax
  "Model a tax (increasing effective price
     to consumers)"
  parameter Types.TaxRate taxRate;
  output Types.Price taxRevenue;
  Interfaces.Market consumers;
  Interfaces.Market producers;
equation
   consumers.price = producers.price*(1+
       taxRate);
   taxRevenue = producers.price*taxRate*
       producers.volume;
   consumers.volume+producers.volume = 0;
end Tax;
```

Add this to our overall system, we then get:

```
model ExponentialMarketWithTaxes
  "Market where consumer and producer have
     exponential relationships"

  Components.MarketAnalysis market(minScale
     =1, maxScale=2.5);
  Components.ExponentialConsumer consumer(
     max_price=12, decay=0.04);
  Components.ExponentialProducer producer(
     min_price=10, growth=0.06);
  Effects.Tax tax(taxRate=0.06);
equation
  connect(consumer.market, market.consumers
     );
  connect(tax.producers, producer.market);
  connect(tax.consumers, market.producers);
end ExponentialMarketWithTaxes;
```

The diagram for this model is shown in Figure 7. Running this model, which includes the same supply and demand curves, we find that the supplied price has risen from \$11.16 to \$11.42 and the supplied volume has dropped from 1.82 to 1.24.

In the untaxed case, the consumers paid \$20.30 for the goods and all that revenue went to the producers. In the taxed case, consumer spending dropped to \$14.17 and, of that, only \$13.36 went to the producer. The remaining \$0.81 was collected as tax revenue. Note that this seems like a dramatic effect for a 6% sales tax. But please note that the supply and demand curves are completely arbitrary in this example.

5.2 Raw vs. Finished Goods

The next example involves manufacturing. Specifically, we have producers of two different *raw* materials and those are then manufactured into a finished good which is sold to consumers. This example demonstrates the concept of *complementary goods*. Two goods are complementary if demand for one drives of demand for the other because purchasers of one may want (or require) the other good as well.

In order to model our manufacturing system, we must introduce the following model of the `Manufacturer`:

```
model Manufacturer "Combines two types of
    goods to form a third"
  parameter Real markup;
  Interfaces.Market production;
  Interfaces.Market supply_A;
  Interfaces.Market supply_B;
equation
  supply_A.volume + production.volume = 0;
  supply_B.volume + production.volume = 0;
  production.price = (supply_A.price+
     supply_B.price)*(1+markup);
end Manufacturer;
```

This model acts as *both* a consumer and a producer. For each good it produces (to the `production` market), it consumes one good from `supply_A` and another from `supply_B`. In addition, the price that it offers its finished goods for is the price it must pay for the two raw goods plus some percentage markup.

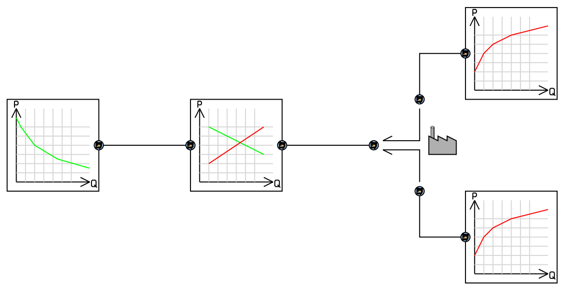


Figure 8. `SupplyChain` model

The final system model, shown in Figure 8, is expressed in Modelica as follows:

```
model SupplyChain "Model of manufacturing
    supply chain"
  Components.Manufacturer manufacturer(
      markup=0);
  Components.MarketAnalysis market;
  Components.ExponentialConsumer consumer(
      max_price=12, decay=0.04);
  Components.ExponentialProducer producer_A
      (growth=0.06, min_price=4);
  Components.ExponentialProducer producer_B
      (growth=0.06, min_price=6);
equation
  connect(market.producers,
      manufacturer.production);
  connect(consumer.market, market.consumers
      );
  connect(producer_A.market,
      manufacturer.supply_A);
  connect(producer_B.market,
      manufacturer.supply_B);
end SupplyChain;
```

For this case, the supplied price for the finished goods is \$11.59 at a supplied volume of 0.87.

5.3 Competition for Resources

We can add an interesting twist to the previous model if we add additional consumers for one of the raw materials. This will drive up demand for that good and, as a consequence, increase the price for that particular raw material. Because our goods are complementary, the manufacturing process requires both and the price of the finished goods should rise as a result of the competition for the raw materials.

In this case, we do not need any new models. As seen in Figure 9, we can simply extend the `SupplyChain` model with another consumer for one of the raw materials, *e.g.*,

```
model ResourceCompetition
  extends SupplyChain;
  Components.ExponentialConsumer
      raw_consumer(decay=0.04, max_price=5)
      ;
equation
  connect(raw_consumer.market,
      producer_A.market);
```

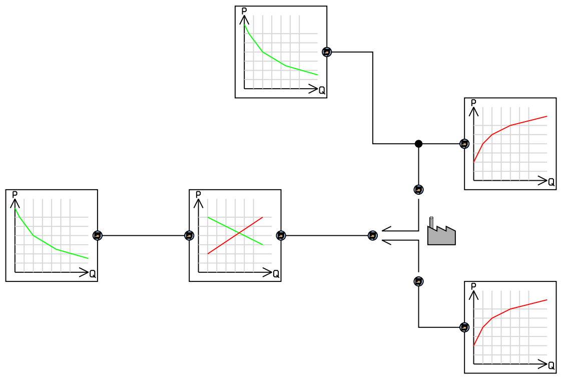


Figure 9. Adding competition for raw materials

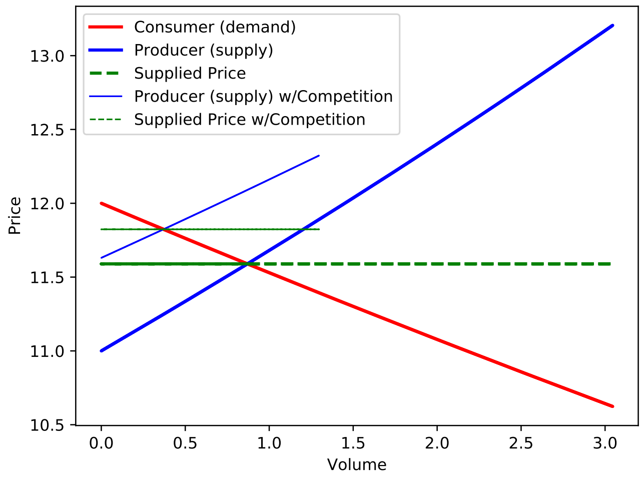


Figure 10. Comparison with and without resource competition

```
end ResourceCompetition;
```

We can see from the results, indicated by the thin lines in Figure 10, that the supplied price of the finished materials has risen to \$11.82 and the supplied volume has been reduced to 0.37 because of this competition for the raw materials.

5.4 International Trade

One final example involves international trade. In this model, shown in Figure 11, we have two distinct consumers and producers. Each is located in their own geographical region. Left alone, each consumer would trade only with the producer in their own geographical region. But if we add the ability to ship goods between the geographies (with the associated transportation costs), then we create a global market. But there are other factors that impact global trade besides just transportation costs. Tariffs may be in place to limit the amount of global trade. Furthermore, currency exchange rates will also affect the price of goods.

For our final example, we include all these effects and the resulting Modelica model is:

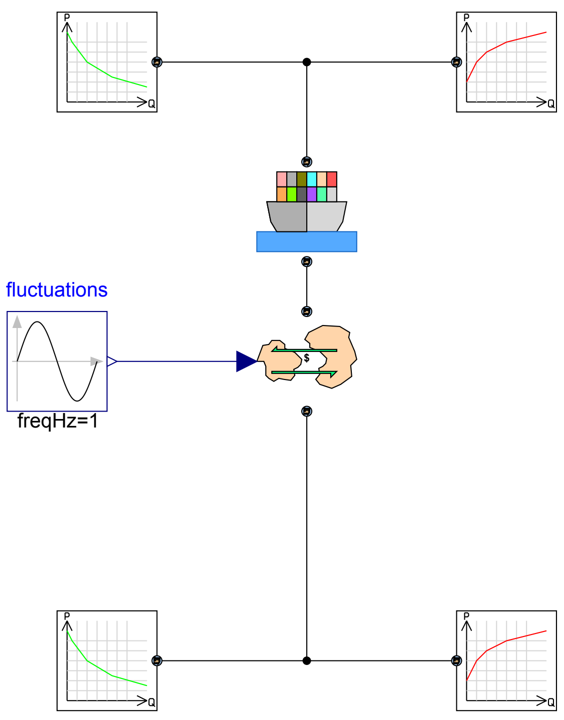


Figure 11. Model of international trade

```
model InternationalTrade
  Components.ExponentialProducer
      foreign_producer(min_price=40, growth
      =0.01);
  Components.ExponentialConsumer
      foreign_consumer(max_price=55, decay
      =0.001);
  Components.ExponentialConsumer
      domestic_consumer(max_price=80, decay
      =0.003);
  Components.ExponentialProducer
      domestic_producer(min_price=50,
      growth=0.02);
  Components.Trade trade(tariff_AB=0.05);
  Modelica.Blocks.Sources.Sine fluctuations
      (
    amplitude=0.05, freqHz=1, startTime
       =0.5,
    offset=1.2) "Fluctuation of currency
       exchange rates";
  Components.Shipping shipping(
      shipping_cost=1);
equation
  connect(foreign_consumer.market,
      foreign_producer.market);
  connect(domestic_consumer.market,
      domestic_producer.market);
  connect(trade.market_B,
      domestic_producer.market);
  connect(fluctuations.y, trade.xrate);
  connect(shipping.remote, trade.market_A);
  connect(shipping.local,
      foreign_producer.market);
```

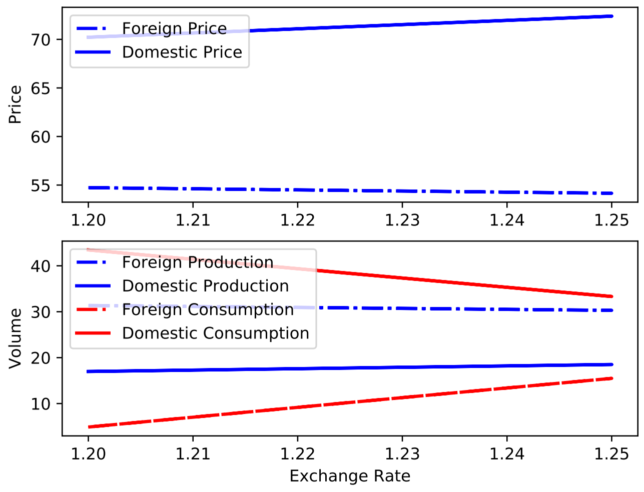


Figure 12. Price and Volume vs. Exchange Rate

```
end InternationalTrade;
```

In this case, the shipping model adds $1 to the cost of any good that moves between the markets. Similarly, the `trade` block handles the currency conversion and imposes a tariff of 5% on goods moving from the foreign market (top) to domestic market (bottom).

In this particular model, there is no `MarketAnalysis` block. Instead, the system is always in equilibrium. However, there is a time varying component to this model because the curreny rate fluctuates. This gives us a chance to see how the currency rate influences both consumers and producers in both geographies.

Figure 12 shows how consumers and producers respond to changes in exchange rate. Specifically, what we see is that as the exchange rate increases, the prices in the "domestic region" (at the bottom of Figure 11) rise. As a result, we can see that while domestic production rises only slightly and foreign production drops slightly (the blue lines in bottom plot of Figure 12), consumption of goods domestically drops while there is a corresponding rise in consumption by foreign consumers. In other words, as the exchange rate rises, the foreign produced goods become less attractive to the domestic market. Since fewer foreign goods are being shipped away from their home market, the effective demand drops and so does the price in the foreign markets which triggers foreign consumers to purchase more.

6 Future Directions

There are a number of additional effects it would have been interesting to model but time and space constraints prevented a deeper study of the topic. In particular, we did not look into market dynamics. There are no states in these models. While some conditions may change (*e.g.*, the exchange rate in the last example), the solutions are still strictly algebraic.

One could imagine adding dynamics in several ways. First, the output of producers could respond (with some lag) to increasing demand and the potential to increase revenue through higher volumes of sales (at cheaper market prices). Furthermore, we haven't examined the impact of stockpiling of resources when prices are relatively low for the purpose of reselling them when they are higher.

These models point out a few limitations in Modelica as well. The first is that we might accidentally mix different goods with an erroneous connection. It would be useful if we could somehow parameterize the various models in terms of the underlying types of goods so that a consumer of light bulbs couldn't accidentally be connected to a producer of turbine blades. It isn't clear how to formulate these models so that such mistakes could be statically detected. One approach would be to establish the type of good as a `parameter` on the `Market` connector. However, this approach would require setting these parameter values all over the place unless we adopted an approach similar to how fluid models leverage media models.

Another limitation is related to how supply and demand curves are expressed. Currently, these curves are instantiated once by each consumer and producer model. However, initial attempts to model market segmentation (*e.g.,* different classes of consumers with different price sensitivities) suggested the potential value of being able to export these curves so that other models could instantiate them for their own purposes. The closest physical analogy would be how some vehicle dynamics libraries express the elevation of a road surface such that each tire of the vehicle can independently query the road for its elevation. Such capabilities might allow the calculation of economic metrics like deadweight loss, *etc.*

Finally, more complex economic models will almost certainly require the need to express constraints and objectives along the lines of what is expressed in Modelica extensions like Optimica (Åkesson 2008). With such expressiveness a Modelica compiler may compile such economic models into general optimization problems or perhaps specialized linear programs.

7 Conclusion

In conclusion, this paper discusses how to formulate typical supply and demand curves as reusable component models. These component models can then be connected together to simulate the behavior of a market. Modelica semantics ensure that each entity in the market uses a consistent price and that all goods are properly accounted for. Additional components have be defined to model the effects of taxation, transportation, tariffs, manufacturing, *etc.*

Although the models here are not meant to be a complete or rigorous approach to modeling supply and demand systems, hopefully the discussions here will provide a reasonable starting point for further development. Furthermore, the content of this paper could assist those intereste in economic models but unfamiliar with Modelica in creating economic models in Modelica. The models described in this paper are provided under an MIT open source license and can be found at `https://github.com/mtiller/EconomicsLibrary`.

References

Åkesson, Johan (2008). "Optimica - An Extension of Modelica Supporting Dynamic Optimization". In: *Proceedings of the Modelica Conference, 2008*. URL: `https://www.modelica.org/events/modelica2008/Proceedings/sessions/session1b3.pdf`.

Casella, Francesco, Giovanni Miragliotta, and Luigi Uglietti (2005). "Analysis of Supply Chain Dynamics through Object Oriented Simulation". In: DOI: `https://doi.org/10.1007/3-7908-1636-1_30`.

Hutchinson, Emma (2016). *Principles of Microeconomics*. OpenStax College. URL: `https://pressbooks.bccampus.ca/uvicecon103/`.

Modelica Association (2017). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.4*. Tech. rep. Linköping: Modelica Association. URL: `https://www.modelica.org/documents/ModelicaSpec34.pdf`.

Posner, Barry and Farid Tayari (2018). *Introduction to Energy and Earth Sciences Economics*. Penn State. URL: `https://www.e-education.psu.edu/ebf200`.

Taylor, Timothy (2018). *Principles of Economics*. OpenStax College. DOI: `10.24926/8668.1601`. URL: `https://doi.org/10.24926/8668.1601`.

Zimmer, Dirk and Daniel Schlabe (2012). "Implementation of a Modelica Library for Energy Management based on Economic Models". In: *Proceedings of the 9th International Modelica Conference*, pp. 133–142. DOI: `10.3384/ecp12076133`. URL: `http://www.ep.liu.se/ecp/076/012/ecp12076012.pdf`.

Modelica Modelling of an Ammonia Stripper

J.A. Redford[1] A.B. Bisinella De Faria[2] J.-P. Saut[1] J. Robert[2] M. Albuquerque[2] J.-P. Merland[2] J.-M. Ghidaglia[3]

[1]Eurobios SCB, 61 av. du Président Wilson, Cachan, France. jredford@eurobios.com
[2]Veolia Recherche & Innovation, Chemin de la Digue, Maisons-Laffitte, France.
[3]CMLA, ENS Paris Saclay, 61 av. du Président Wilson, Cachan, France.

Abstract

This work presents a Modelica model for an ammonia stripper that is used to process waste (digestate) from a biogas production unit. The model includes the chemical balance equations between species in the liquid and gas, and includes the exchanges between both phases and the energy consumption of the unit. Results show the expected behaviour with an increasing pH with time, with further validation and calibration being necessary once experimental results are available. This is a novel use of Modelica designed to expand the library of processes that are simulated using this approach.

Keywords: nutrients recovery, chemical reactions, process engineering, environment

1 Introduction

Today, the balance between the requirement for a resource and its availability has changed. This has led to a paradigm shift from resource consumption that is followed by a waste production to a use-and-recover approach, which converts the waste into a product. However, in order to reduce environmental impact and increase nutrient recycling there is a growing demand for predictive tools that might help to better manage these processes and close the recycling loop.

In this context, anaerobic digestion has been studied over the last decades and is known today as a process that allows recovery of large amounts of energy by producing biogas. However, the residual flow of the digested product, known as a digestate, needs to be valorized in order to fully capitalize on the nutrient recycling, in addition to the conventional energetic valorization. The digestate has, among other nutrients, large amounts of nitrogen that can be used as fertilizers on agricultural land. Traditionally, the residual nitrogen is treated (by nitrification and denitrification for instance) without the aim of recovering the nitrogen. However, as resource recovery is gaining interest, several technologies are available today that valorize the nitrogen in the digestate, in order to produce an environmental, legislative and cost friendly byproducts.

Among the most used technologies, one may cite ammonia stripping, struvite precipitation and membrane processes. Struvite ($NH_4MgPO_4 \cdot 6H_2O$) is a slow-release fertilizer, however its nitrogen concentration is limited and the precipitation is triggered by magnesium, which demands the addition of chemical products. Concerning membrane processes, pretreatment techniques are required in order to increase the membrane lifetime. Therefore, ammonia stripping is a low cost technique that allows recovery of nitrogen from a liquid phase to a gas phase, and then via an acid scrubber it is possible to recover a nitrogen salt, such as the ammonium sulphate salt $(NH_4)_2SO_4$.

The objective of this study is to develop a model capable of predicting nitrogen removal from a digestate and how this is impacted by variation of the main influencing operational parameters. When calibrated, this model can be used to predict stripping performance, envisage other process configurations, optimize recovery and estimate operational costs. In order to be easily used in different contexts, the developed model should include the possibility to simulate both mass and energy balances, along with the kinetics of the physico-chemical and liquid to gas transfer reactions involved in the process. It is also important to have the capacity for expansion in order to allow future enrichment by experimental results, as well as the use of the module, both in-situ and in a larger context of plant-wide modelling approach.

1.1 Process engineering using Modelica

The use of Modelica in chemical process engineering is developing quickly. While Jain et al. (2017) state that 'OpenModelica … lacks good chemical engineering support', much work is under way to correct this. Marx-Schubach and Schmitz (2017) have created a library and model for an absorber used in a carbon capture process. Åkesson et al. (2011) model carbon capture, which involves stripping. They include chemistry for a system of equations along with calculation of equilibrium equations. Comparison is made with experimental results. Baharev and Neumaier (2012) aim to create the foundations of a general purpose library for chemical process modelling. Joos et al. (2009) model several chemical processes (absorption, adsorption and rectification). Küssel et al. (2009) model rotary kilns (i.e. combustion) and make comparisons with computational fluid dynamics (CFD) simulations. Both approaches showing good correspondence, but Modelica is less computationally demanding, which means that the authors' aims for real time operation in control loop is realistic. Cellier and Greifeneder (2009) carried out simulations of basic reactions to demonstrate the use of Modelica for simulation of chemical reaction systems in an object-oriented physically inspired manner.

Windahl et al. (2015) develop a library of thermodynamic properties and give an air separation column with multiple stages as a test case. CAP-OPEN (Computer-Aided Process Engineering-OPEN) is used and there is strong interest in the interface structure linking Modelica and external libraries. Tummescheit and Eborn (2002) add support to 'ThermoFluid' for chemical reactions and membrane diffusion.

De Canete et al. (2013) use Modelica for simulation of distillation columns. They have accounted for the column trays with multiple stacked trays. This work is aimed at developing a control system with an Neurofuzzy network approach.

Concerning ammonia stripping, Vaneeckhaute et al. (2018) recently published a generic nutrient recovery model (NRM) library that is mainly coded in Modelica and based on detailed chemical solution speciation and reaction kinetics for nitrogen recovery.

2 The Model

A simplified model for an ammonia bubble stripper is now presented. Figure 1 shows the basic configuration of the simplified stripper along with the inputs, outputs and the processes under consideration. Digestate enters the stripper with a given flowrate and initial composition. Air is added at the base of the stripper with a given temperature and relative humidity. At the top, the result is a stripped gas containing air, water and ammonia. The liquid output is the digestate with reduced TAN (total ammonia nitrogen). Heat is lost from the liquid through warming of the gas and the liquid to gas water mass transfer (evaporation). Hence, heating is required to maintain the digestate at the correct temperature and this represents one of the running costs of the stripper unit.

With respect to the gas phase, the stripper is split into slices that are stacked vertically. The liquid phase is assumed to be fully mixed and is represented as a single entity. As the gas bubbles rise they will gradually strip the ammonia from the liquid phase.

2.1 Chemical equations

In Figure 2 the model inputs are in red and the outputs are in blue. The internal processes are in green, where the mass balances include $NH_{3(aq)}$, NH_4^+ , HCO_3^- , CO_3^{2-} , $CO_{2(aq)}$, N_{org} , H_2O , H^+ and OH^- species for the liquid phase, NH_3, $CO_2$and H_2O species in the gas phase. The energy balance includes water vaporisation and air heating. In order to model this process we must consider the chemical equilibrium between the TAN and DIC (dissolved inorganic carbon) species, the mass transfer of NH_3, $CO_2$and H_2O between liquid and gas, and an energy balance. The dynamics in the column depend on time (for both liquid and gas concentrations) and column height (gas concentration), meaning that the stripping column has to be discretised spatially and temporally.

2.1.1 Modelling pH and species concentrations

In the liquid phase three conjugate acid-base pairs are considered here; $NH_{3(aq)}$ /NH_4^+ , HCO_3^- /CO_3^{2-} and HCO_3^- /$CO_{2(aq)}$. Initially focusing on nitrogen species, two reactions might be written as follows

$$NH_3 + H^+ \rightarrow NH_4^+ \,, \tag{1}$$

with reaction rate k_a (kinetic constant of association), and

$$NH_4^+ \rightarrow NH_3 + H^+ \,, \tag{2}$$

with a reaction rate k_d (kinetic constant of dissociation). Therefore, for the association, it is possible to relate the species concentrations with

$$\frac{d[NH_4^+]}{dt} = k_a[NH_3][H+] \,, \tag{3}$$

and for the dissociation

$$\frac{-d[NH_4^+]}{dt} = k_d[NH_4^+] \,. \tag{4}$$

The absolute association rate equals dissociation, thus

$$k_a[NH_3][H^+] = k_d[NH_4^+] \,, \tag{5}$$

$$[NH_3][H^+]/[NH_4^+] = \frac{k_d}{k_a} = K_a \,, \tag{6}$$

where K_a is the acid dissociation constant. Similarly, equations for chemical equilibrium might be arranged for NH_4^+ , for example, as "Association - Dissociation", giving

$$\text{Chemical rate}_{NH_4^+} = -k_d[NH_4^+] + (k_a[NH_3][H^+]) \,. \tag{7}$$

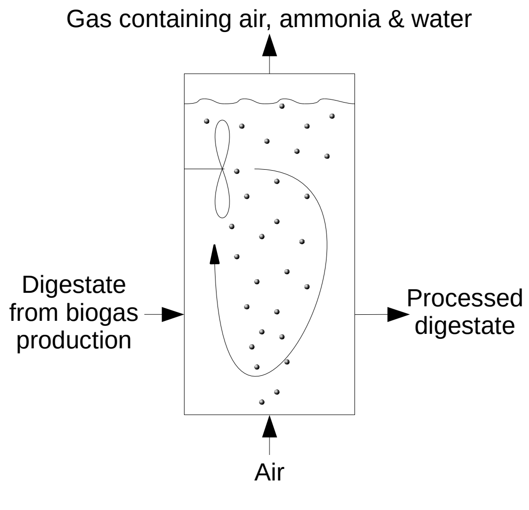

Gas containing air, ammonia & water
Digestate from biogas production
Processed digestate
Air

Figure 1. Stripping column exchanges.

This same approach can be applied to the other chemical reactions. Values for K_a are found from the LLNL[1] database in the form of $-\log K_a$. These values might also be expressed as pK_a, which represents $-\log K_a$. (Note that, $-\log[\mathrm{H}^+] = \mathrm{p}H$.)

Using the pK_a and pH definition equations, the liquid NH_3 fraction, which will be partially transferred to the gas in the transfer step, can be calculated using

$$f_{\mathrm{NH_3}} = \frac{10^{\mathrm{p}H - \mathrm{p}K_{a,\mathrm{NH_4^+/NH_3}}}}{10^{\mathrm{p}H - \mathrm{p}K_{a,\mathrm{NH_4^+/NH_3}}} + 1} . \tag{8}$$

Therefore, from TAN, the ammonia concentration in liquid is given by

$$C_{\mathrm{NH_3}} = f_{\mathrm{NH_3}} \mathrm{TAN} , \tag{9}$$

and by definition,

$$f_{\mathrm{NH_4^+}} = 1 - f_{\mathrm{NH_3}} . \tag{10}$$

The same classical approach can be applied to find the DIC composition, which will not be given here to save space. The values of k_a and k_d depend on the kinetics and can be obtained experimentally. However, the kinetic rate constants are fixed at high values to ensure that the species present in the system almost instantaneously reach chemical equilibrium. This approach was proposed by Lizarralde et al. (2015).

2.1.2 Gas transfer

At time t the liquid concentrations will be constant across all the column slices, i.e. completely mixed, and the gas concentration will change with height/slice. As with the real process, equilibrium between aqueous and bubble gaseous phases is not reached by top of column, meaning that the bubbles will leave the stripper without being saturated in NH_3. The mass transfer from the liquid to each gas slice is calculated using the mass transfer rate described in Matter-Müller et al. (1981). For NH_3

$$M_{\mathrm{NH_3,liquid\ to\ gas}} = k_{L,\mathrm{NH_3}} a V_L (C_{\mathrm{NH_3,liq}} - C^*_{\mathrm{NH_3}}) . \tag{11}$$

The gas bubbles are not perfectly spherical, and they will also coalesce and turbulence will cause them to break up. So, it is not possible to calculate the transfer area a. Also, a real digestate contains a significant concentration of suspended solids and other ionic species, which makes deriving a value of mass transfer coefficient k_L difficult. Instead, a global value of $k_L a_{\mathrm{NH_3}}$ is found experimentally, hence

$$M_{\mathrm{NH_3,liquid\ to\ gas}} = k_L a_{\mathrm{NH_3}} V_L (C_{\mathrm{NH_3,liq}} - C_{\mathrm{NH3^*}}) . \tag{12}$$

Considering liquid gas transfer, Henry's law describes relationship between gas concentration (expressed by its partial pressure in atm) and the saturation liquid concentration C^*_{liq}. For instance, in the case of NH_3

$$\begin{aligned} &\mathrm{NH_{3(\mathit{aq})}} \leftrightarrow \mathrm{NH_{3(\mathit{g})}} \text{ with} \\ K_{e,\mathrm{NH_3(25°C)}} &= [\mathrm{NH_{3(\mathit{g})}}]/[\mathrm{NH_{3(\mathit{aq})}}] \\ &= 0.016\,\mathrm{atm/(mol/kg)} , \end{aligned} \tag{13}$$

with $[\mathrm{NH_{3(\mathit{g})}}]$ expressed as a partial pressure of NH_3.

Also, from Matter-Müller et al. (1981)

$$C^*_{\mathrm{liq}} = C_g / H^{cc} , \tag{14}$$

where H^{cc} is Henry's dimensionless constant. $1/H^{bp}$ values are available in atm/(mol/kg) units and conversion to a dimensionless constant can be obtained using

$$H^{bp} \approx H^{cp} / \rho_{\mathrm{H_2O}} , \tag{15}$$

$$H^{cc} = H^{cp} R T , \tag{16}$$

where $\rho_{\mathrm{H_2O}}$ is the solvent density (kg/m^3), R is the universal gas constant (8.206×10^5 m^3 atm/(K mol), T is the temperature (K) and H^{cp} is given in mol/(m^3 atm).

The mass balance equations for all species in both liquid and gas phases can be found from chemical equilibria and liquid/gas transfer equations. For any liquid species, HA, present in the column and supposing that $\mathrm{HA} \leftrightarrow \mathrm{H^+} + \mathrm{A^-}$, the mass balance can be written as

$$\begin{aligned} \frac{dM_{\mathrm{HA,liquid}}}{dt} &= m_{\mathrm{HA,in,dig}} - m_{\mathrm{HA,out,dig}} \\ &+ k_{a,\mathrm{HA}} \Big(K_a C_{\mathrm{HA}}(t) - C_{\mathrm{H^+}}(t) C_{\mathrm{A^-}}(t) \Big) V_L \\ &- \int m_{\mathrm{HA},gas} dh . \end{aligned} \tag{17}$$

The last term is neglected if species does not have liquid/-gas transfer and it is the integrated amount of the transferred species over the column height that needs to be considered in the liquid mass balance.

[1] Lawrence Livermore National Laboratory

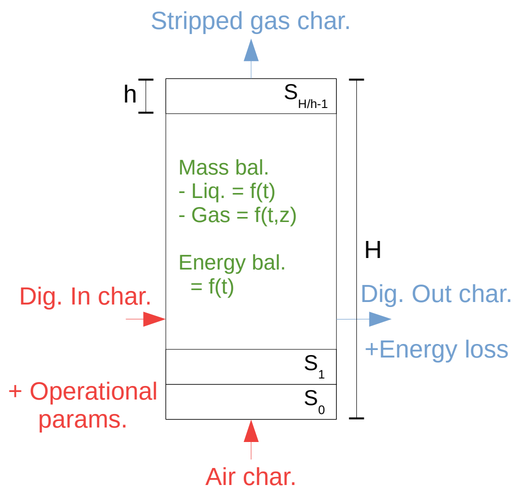


Figure 2. Stripping column schematic.

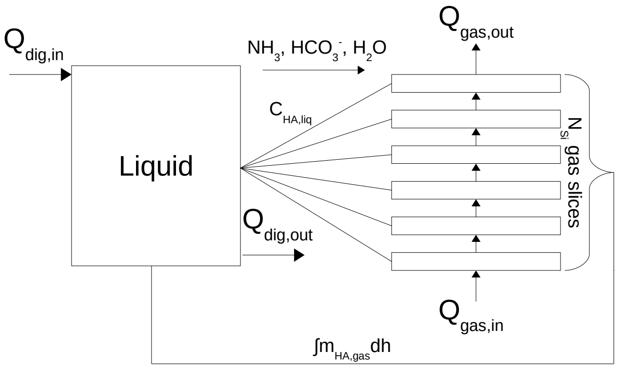


Figure 3. Proposed model structure

For HA gas species, the mass balance for any slice S_i is

$$\frac{dM_{\text{HA,gas}}}{dt} = m_{\text{HA,in,gas}}|_{S_i} - m_{\text{HA, out, gas}}|_{S_i} + (k_L a_S (C_{\text{HA,liq}}(t) - C^*_{\text{HA}}(t,h)) V_{si}) \,. \tag{18}$$

For slice S_i, the volume tends to zero (sufficiently small slices), thus the previous equation becomes

$$\frac{dM_{\text{HA,gas}}}{dt} = - dM_{\text{HA}}(t,h) + (k_L a_S (C_{\text{HA,liq}}(t) - C^*_{\text{HA}}(t,h)) A dh) \,. \tag{19}$$

2.1.3 Energy balances

A simple approach is used for the energy balance calculation. The input air flow is considered to change quickly enough to the temperature of the column that it is instantaneous. The gas will also be instantaneously saturated by water (with respect to the new gas temperature) and thus, the mass of evaporated water can be calculated for the whole column.

Finally, the energy lost by the column when evaporating the water (in the output gas) can be calculated using the water vaporization enthalpy and the energy necessary to heat the air input from ambient temperature to the column temperature.

3 Construction of Modelica model

3.1 Compilation of model equations

Figure 3 shows how the liquid and gas are to be linked in the model. The NH_3 species calculation progresses as follows. Given the liquid pH, pK_a and TAN, f_{NH_3}, $f_{NH_4^+}$ and then C_{NH_3}, $C_{NH_4^+}$ can be calculated from (8), (9) and (10).

The initial concentrations are calculated using

$$[\text{NH}_{3(\text{aq})}]_0 = f_{\text{NH}_3} \text{TAN}_0 / MM_{\text{TAN}} \tag{20}$$
$$[\text{NH}_4]_0 = f_{\text{NH}_4} \text{TAN}_0 / MM_{\text{TAN}} \tag{21}$$
$$N_{\text{org}} = \text{TKN} - \text{TAN} \tag{22}$$
$$[\text{HCO}_3^-]_0 = f_{\text{HCO}_3^-} \text{DIC}_0 \tag{23}$$
$$[\text{CO}_3^{2-}]_0 = f_{\text{CO}_3^{2-}} \text{DIC}_0 \tag{24}$$
$$[\text{CO}_{2(\text{aq})}]_0 = f_{\text{CO2(aq)}} \text{DIC}_0 \tag{25}$$

where mass fractions are found from equations (8), (10) and similar for DIC, and the molecular mass of TAN is 14 g/mol. TKN is Total Kjeldahl Nitrogen. DIC_0 is taken as 360 mol/m^3. The initial concentrations of hydrogen ion and hydroxide are

$$[\text{H}^+]_0 = 10^{-\text{p}H_0} \tag{26}$$
$$[\text{OH}^-]_0 = K_{a,\text{OH}^-/\text{H}^+} / [\text{H}^+]_0 \tag{27}$$

This step is only needed to find the initial concentration of each species, and then the time dependent concentrations are found by resolution of the system of conservation equations. The concentration of H^+ is known during simulation meaning that pH is also known.

Equations (17) and (19) are solved for liquid and gas, respectively. The digestate inflow is assumed to be the same as the initial composition; pH_0, TAN_0, DIC_0, TKN_0. Then we can calculate

$$m_{\text{NH}_3,\text{in,dig}} = Q_{\text{dig,in}} C_{\text{NH}_3,0} \,, \tag{28}$$

while the digestate outflow has the same composition as the tank, hence

$$m_{\text{NH,out,dig}} = Q_{\text{dig,out}} C_{\text{NH}_3}(t) \,. \tag{29}$$

$Q_{\text{dig,out}}$ is the inflow minus the amount of evaporated water, which can be calculated independently of other equations as proposed at the end of this section

$$Q_{\text{dig,out}} = Q_{\text{dig,in}} - m_{\text{H}_2\text{O}} / \rho \text{H}_2\text{O}_{(\text{aq})} \,. \tag{30}$$

In equation (17), mass transfers resulting from the change in concentration caused by changes in equilibrium are in the term

$$k_a (K_a C_{\text{NH}_4^+}(t) - (C_{\text{H}^+}(t) C_{\text{NH}_3}) V_L) \,. \tag{31}$$

We need to make the gas calculation to find $\int m_{\text{HA,gas}} dh$ using equation (19) for each slice (note that it is easier to use equation (18) as the slices have a finite thickness, i.e. not tending towards zero). For each slice

$$\frac{dM_{\text{NH}_3,\text{gas}}}{dt} = m_{\text{NH}_3,\text{in,gas}}|_{S_i} - m_{\text{NH}_3,\text{out,gas}}|_{S_i} + m_{\text{NH}_3,\text{lg}}|_{S_i} \,, \tag{32}$$

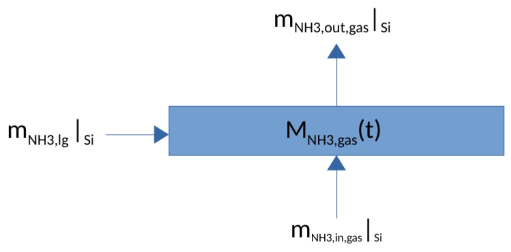


Figure 4. Mass transfers in and out of each gas slice.

where the liquid to gas transfer is $m_{\mathrm{NH_3,lg}}|_{S_i} = k_L a_{\mathrm{NH_3}}(C_{\mathrm{NH_3,liq}}(t) - C^*_{\mathrm{NH_3}}(t)|_{S_i})V_{S_i}$ and the mass transfers in and out of each gas slice are summarized in Figure 4. The bottom slice has

$$m_{\mathrm{NH_3,\,in,\,gas}} = Q_{\mathrm{air}} C_{\mathrm{NH_3,\,gas},N_0} = 0\,, \tag{33}$$

and for subsequent slices, the inter-slice mass flow rate for the NH_3 species is

$$m_{\mathrm{NH_3,\,in,\,gas}}|_{S_i} = m_{\mathrm{NH_3,\,out,\,gas}}|_{S_i - 1}\,. \tag{34}$$

The liquid concentration $C_{\mathrm{NH_3,\,liq}}$ is known from earlier calculations and the saturation concentration is calculated from

$$C^*_{\mathrm{NH_3}} = C_{\mathrm{NH_3,\,gas}}|_{S_i} / H^{cc}_{\mathrm{NH_3}}\,, \tag{35}$$

where $H^{cc}_{\mathrm{NH_3}} = \rho_{\mathrm{H_2O}} H^{bp}_{\mathrm{NH_3}} R T_L$ and $\rho_{\mathrm{H_2O}} = 103$ kg/m^3, $R = 8.206 \times 10^{-5}$ m^3 atm / (K mol) and $1/H^{bp}_{\mathrm{NH_3}} = 0.016$ atm/(mol/kg). The liquid temperature T_L is chosen by the user, where 45°C is used in the demonstration shown later.

In order to calculate $m_{\mathrm{NH_3,out,gas}}|_{S_i}$ in equation (32), the gas passing out of the slice is taken to have the same concentration as found in the slice, which is possible because the slices are sufficiently small. Hence $C_{\mathrm{g},S_i} = M_{\mathrm{NH_3,gas}} / (\rho_{\mathrm{air}} V_{S_i})$ and we can then say

$$m_{\mathrm{NH3,out,gas}}|_{S_i} = C_{\mathrm{g},S_i} \rho_{\mathrm{air}} Q_{\mathrm{air}}\,. \tag{36}$$

The system is solved simultaneously for t and h, and for each dM/dt (or VdC/dt), where the only unknown variable in each equation is the concentration of the component itself or another component that also has its own dM/dt. Thus the number of equations and unknowns are the same, meaning that Modelica can now be used to find $M_{\mathrm{NH3,gas}}(t)$ for each slice, for a given initial $M_{\mathrm{NH3,gas}}(0)$.

The total mass transfer of NH_3 from liquid to gas is calculated by summing across the slices

$$\int m_{\mathrm{HA,gas}} dh = \sum_{S_i} m_{\mathrm{NH_3,lg}}|_{S_i}\,, \tag{37}$$

which can then be used in (17). Equation (17) will apply to NH_3, CO_3^{2-}, HCO_3^- H_2O, H^+, NH_4^+, N_{org}, and so on. The calculation given above applies to reactions including HCO_3^-, while all other HA species are the same apart from not having the liquid to gas mass transfer term at the end of (17).

The liquid species equations are too large to be reproduced here, but they have the form

$$\begin{aligned}\frac{dM_{\mathrm{HA}}}{dt} &= m_{\mathrm{HA,in}} - m_{\mathrm{HA,out}} \\ &+ k_{a,\mathrm{HB/HA}}\left(K_{a,\mathrm{HB/HA}}[\mathrm{HB}] - [\mathrm{H^+}][\mathrm{HA}]\right)V_{\mathrm{L}} \\ &- \int m_{\mathrm{HA,gas}} dh\,,\end{aligned} \tag{38}$$

with the last term only included if there is liquid to gas mass transfer.

In the case of N_{org} the reaction only goes in one direction (to NH_4^+) and is controlled by $k_{ammonification}$ (it should be easy to add saturation/inhibition coefficients later on). $k_{ammonification}$ has to be calibrated later when experimental data is available, but it can be set, for the moment, as a low value.

It is assumed that $C_{\mathrm{H_2O}} \to 1$ because water is the solvent, and so it will not appear in the reaction equations (except for water evaporation). Values for pK_a, and K_a are given by the formulas as a function of temperature.

The equations used for the gas phases of CO_2 and NH_3 are then calculated. The calculations of C^* for the gas phase species in a slice are made using

$$C^*_{\mathrm{NH_3g}} = C_{\mathrm{NH_3g}} / (\rho_{\mathrm{H_2O}} R H^{bp}_{\mathrm{NH_3g}} T_L)\,, \tag{39}$$

$$C^*_{\mathrm{CO_2g}} = C_{\mathrm{CO_2g}} / (\rho_{\mathrm{H_2O}} R H^{bp}_{\mathrm{CO2g}} T_L)\,. \tag{40}$$

with the mass transfer rate to each slice from the liquid

$$(m_{\mathrm{lg,NH3g}})_{\mathrm{sl}} = k_L a_{\mathrm{NH_3}} V_{\mathrm{L,sl}} (C_{\mathrm{NH_3l}} - C^*_{\mathrm{NH_3g}})\,, \tag{41}$$

$$(m_{\mathrm{lg,CO2g}})_{\mathrm{sl}} = k_L a_{\mathrm{CO_2}} V_{\mathrm{L,sl}} (C_{\mathrm{HCO_3}} - C^*_{\mathrm{CO_2g}})\,, \tag{42}$$

and note the use of $C_{\mathrm{HCO_3}}$ in (42).

We must calculate concentrations in the gas and for that it is necessary to have a gas volume per slice as we are working with the amount of substance (moles), hence

$$C_{\mathrm{NH_3g}} = \frac{M_{\mathrm{NH_3g}}}{\rho_{\mathrm{air}} V_{\mathrm{L,sl}} \alpha_{\mathrm{gas}}}\,, \tag{43}$$

$$C_{\mathrm{CO_2g}} = \frac{M_{\mathrm{CO_2g}}}{\rho_{\mathrm{air}} V_{\mathrm{L,sl}} \alpha_{\mathrm{gas}}}\,. \tag{44}$$

Note, α_{gas} is the gas volume fraction of liquid, which is needed if we would like to know the concentration relative to gas volume (moles of NH_{3g} / m^3 of gas).

The out-flowing molecular masses from a slice for each species are

$$(m_{\mathrm{NH_3g,out}})_{sl} = (C_{\mathrm{NH_3g}})_{\mathrm{sl}} m_{\mathrm{air}}, \tag{45}$$

$$(m_{\mathrm{CO_2g,out}})_{sl} = (C_{\mathrm{CO_2g}})_{\mathrm{sl}} m_{\mathrm{air}}, \tag{46}$$

where the slices are connected by making

$$(m_{NH_3g,in})_{sl} = (m_{\mathrm{NH_3g,out}})_{\mathrm{sl}} - 1, \tag{47}$$

$$(m_{CO_2g,in})_{sl} = (m_{\mathrm{CO_2g,out}})_{\mathrm{sl}} - 1. \tag{48}$$

Hence, the rate of change in moles of substance for each gas species in a slice is

$$\frac{dM_{\mathrm{NH_3g}}}{dt} = m_{\mathrm{lg,NH_3g}} + m_{\mathrm{NH_3g,in}} - m_{\mathrm{NH_3g,out}}\,, \tag{49}$$

$$\frac{dM_{\mathrm{CO_2g}}}{dt} = m_{\mathrm{lg,CO_2g}} + m_{\mathrm{CO_2g,in}} - m_{\mathrm{CO_2g,out}}\,, \tag{50}$$

assuming that the H_2O immediately saturates the gas at the gas inflow temperature (ambient temperature). The gas will first be heated to achieve the column temperature and then an amount of water will be evaporated.

The rate of water mass evaporation can be calculated with

$$m_{\mathrm{H_2O}} = Q_{\mathrm{air\,25^\circ C}}\rho_{\mathrm{air\,25^\circ C}}(X_T - X_{25^\circ C}) \tag{51}$$

where m_{H_2O} is the rate of evaporation in kg water/h, $Q_{air25^\circ C}$ is the volumetric flowrate of air at 25°C in m^3/h, $\rho_{air25^\circ C}$ is the density of air at 25°C in kg/m^3, X_T is the absolute humidity of water at the working liquid temperature T_L in kg water/kg "dry" air, $X_{25^\circ C}$ is the absolute humidity of water at the inlet air temperature in kg water/kg "dry" air.

The saturation pressure (in Pa) depends only on the air temperature (in °C) and is calculated using the Magnus formula

$$p^*_{\mathrm{H_2O}} = 611.23 \exp\left(\frac{17.5043T}{T + 242.2}\right), \tag{52}$$

where * denotes saturation. Partial pressure is calculated using $p_{\mathrm{H_2O}} = RH p^*_{\mathrm{H_2O}}$, where RH is the relative humidity. The effect of humidity is seen in equations (53) and (51); in the case of X_T we have 100% RH because the air is instantaneously saturated, and for $X_{25^\circ C}$ we have 60% RH.

The absolute humidity in "dry" air X (kg water/kg "dry" air) is found with

$$X = \frac{p_{\mathrm{H_2O}}}{P_{\mathrm{tot}} - p_{\mathrm{H_2O}}}\frac{18}{28.84}, \tag{53}$$

and the air density depends only on temperature (in °C) and can be calculated with

$$\rho_{air} = 1.292\frac{273.25}{273.15 + T}\frac{p_{\mathrm{H_2O}}}{P_{\mathrm{tot}}}, \tag{54}$$

The rate of energy consumed by the system for water evaporation, in kJ/h, is then calculated by

$$F_{\mathrm{evap}} = m_{\mathrm{H_2O}}\Delta H_{\mathrm{vap\ water}}\,, \tag{55}$$

where $\Delta H_{\mathrm{vap\ water}}$ is the enthalpy of vaporization of water calculated using the correlation

$$\Delta H_{\mathrm{vap\ water}} = \frac{-0.0439T_L + 45.084}{0.018}. \tag{56}$$

Note, this equation has T_L in °C and ΔH with units J/kg. The air temperature is raised to the liquid temperature, meaning that

$$F_{\mathrm{air}} = Q_{\mathrm{air}}C_{p,\mathrm{air}}\rho_{\mathrm{air}}(T_L - T_G)\,. \tag{57}$$

The temperature of the bubble stripper is maintained by a heater and $F_{\mathrm{evap}} + F_{\mathrm{air}}$ is the power consumed.

3.2 Modelica implementation

The stripper model contains the definition of several smaller submodels (Figure 5). Liquid, gas and evaporated water have their own models. The gas phase is spatially 1D-discretised, as shown for the model in Section 2. This discretisation is achieved through the use of a 1D array of *Slice* models, with each slice being an elementary volume cell. Figure 5 shows the different connectors used to link the submodels.

The model is meant to be one component in a larger set or library, so the limit conditions are not defined in the stripper component, but are instead defined in external components (Figure 6).

Species are declared using the enumeration class type, e.g. species in the liquid phase

```
type LSpecies = enumeration(NH3l, NH4,
    Norg, HCO3, CO32, CO2l, Hp, OH);
constant LSpecies NH3l = LSpecies.NH3l;
constant LSpecies NH4  = LSpecies.NH4;
constant LSpecies Norg = LSpecies.Norg;
constant LSpecies HCO3 = LSpecies.HCO3;
constant LSpecies CO32 = LSpecies.CO32;
constant LSpecies CO2l = LSpecies.CO2l;
constant LSpecies Hp   = LSpecies.Hp;
constant LSpecies OH   = LSpecies.OH;
```

Then balance equations, such as

$$\begin{aligned}\frac{dM_{\mathrm{NH_3,liquid}}}{dt} &= m_{\mathrm{NH_3},in,dig} - m_{\mathrm{NH_3},out,dig} \\ &+ k_{a,\mathrm{NH_4}}\left(K_a C_{\mathrm{NH_4}}(t) - C_{\mathrm{H^+}}(t)C_{\mathrm{NH_3}}(t)\right)V_L \\ &- \int m_{\mathrm{NH_3},gas}dh\,,\end{aligned} \tag{58}$$

(see Eq. (17)), are easily written in Modelica code as

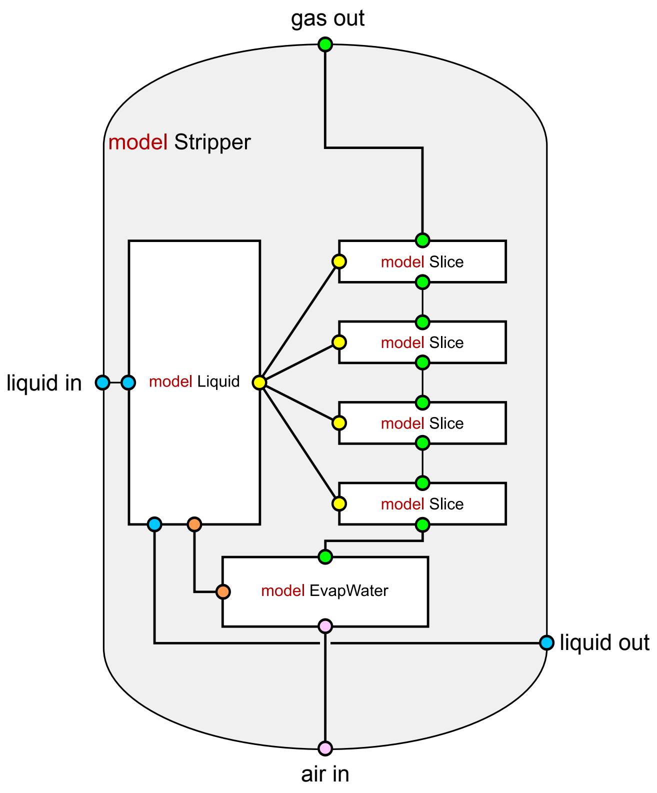
```
der(Ml[NH3l]) = mdig[NH3l] + mdigout[NH3l]
  + kaNH4 * (KaNH4(liqGas.TL) * CL[NH4]
    - CL[Hp] * CL[NH3l]) * VL +
    liqGas.mlgtot[NH3lg];
```

given that we have previously defined[2]

```
AmountSubstance Ml[LSpecies] "molar mass
    of each liquid species";
SI.MolarFlowRate mdig[LSpecies] "molar
    mass inflow rate of digestate of each
     species";
SI.MolarFlowRate mdigout[LSpecies] "molar
     mass outflow rate of digestate of
    each species";
AcidAssConst kaNH4 "acid association
    constant";
SI.Concentration CL[LSpecies] "liquid
    concentration of each species";
SI.Volume VL "liquid volume";
```

with function `KaNH4` and a connector `LiqGas` of type `LiquidGas` (Figure 5). Note that there are differences

[2]Units prefixed with SI are from package Modelica.SIunits. The other ones are custom units.



Connector type	Variables	Species
Port 🟢	SI.Pressure p flow VolumeFlowRate Qair SI.Concentration CG[2]	NH_{3_g} CO_{2_g}
Liquid+Gas 🟡	SI.Temperature TL SI.Volume VL SI.Concentration CLG[2] flow SI.MassFlowRate mlgtot[2] "liquid to gas mass transfer rate";	$NH_{3_{\ell,g}}$ $HCO^-_{3_{\ell,g}}$
Liquid 🔵	SI.Pressure p flow VolumeFlowRateDisph Qdig SI.Concentration CL[8] SI.Temperature TL	NH_{3_ℓ} CO^{2-}_3 NH_4 CO_{2_ℓ} N_{org} H^+ HCO^-_3 OH^-
Air 🟣	SI.Pressure p flow VolumeFlowRateDisph Qair RelativeHumidity RH SI.Temperature T	
$H_2$0Gas 🟠	SI.Temperature TL SI.Pressure ptot flow SI.MassFlowRate mH2Oloss "rate of H2O mass lost from liquid to gas";	H_2O_g

Figure 5. Stripper hierarchical model as implemented in Modelica. The stack of slices defines the spatial discretisation of the gas phase. The number of slices can be varied, with four being represented in the this figure. The connections between the slices are created automatically inside a `for` loop. The table lists the different connectors used in the stripper model, with their variables. Some of the variables are arrays (e.g. *concentration* in *Port*). Each element of an array is related to a chemical species in the right-hand column of the table.

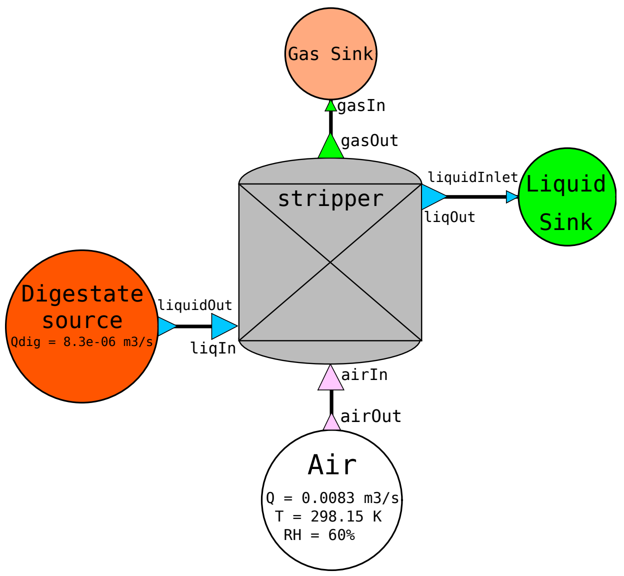


Figure 6. Modelica stripper model connected to other models to specify limit conditions.

in the signs before the flow rate variables. In Modelica models, the flow variables are signed[3] so that the outflow variables are added in the Modelica equations.

4 Results

The trend for the change in p*H* found by the simulation (Figure 7) is similar to that observed in experiment, but time scales differ. It will be necessary to make careful studies to find good values for all parameters before reliable predictions can be made.

5 Conclusion

A simple model for an ammonia stripper has been created using Modelica, which is a further demonstration of the use of Modelica in process engineering applications for problems including chemical reactions. The correct trend has been shown by results with increasing p*H* with time. There is now a need for experimental data to further validate and calibrate the model. Depending on the comparison with experiment, improvement can be made by considering the high ionic strength of the digestate, where the concentration needs to be corrected using activity coefficients. Compressor and pump calculations could also be

[3]Positive if the corresponding entity enters the component, negative if it leaves.

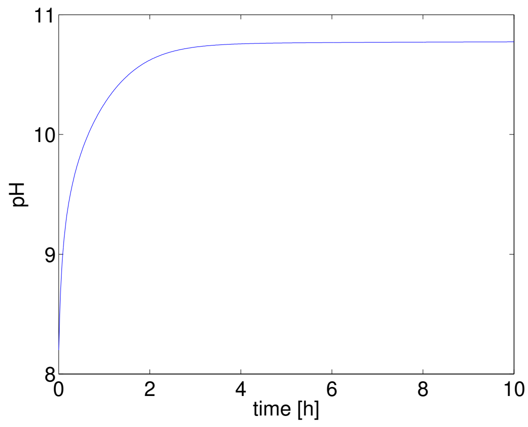


Figure 7. Change in pH with time.

made.

References

Johan Åkesson, R Faber, CD Laird, K Prölss, H Tummescheit, S Velut, and Y Zhu. Models of a post-combustion absorption unit for simulation; optimization and non-linear model predictive control schemes. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*, number 063, pages 64–74. Linköping University Electronic Press, 2011.

Ali Baharev and Arnold Neumaier. Chemical process modeling in modelica. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 76, pages 955–962. Linköping University Electronic Press, 2012.

François E Cellier and Jürgen Greifeneder. Modeling chemical reactions in modelica by use of chemo-bonds. In *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, number 043, pages 142–150. Linköping University Electronic Press, 2009.

J Fernandez De Canete, Alfonso Garcia-Cerezo, Inmaculada García-Moral, P Del Saz, and Ernesto Ochoa. Object-oriented approach applied to anfis modeling and control of a distillation column. *Expert Systems with Applications*, 40 (14):5648–5660, 2013.

Rahul Jain, Kannan M Moudgalya, Peter Fritzson, and Adrian Pop. Development of a thermodynamic engine in openmodelica. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, number 132, pages 89–99. Linköping University Electronic Press, 2017.

Andreas Joos, Karin Dietl, and Gerhard Schmitz. Thermal separation: An approach for a modelica library for absorption; adsorption and rectification. In *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, number 043, pages 804–813. Linköping University Electronic Press, 2009.

Uwe Küssel, Dirk Abel, Matthias Schumacher, and Martin Weng. Modeling of rotary kilns and application to limestone calcination. In *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, number 043, pages 814–822. Linköping University Electronic Press, 2009.

I Lizarralde, T Fernández-Arévalo, C Brouckaert, P Vanrolleghem, DS Ikumi, GA Ekama, E Ayesa, and P Grau. A new general methodology for incorporating physico-chemical transformations into multi-phase wastewater treatment process models. *Water research*, 74:239–256, 2015.

Thomas Marx-Schubach and Gerhard Schmitz. Optimizing the start-up process of post-combustion capture plants by varying the solvent flow rate. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, number 132, pages 121–130. Linköping University Electronic Press, 2017.

Christine Matter-Müller, Willi Gujer, Walter Giger, and Werner Stumm. Non-biological elimination mechanisms in a biological sewage treatment plant. In *Water Pollution Research and Development*, pages 299–314. Elsevier, 1981.

Hubertus Tummescheit and Jonas Eborn. Chemical reaction modeling with thermofluid/mf and multiflash. In *Proceedings of the 2th Modelica Conference*, 2002.

C. Vaneeckhaute, F.H.A. Claeys, F.M.G. Tack, E. Meers, E. Belia, and P.A. Vanrolleghem. Development, implementation, and validation of a generic nutrient recovery model (nrm) library. *Environmental Modelling & Software*, 99:170 – 209, 2018.

Johan Windahl, Katrin Prölss, Maarten Bosmans, Hubertus Tummescheit, Eli van Es, and Awin Sewgobind. Multicomponentmultiphase-a framework for thermodynamic properties in modelica. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, number 118, pages 653–662. Linköping University Electronic Press, 2015.

Algorithms for Component-Based 3D Modeling

Andrea Neumayr[1] Martin Otter[1]

[1]DLR, Institute of System Dynamics and Control, Germany, {andrea.neumayr,martin.otter}@dlr.de

Abstract

The experimental modeling environment Modia3D is used to test and evaluate ideas to model and simulate larger and more complex 3-dimensional systems than is possible with a pure equation-based modeling system such as current Modelica. The goal is to closely combine equation-based modeling with component-based 3D modeling as used in modern game engines. In this article some key algorithms are discussed that have been developed for Modia3D. The overall objective is to utilize the results for the design of the next Modelica language generation.

Keywords: Modelica, Modia, Modia3D, Julia, DAE, equation-based modeling, component-based modeling, multi-body, collision handling

1 Introduction

The Modelica standard library[1] supports the modeling of 3-dimensional multi-body systems with its sub-library Modelica.Mechanics.MultiBody (Otter et al., 2003). There have been several attempts to improve this library with regards to visualization, collision handling or support of larger models, for example (Otter et al., 2005; Höger et al., 2012; Hofmann et al., 2014; Elmqvist et al., 2015; Bardaro et al., 2017). Over the years it was recognized that this is hard because the technology of current Modelica has some natural limitations:

- No modern data structures, like dictionaries or trees, or objects with member functions are supported in Modelica, but they are standard in high level programming languages and are needed to model for example 3D meshes or collision detection algorithms. Then, the only choice is to interface external programs with Modelica models: Developing such algorithms from scratch in, say, C++, and then interface to Modelica is too much effort.
  Using existing code is hard either, because only partial, incompatible solutions are available. For example, it would be nice to interface the Bullet Physics SDK[2] to Modelica to get a state-of-the-art collision handling package. However, this engine determines only the penetration depth of colliding bodies, but for variable-step solvers in offline simulation also zero-crossing functions for DAE-solvers are needed that require the Euclidean distance between non-colliding shapes as well (Neumayr and Otter, 2017). Visualization, collision handling, mass properties calculations require geometric information. Integrating such different description forms in Modelica is hard due to the missing modern data structures. Whenever such packages are integrated, shapes need a unique identification, but this feature is hard to provide in Modelica.

- Modelica tools typically support only generic symbolic transformation algorithms. It is hard or impossible to utilize algorithms which are specialized for a particular model class, for example to remove redundant equations of nonlinear-equation systems due to kinematic loops, to compute a common mass and center of mass of rigidly connected bodies and use it in the simulation, or to use an O(n) multi-body algorithm. In Modelica, a user would have to use a pre-processor that generates Modelica code, see e.g. (Elmqvist et al., 2009).

- Since Modelica compilers typically expand the models for the symbolic engine, the same equation is analyzed many times. For example if a mechanical system has 100 bodies, then the equations of a body are present 100 times in the generated code. C or C++ compilers are not designed to handle huge code parts in a good way. Therefore, there are natural limitations on the model size. For fluid models there are some solutions available, where code for a component is generated only once and reused many times, e.g. (Sahlin and Grozman, 2003). For multi-body systems such solutions might be possible, but yet need to be developed.

The article *Component-Based 3D Modeling of Dynamic Systems* (Neumayr and Otter, 2018) starts an approach to cope with the underlying inherent issues. The basic idea is to combine 3D modeling techniques closely with equation-based modeling à la Modelica within one high level programming environment. Modia[3] (Elmqvist et al., 2016, 2017) is used for the equation-based modeling. It is implemented with the Julia programming language[4] (Bezanson et al., 2017). Julia allows to program numerical algorithms conveniently on a high level. It supports modern data structures, multiple dispatch, metaprogramming, has a just-in-time-compiler and has excellent performance benchmarks relative to C.

[1]https://github.com/modelica/ModelicaStandardLibrary
[2]https://github.com/bulletphysics/bullet3
[3]https://github.com/ModiaSim/Modia.jl
[4]https://julialang.org

Modia utilizes Julias metaprogramming features to integrate an equation-based modeling language with a programming language (e.g. a Modia model can be stored in a dictionary that in turn is inquired in another Modia model to select and use a submodel from this dictionary). Modia3D[5] is designed to model 3D systems, initially only mechanical systems, but it shall be expanded into other domains in the future. It is implemented in Julia and utilizes ideas of multi-body programs and game engines. In the near future, Modia and Modia3D shall be closely integrated, e.g. using a Modia3D model in Modia or using Modia models in Modia3D. Up to now, Modia3D is implemented for functionality and not tuned for efficiency. Therefore, there are no benchmarks yet and in particular no comparison with Modelica models. For animation the free community edition as well as the professional edition[6] of the *DLR Visualization* library[7] (Bellmann, 2009; Hellerer et al., 2014) are used. The overall goal is to apply the results of the Modia/Modia3D prototyping into the design of the next Modelica language generation.

The user's view of Modia3D was introduced in (Neumayr and Otter, 2018) to show the very flexible definition of 3D systems. In this article, several key algorithms are discussed which have been developed for the Modia3D prototype.

2 Component-Based 3D Modeling

Modia3D has two design patterns: the *component-based* and the *hierarchical structure*. The ideas for component-based structuring are from modern game engines, such as Unity or Unreal Engine, which have a component-based design. In the context of game engines a coordinate system is located in 3D and has a container with optional components (in Unity such an object is called *GameObject*[8], in Unreal Engine it is named *Actor*[9], and in Three.js it is called *Object3D*[10]). Each of these components has optional properties such as geometry, visualization, dynamics, collision properties, light, camera, sound, etc., see for example (Nystrom, 2014)[11]. This is a very flexible way to define many optional components and variants and treat them in a modular way. In this paper, this very special design of the generic *component-based design pattern* is called *component-based 3D modeling*. The Julia programming language is particularly suited for this programming pattern. In Section 2 a brief overview to component-based 3D modeling and the features used in this paper is given.

Hierarchical structuring for grouping, aggregating and defining 3D objects is performed with the Modia3D macro `@assembly`. A Julia macro is a metaprogramming[12] language element and starts with `@`. It generates an abstract syntax tree (AST) of Julia code which is automatically compiled and executed at the line where the macro is called. For further information, see (Neumayr and Otter, 2018).

Object3D

In Modia3D, component-based 3D modeling is performed with so-called *Object3D* objects. An Object3D consists of a 3D coordinate system that has optional associated properties collected in the `data` container. Furthermore, an Object3D stores connections to other Object3Ds, via joint, force, or sensor elements (see Figures 2, 1). The code-snippet[13] of the following Julia constructor call[14] creates a new Object3D object `obj`:

```
1  obj = Object3D(parent,data,r=[0,0,0],
2                 R=eye(3),fixed=true)
```

Each `obj` can be defined relative to a `parent` Object3D, with the position vector `r` and the rotation matrix `R`. It is *rigidly connected* to its `parent` if `fixed=true`, and it can move freely if `fixed=false`. The initial position and rotation matrix is defined with `r`, `R`. An Object3D is said to be a reference Object3D, if no `parent` Object3D is given. The 14 Object3Ds of Figure 2 demonstrate different properties and are used below to explain a core algorithm.

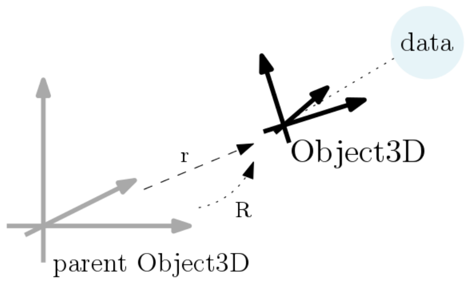


Figure 1. Object3D defined relatively to its parent.

Joint Object

Two Object3Ds can be connected via a joint. In Figure 2 there are several joints, one joint is e.g. between `obj2` and `obj4`.

```
3  joint1 = Revolute(obj2,obj4;axis=3)
```

A reference to the revolute joint is stored in `obj4`. In case the joint introduces a kinematic loop, it is replaced internally by a cut-joint (in Figure 2 this happens e.g. with the joint between `obj5` and `obj13`). A cut-joint is referenced by the two Object3Ds that are constrained by it.

[5]https://github.com/ModiaSim/Modia3D.jl
[6]https://visualization.ltx.de/
[7]http://www.systemcontrolinnovationlab.de/the-dlr-visualization-library/
[8]https://docs.unity3d.com/Manual/GameObjects.html
[9]https://docs.unrealengine.com/en-us/Engine/Components
[10]https://threejs.org/docs/index.html#api/core/Object3D
[11]http://gameprogrammingpatterns.com/component.html
[12]https://docs.julialang.org/en/stable/manual/metaprogramming/
[13]For better reference every code-snippet is marked with a unique line number on the left-hand side.
[14]When calling a Julia function, all optional keyword arguments (name-value pairs) can be given in any order. They are set after the positional arguments (here: `parent` and `data`).

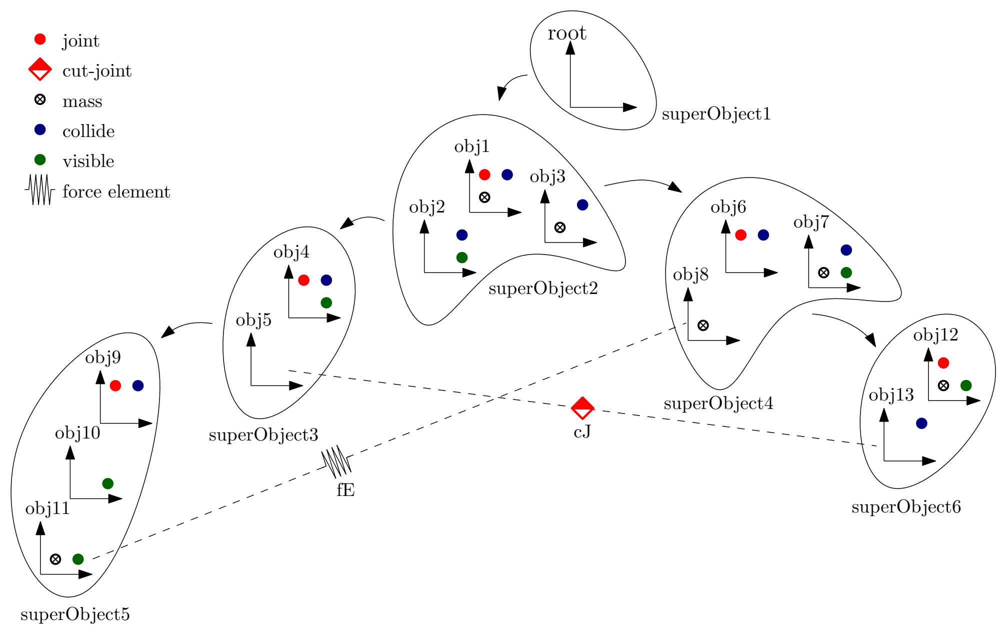


Figure 2. 14 Object3Ds with different properties like they are allowed to collide, can have a mass, are visible and/or can have a force element, are grouped into six rigidly attached general super-objects disjunct via joints and cut-joints.

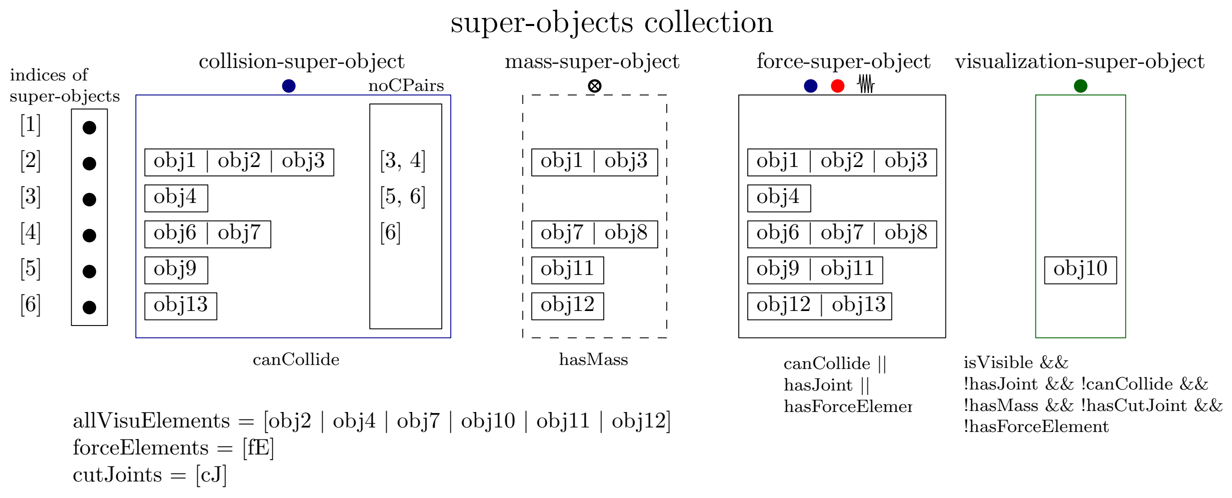


Figure 3. A super-object collection holds four different super-objects types for collision, mass, force computation, and visualization.

Force Object

Two Object3Ds can be connected via a force element. In Figure 2 there is a force element between `obj11` and `obj8`.

```
spring = Spring(obj11,obj8;c = 1e3)
```

A force element is referenced by the two Object3Ds on which it is acting.

Geometry Object

A geometry, such as a sphere, box, cylinder, or a mesh can be defined and associated with an Object3D. For example, a sphere is associated with `obj4` in Figure 2.

```
sphere1 = Object3D(obj4,Sphere(0.9),
                   r=[0,0,0.8])
```

Visualization Object

Visualization objects are an interface to the DLR Visualization library (Bellmann, 2009; Hellerer et al., 2014). For defining the visualization properties a `Material` object has to be associated to a geometry object, or special visualization objects can be used (for example a `CoordinateSystem`). The following constructor call generates a new `Material` object and associates it to a sphere geometry.

```
vmat = Material(color=[0,0,255],
         wireframe=false,transparency=0.5,
         shininess=0.7,reflectslights=true)
sphere2 = Object3D(obj12,
              Sphere(0.9,material=vmat),
              r=[0,0,0.8])
```

A geometry object is only visualized if a visualization material is defined for the object.

MassProperties Object

A `MassProperties` object can be associated with an Object3D to define mass, center-of-mass and inertia tensor with respect to this Object3D. There are various options to define mass properties, for example defining them explicitly (with or without a geometry) or computing them from the volume of a geometry object and of a given density.

Collision Object

The geometry of the associated Object3D takes place in collision handling if a contact response characteristics is defined via an `AbstractContactMaterial` object. For example, an elastic response characteristic with a linear spring and damper is defined with `ContactMaterial-Elastic()`.

3 Collision Handling

Collision detection in Modia3D is based on the Minkowski Portal Refinement algorithm (MPR-algorithm) (Snethen, 2008), which computes the shortest penetration depth of two convex shapes/convex hulls. The MPR-algorithm is much simpler to implement and has less numerical problems than the often used GJK/EPA-standard algorithms (Gilbert et al., 1988; Bergen, 2003), because it only works with triangles and not with tetrahedrons.

A Modia3D model is mathematically defined as a Differential-Algebraic-Equation system (DAE) with $\boldsymbol{x} = \boldsymbol{x}(t)$ and a regular Jacobian $\boldsymbol{J}$ (1c):

$$\begin{aligned} \mathbf{0} &= \begin{bmatrix} \boldsymbol{f}_d(\dot{\boldsymbol{x}}, \boldsymbol{x}, t, z_i > 0) \\ \boldsymbol{f}_c(\boldsymbol{x}, t, z_i > 0) \end{bmatrix} \quad (a) \qquad \boldsymbol{J} = \begin{bmatrix} \dfrac{\partial \boldsymbol{f}_d}{\partial \dot{\boldsymbol{x}}} \\ \dfrac{\partial \boldsymbol{f}_c}{\partial \boldsymbol{x}} \end{bmatrix} \quad (c) \qquad (1) \\ \boldsymbol{z} &= \boldsymbol{f}_z(\boldsymbol{x}, t) \quad (b) \end{aligned}$$

Therefore, (1a) is an index 1 DAE and (1b) defines zero-crossing functions $\boldsymbol{z}(t)$. To speed up the simulation and to improve the robustness of the integration, Modia3D uses the distances between convex shapes as zero-crossing functions $z_i(t)$ (1b).

In the original version of the MPR-algorithm (Snethen, 2008) only penetration depths are determined. In Modia3D improvements of the MPR-algorithm are utilized that have been proposed in (Kenwright, 2015; Neumayr and Otter, 2017), in particular to compute the distances of shapes that are not in contact and treating special collision situations properly.

In Modia3D collision handling of n potentially colliding shapes is performed in the following (mostly standard) way:

1. *Broad Phase*
   The shapes are approximated by bounding volumes where potential collisions can be very cheaply determined resulting in $O(n^2)$ cheap tests. When using special data structures (such as octrees or kd-trees), it is possible to reduce the number of cheap tests to $O(n\log(n))$.

2. *Narrow Phase*
   For the potentially colliding shape pairs as identified in the broad phase, the signed distances are computed with the improved MPR-algorithm (Neumayr and Otter, 2017).

3. *Response Calculation*
   If two shapes are penetrated, a force and/or torque is applied at the contact point, such as a spring - damper force element, depending on the penetration depth.

The broad phase in Modia3D uses *AABB*s (= Axis Aligned Bounding Boxes) (see e.g. (Bergen, 2003)). Each AABB approximates one shape and only if the AABBs are intersecting, the distance between these two possibly colliding shape pairs is calculated in the narrow phase. A preprocessing of the tree-structure is executed to reduce the number of possible collision pairs to n_{pp} before the broad phase is processed. This leads to $n_{pp} \leq O(n^2)$ tests. There are two preprocessing rules:

1. Rigidly attached shapes cannot collide with each other.

2. Shapes connected by a joint cannot collide with each other if the joint specific option `canCollide` is set to `false` (the default setting).

4 Object Preprocessing

In this section a central preprocessing step of Modia3D is explained. The goal is to evaluate efficiently many objects of different kinds during integration.

For example, the position and orientation of a visualization object should only be computed when needed (at communication points), and not in every model evaluation. Furthermore, if mass properties are associated with rigidly connected Object3Ds (two or more), then the resultant mass properties of all these objects is computed once in the preprocessing step (note, such an operation is hard to automatically perform with a Modelica multi-body model).

Figure 2 presents an example of a Modia3D model, and the connected objects are given. The goal is to generate the data structure that is shown in Figure 3. Afterwards, the usage of this data structure for an efficient evaluation of the model during integration is explained.

4.1 Super-Objects

Rigidly connected Object3Ds are grouped together into so-called *super-objects*. Super-objects are disjunct via joints. Without any further assumptions, the grouping of the 14 Object3Ds of Figure 2 leads to six general super-objects (Figure 4). Figure 5 also shows these six super-objects connected via joints/cut-joints. The super-objects $2, 3, 4$ and 6 are forming a kinematic loop. This kinematic loop is detected and the joint between super-object 3 and 6 is

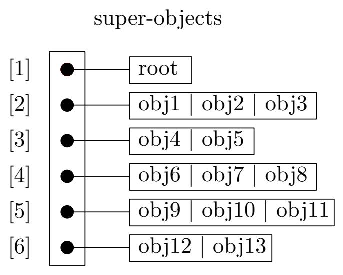


Figure 4. Six general super-objects.

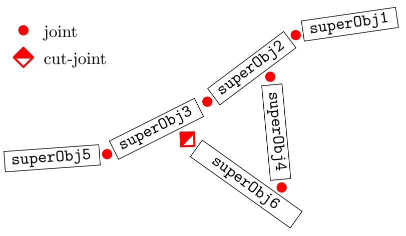


Figure 5. Six super-objects are connected via joints/cut-joints.

internally replaced by a cut-joint. Object3Ds can have several properties that are collected in different super-object data structures.

4.1.1 Super-Objects Collection

In the *super-objects collection* all information about different super-object types is stored, for example super-objects for collision handling, mass and force computation as well as visualization. The super-objects collection of the above mentioned example (Figure 2) with its different super-objects is shown in Figure 3. For each of the super-object types, there is a function `assignObj(obj,superObjType)` to store a reference of the object in the corresponding data structure identified by `superObjType`. A super-object collection has 3 additional containers: `allVisuElements` stores every Object3D which has visualization properties, all force elements and all cut-joints are stored respectively in `forceElements` and `cutJoints` (see Figure 3).

4.1.2 Super-Objects for Collision Handling

A geometry associated with an Object3D takes place in collision handling, in case a contact material is defined (see Section 2) and hence it gets assigned to a *collision-super-object* (lines 13 - 17).

```
13 function assignObj(obj::Object3D,
14            superObjType::SuperObjCollision)
15  if canCollide(obj)
16   push!(superObjType.superObj, obj)
17 end; end
```

All Object3Ds within one collision-super-object are rigidly connected, so they cannot collide with respect to each other and therefore they already fulfill the first preprocessing rule (see Section 3). To fulfill also the second preprocessing rule, all collision-super-objects which are disjunct by a joint/cut-joint are not allowed to collide either (if option `canCollide = false`). Therefore, the indices of collision-super-objects which are not allowed to collide, are stored in a collision-list called *no collision pairs* (= `noCPairs`). For example `superObject2` is not allowed to collide with `superObject3` and vice versa (see Figures 2, 3, 5). It is sufficient to store this relation only once for the first executed super-object. This leads to the corresponding `noCPairs` container for collision-super-objects (see Figure 3).

4.1.3 Super-Objects for Mass Computation

In case mass properties are defined for an Object3D it gets assigned to a *mass-super-object* (lines 18 - 22). In a later step, a resultant mass, center-of-mass and inertia tensor is computed for all mass properties of one mass-super-object.

```
18 function assignObj(obj::Object3D,
19            superObjType::SuperObjMass)
20  if hasMass(obj)
21   push!(superObjType.superObj, obj)
22 end; end
```

4.1.4 Super-Objects for Force Computation

All Object3Ds which are allowed to collide, or have a joint, or a force element are stored in a *force-super-object* (lines 23 - 28). For these Object3Ds kinematic laws (positions, translation matrices, etc.) and especially forces need to be re-calculated in each solver step.

```
23 function assignObj(obj::Object3D,
24            superObjType::SuperObjForce)
25  if (canCollide(obj) || hasJoint(obj) ||
26      hasForceElement(obj))
27   push!(superObjType.superObj, obj)
28 end; end
```

4.1.5 Super-Objects for Visualization

Object3Ds within a *visualization-super-object* are exclusively for visualization (lines 29 - 36). The positions and rotation matrices only need to be calculated at communication points with the visualization engine.

```
29 function assignObj(obj::Object3D,
30            superObjType::SuperObjVisu)
31  if (isVisible(obj) && !hasJoint(obj) &&
32      !hasMass(obj) && !canCollide(obj) &&
33      !hasForceElement(obj) &&
34      !hasCutJoint(obj))
35   push!(superObjType.superObj, obj)
36 end; end
```

4.2 Algorithm for Constructing Super-Objects

The goal of this section is to introduce an algorithm to detect and group rigidly attached super-objects. This algorithm is based on a *depth-first search algorithm* (DFS) (Tarjan, 1972; Hopcroft and Tarjan, 1974). The depth-first search as well as the augmented version takes $O(n)$ time.

4.2.1 Depth-First Search Algorithm

The depth-first search algorithm explores each branch as far as possible to its leaves-level, afterwards it is stepping back (see Figure 6). This procedure uses a stack and is

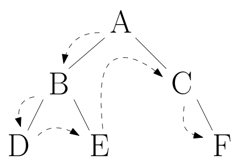


Figure 6. Example of a DFS.

executed until the stack is empty and hence every node has been visited exactly once. Below a Julia pseudo code is shown (lines 37 - 45). The DFS-algorithm works with a Last In First Out - stack (LIFO). Therefore, `append!` (line 44) inserts all children of an `obj` at the end of the stack and `pop!` (line 42) returns the last item. All nodes are stored in the above described order and a depth-first search of the example in Figure 6 would lead to `result = [A,B,D,E,C,F]`.

```
37 stack  = []
38 result = []
39 function DFS(root)
40    push!(stack,root)
41    while length(stack) > 0
42       obj = pop!(stack)
43       push!(result,obj)
44       append!(stack,obj.children)
45 end; end
```

4.2.2 Augmented Depth-First Search Algorithm

The augmented depth-first search is based on the idea of the depth-first search algorithm (Section 4.2.1). Below, a Julia pseudo code of the augmented depth-first search is presented (lines 54 - 79). It creates a super-object collection which holds all described super-object types. The assignment of each Object3D takes place in function `assignAll(...)` (lines 66, 71).

Every Object3D, except one, has a parent object and all Object3Ds and their associated properties are connected together and build up a tree, see Figure 2. The Object3D without a parent is treated as the world object and it is the root of the tree. The world object is not allowed to have any additional properties, except for visualization. In case any other Object3D has no parent an error occurs. Since the Object3Ds form a tree, the root of every super-object is an Object3D that is connected via a joint or it can move freely with respect to its parent. This parent is located on a different super-object.

The augmented DFS-algorithm works with a stack-like buffer and a stack.

buffer Whenever the root of a new super-object is found, it is pushed on the buffer. The index of an element in the buffer is also the index of the super-object. Therefore, the maximally reached length of the buffer is equal to the total amount of general super-objects. Additionally, variable `actPos` holds the index of the super-object that is currently processed. When the processing of a super-object is finished, this variable is incremented by one as long as there are elements on the buffer.

stack Starting from the root of a super-object, all Object3Ds for this super-object are inspected with the help of this stack. Whenever a boundary (an Object3D with a joint or a freely moving Object3D) is reached, this Object3D is pushed on the buffer (and not on the stack). All Object3Ds of the super-object have been inspected in depth-first order (see Figure 4), if the stack is empty.

The properties of a super-object are stored in the following structure:

```
46 mutable struct SuperObjs
47  superObjCollision::SuperObjCollision
48  superObjMass::SuperObjMass
49  superObjForce::SuperObjForce
50  superObjVisu::SuperObjVisu
51  noCPair::Array{Int64,1}
52  ...
53 end
```

Hereby, every essential property of an Object3D is an element of this struct (such as `superObjMass` which holds all Object3Ds that have a mass) of the corresponding super-objects.

The top-level part of the algorithm:

```
54 stack  = []
55 buffer = []
56 coll   = SuperObjCollection()
57 augmentDFS!(root_obj)
```

initializes the stack, the buffer and the super-object collection and then calls function `augmentedDFS!` with the root of the Object3D tree (= the topmost parent Object3D of the root level assembly) as input argument. The details of function `augmentedDFS!` are given below:

```
58 function augmentedDFS!(root::Object3D)
59   push!(buffer, root)
60   actPos = 1
61   nPos   = 1
62   while actPos <= nPos
63     superObj = SuperObjs()
64     obj = buffer[actPos]
65     if obj != root
66       assignAll(superObj,obj)
67     end
68     fillStackOrBuffer!(superobj,obj)
69     while length(stack) > 0
70       objChild = pop!(stack)
71       assignAll(superObj,objChild)
72       fillStackOrBuffer!(superObj,objChild)
73     end
74     safeSuperObjsToCollection(coll,superObj)
75     nPos = length(buffer)
76     actPos += 1
77   end
78   addIndicesOfCutJointsToSuperObj(coll)
79 end
```

First, the root Object3D is pushed on the buffer and the current element of the buffer `actPos` is set to one. Afterwards all elements of the buffer are inspected. For every element of the buffer a depth-first search is performed. All

Object3Ds are pushed on the stack that are rigidly connected with their parents. Otherwise, it is pushed on the buffer. This decision is made with function `fillStackOrBuffer!`:

```
80 function fillStackOrBuffer!(superObj,obj)
81  for child in obj.children
82   if isNotRoot(child)
83    if isNotFixed(child)
84     push!(buffer,child)
85     if !child.joint.canCollide
86      push!(superObj.noCPair,length(buffer))
87     end
88    else
89     push!(stack,child)
90 end; end; end; end
```

For each element of the `SuperObjs` data structure (lines 46 - 53) the `assignAll` function:

```
91 function assignAll(superObj,obj)
92   for val in fieldnames(typeof(superObj))
93     assignObj(getfield(superObj,val), obj)
94   end
95 end
```

calls the `assignObj` function to store the Object3D in the particular specialized super-object. The right `assignObj` function is chosen via multiple dispatch of the Julia programming language, see Section 4.1.

4.3 Algorithms for Using Super-Objects

In this section, the usage of the generated data structure is shortly sketched for an efficient evaluation during integration.

4.3.1 Usage of Collision-Super-Objects

The MPR-algorithm computes the distance between two shapes in the narrow phase `narrowPhase_MPR` (line 111) if their AABBs are intersecting in the broad phase `broadPhase_checkAABB` (line 110). All Object3Ds (line 104) of the actual super-object (line 102) are allowed to collide with all Object3Ds (line 109) of the subsequent super-object (line 107). In case the actual super-object is not allowed to collide with the subsequent super-object, the index of the subsequent super-object is stored in `noCPairs` (see Figure 3 and Section 4.1.2).

```
96  # counter
97  # is: actual super-object
98  # js: subsequent super-object
99  # i: Object3D of is_th super-object
100 # j: Object3D of js_th super-object

101 for is = 1:length(collSuperObjs)
102  actSuperObj = collSuperObjs[is]
103  for i = 1:length(actSuperObj)
104   actObj = actSuperObj[i]
105   for js = is+1:length(collSuperObjs)
106    if !(js in noCPairs[is])
107     nextSuperObj = collSuperObjs[js]
108     for j = 1:length(nextSuperObj)
109      nextObj = nextSuperObj[j]
110      if broadPhase_checkAABB(actObj,nextObj)
111       narrowPhase_MPR(actObj,nextObj)
112 end; end; end; end; end; end
```

4.3.2 Usage of Mass-Super-Objects

If there are two or more Object3Ds with mass-properties in a super-object, the resultant mass, inertia tensor and center of mass is computed and a new Object3D is constructed at the center-of-mass location. The previous mass-properties objects are removed.

4.3.3 Usage of Force- and Visualization-Super-Objects

In general, the Object3Ds on a super-object form a tree. This tree is reconstructed so that every Object3D has the root of the super-object as parent, in order to avoid unnecessary coordinate transformations during integration. All Object3Ds with exception of the Object3Ds that are only used for visualization, are stored in an Object3D vector `Object3DEvaluation` in depth-first order. During integration, the absolute position, rotation, velocity, angular velocity, acceleration, angular acceleration of all Object3Ds are computed at every model evaluation by traversing this vector from index 1 up to its last index and computing the absolute quantities of an Object3D with its relative quantities and the absolute quantities of its parent Object3D. Furthermore, the forces and torques due to the acceleration/angular acceleration of the mass properties Object3Ds from section 4.3.3 are computed and stored in the respective Object3D, as well as the forces and torques of all `forceElements` and of all contact force elements. Afterwards, vector `Object3DEvaluation` is traversed from its last index down to index 1 and the resultant force and torque at an Object3D is transformed and summed to the force and torque of its parent Object3D. Finally, the projection of the forces/torques at all joints into the non-constrained motion of the respective joint results in the residues $\bar{\boldsymbol{f}}_2$ of equation (7).

The handling of the cut-joints is a bit more involved (to compute the residues $\bar{\boldsymbol{f}}_3, \bar{\boldsymbol{f}}_4$) and is not further elaborated here. The absolute position and rotation of the Object3Ds that have only visualization-objects need to be computed only at communication points. This calculation is a simple extension of the approach sketched above.

5 Simulation

Once the preprocessing steps are finished, the model is transformed to DAE form (1) as sketched in section 4.3.3 and solved with Sundials IDA (Hindmarsh et al., 2005, 2015) that uses a variable-step, variable-order BDF-integration (Backward Differentiation Formula) method. The transformation of a multi-body system with kinematic loops (for an example see figure 7) to the form (1) is sketched in (Otter and Elmqvist, 2017) and shortly repeated here:

Starting point are the equations of motion of a multi-body system, see, e.g. (Arnold, 2016):

$$\begin{aligned} \dot{\boldsymbol{q}} &= \boldsymbol{v} \\ \boldsymbol{M}(\boldsymbol{q},t)\dot{\boldsymbol{v}} + \boldsymbol{G}^T(\boldsymbol{q},t)\boldsymbol{\lambda} &= \boldsymbol{h}(\boldsymbol{q},\boldsymbol{v},t) \\ \boldsymbol{0} &= \boldsymbol{g}(\boldsymbol{q},t) \end{aligned} \tag{2}$$

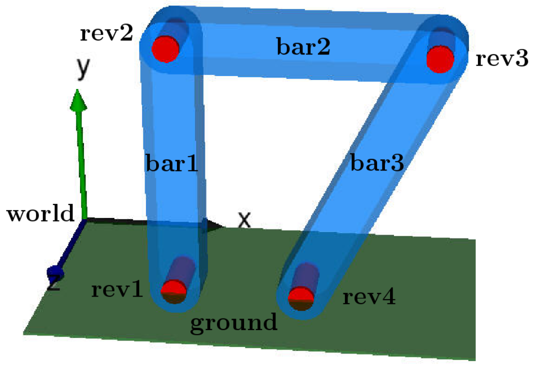


Figure 7. Modia3D model of a multi-body system with a kinematic loop.

where $\boldsymbol{q}$ are the generalized coordinates (here: joint coordinates, such as a revolute angle), $\boldsymbol{v}$ are the generalized velocities, $\boldsymbol{\lambda}$ are the generalized forces/torques in the cut-joints, $\boldsymbol{M} = \boldsymbol{M}^T$ is the positive definite mass matrix, $\boldsymbol{g}$ are the kinematic constraint equations of the cut-joints on position level, $\boldsymbol{G} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}$ are the partial derivatives of the constraints equations with respect to $\boldsymbol{q}$ and has full row rank. This DAE has index 3 and gives rise to numerical problems when integrating it directly. Instead, with the method of (Gear et al., 1985; Gear, 1988) it can be transformed to the following index 1 form, see (Otter and Elmqvist, 2017):

$$\begin{aligned}
\boldsymbol{0} &= \dot{\boldsymbol{q}} - \boldsymbol{v} + \boldsymbol{G}^T(\boldsymbol{q},t)\dot{\boldsymbol{\mu}}_{int} \\
\boldsymbol{0} &= \boldsymbol{M}(\boldsymbol{q},t)\dot{\boldsymbol{v}} + \boldsymbol{G}^T(\boldsymbol{q},t)\dot{\boldsymbol{\lambda}}_{int} - \boldsymbol{h}(\boldsymbol{q},\boldsymbol{v},t) \\
\boldsymbol{0} &= \boldsymbol{g}(\boldsymbol{q},t) \\
\boldsymbol{0} &= \boldsymbol{G}(\boldsymbol{q},t)\boldsymbol{v} + \boldsymbol{g}^{(1)}(\boldsymbol{q},t)
\end{aligned} \tag{3}$$

where (a) the derivative of the constraint equations $\boldsymbol{0} = \boldsymbol{g}(\boldsymbol{q},t)$ are added as new equations, (b) new unknowns $\dot{\boldsymbol{\mu}}_{int}$ are introduced that are used for stabilizing the DAE and (c) the generalized constraint forces $\boldsymbol{\lambda}$ are replaced by $\dot{\boldsymbol{\lambda}}_{int}$ the derivatives of its integral. The question is how these equations can be constructed by Modia3D:

IDA and other DAE integrators assume that the DAE is mathematically described as:

$$\boldsymbol{0} = \boldsymbol{f}(\dot{\boldsymbol{y}}(t), \boldsymbol{y}(t), t) \tag{4}$$

For the solution, the following Jacobian is computed numerically (c_h is a step-size dependent variable that is provided by the integrator):

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}} + c_h \cdot \frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{y}}} \tag{5}$$

This Jacobian can be automatically generated by IDA, but is optionally also provided by Modia3D (to experiment with sparse Jacobians).

One problem is that the new term $\boldsymbol{G}^T(\boldsymbol{q},t)\dot{\boldsymbol{\mu}}_{int}$ does not appear in the equations of motions and should be "somehow" constructed. The DAE variables $\boldsymbol{y}$ of the IDA interface are defined as:

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \boldsymbol{y}_3 \\ \boldsymbol{y}_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{v} \\ \boldsymbol{\lambda}_{int} \\ \boldsymbol{\mu}_{int} \end{bmatrix} \tag{6}$$

Hereby $\boldsymbol{q}$ are the generalized coordinates of the joints in the tree of the super-objects (for example an angle of a revolute joint), $\boldsymbol{v}$ are the generalized velocities of these joints (for example the angular velocity of a revolute joint), $\boldsymbol{\lambda}_{int}$ is the integral of the generalized cut-forces in the cut-joints (for example the cut-forces of a spherical cut-joint) and $\boldsymbol{\mu}_{int}$ does not appear in the model.

In a first step the residues of the model equations are computed in the following form (note that $\boldsymbol{y}, \dot{\boldsymbol{y}}$ are provided by the IDA integrator as input arguments to the model):

$$\bar{\boldsymbol{f}} = \begin{bmatrix} \bar{\boldsymbol{f}}_1 \\ \bar{\boldsymbol{f}}_2 \\ \bar{\boldsymbol{f}}_3 \\ \bar{\boldsymbol{f}}_4 \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{y}}_1 - \boldsymbol{y}_2 \\ \boldsymbol{M}\dot{\boldsymbol{y}}_2 + \boldsymbol{G}^T\dot{\boldsymbol{y}}_3 - \boldsymbol{h}(\boldsymbol{y}_1, \boldsymbol{y}_2, t) \\ \boldsymbol{g}(\boldsymbol{y}_1, t) \\ \boldsymbol{G}(\boldsymbol{y}_1, t)\boldsymbol{y}_2 + \boldsymbol{g}^{(1)}(\boldsymbol{y}_1, t) \end{bmatrix} \tag{7}$$

Hereby, $\bar{\boldsymbol{f}}_1$ is directly computed from the input arguments, $\bar{\boldsymbol{f}}_2$ are the generalized forces of the joints in the super-object tree (for example the projection of the cut-torque in a revolute joint to its axis of rotation, see section 4.3.3), $\bar{\boldsymbol{f}}_3$ are the generalized kinematic closure conditions of the cut-joints on position level and $\bar{\boldsymbol{f}}_4$ are the generalized kinematic closure conditions of the cut-joints on velocity level.

From time to time (so not in every step) the integrator requires a Jacobian (5). First, the part of the Jacobian is computed numerically where $\dot{\boldsymbol{\mu}}_{int}$ is not yet taken into account (using (7)):

$$\bar{\boldsymbol{J}} = \frac{\partial \bar{\boldsymbol{f}}}{\partial \boldsymbol{y}} + c_h \cdot \frac{\partial \bar{\boldsymbol{f}}}{\partial \dot{\boldsymbol{y}}} \tag{8}$$

It can be noted that matrix $\boldsymbol{G}$ is part of this Jacobian (the rows of this Jacobian with respect to $\bar{\boldsymbol{f}}_4$ and the columns with respect to $\boldsymbol{y}_2$):

$$\boldsymbol{G} = \frac{\partial \bar{\boldsymbol{f}}_4}{\partial \boldsymbol{y}_2} = \bar{\boldsymbol{J}}_{42} \tag{9}$$

It is now possible to compute the Jacobian (5) as required by IDA:

$$\boldsymbol{J} = \bar{\boldsymbol{J}} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & c_h \cdot \bar{\boldsymbol{J}}_{42}^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \tag{10}$$

Furthermore, in every model evaluation also the residues (4) can be calculated as required by IDA:

$$\boldsymbol{f} = \bar{\boldsymbol{f}} + \begin{bmatrix} \bar{\boldsymbol{J}}_{42}^T \cdot \boldsymbol{y}_4 \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} \tag{11}$$

Since the Jacobian (5) is not computed in every integrator step, $\bar{\boldsymbol{J}}_{42}^T \cdot \boldsymbol{y}_4$ need not be identical to $\boldsymbol{G}^T(\boldsymbol{q},t)\dot{\boldsymbol{\mu}}_{int}$ because $\bar{\boldsymbol{J}}_{42}^T$ is potentially computed at a previous time instant. However, this is uncritical because the method of Gear to stabilize the DAE only requires that matrix $\boldsymbol{G} \cdot \bar{\boldsymbol{J}}_{42}^T$ must be regular (see for example (Otter and Elmqvist, 2017): the derivation after eq. (13) shows that $\dot{\boldsymbol{\mu}}_{int} = \boldsymbol{0}$). In the unlikely situation that this approximation of the stabilizing term looses rank, the integrator will most likely detect a problem with its variable step-size control and will force a new computation of the Jacobian, that will solve the issue.

6 Relation to other Work

Modern games use physics engines, like Havok or PhysX for collision detection and rigid body simulations (Gregory, 2014). Physics engines of games work with fixed-step size solvers and are interactive real-time simulations. Modia3D simulates the system with variable-step size solver because the target of its initial version is offline simulation. Therefore, there are natural differences between the implementation approaches, for example in a physics engine the position and orientation of visual objects need to be computed in every model evaluation, whereas in Modia3D this is only needed at communication points (if the visual object does not take place in collision handling). To improve efficiency, this is specially handled in Modia3D.

Game engines typically use a scene graph[15] (Gregory, 2014) to describe the representation of the 3D objects. Often this is a tree data structure where all operations applied on a node effect all children nodes. Changes to nodes might be material data, such as the color of objects, or 3D transformations. Usually, closed kinematic loops are not supported by game engines or are approximated with various techniques. Therefore, a scene graph with a tree data structure is sufficient.

In Modia3D kinematic loops are inherently supported and therefore a pure tree data structure does not reflect the system. From a users point of view a 3D system is a graph with loops. It seems therefore not useful to apply, say, a color to a node and define that this color holds for all children, because the children might be part of a loop that includes the node. Instead in Modia3D, an object such as a material object might be defined once and then references to this object might be used in the various nodes. For practical reasons, the graph is represented internally as a tree with additional information for the closing conditions of kinematic loops. However, this is hidden from the user and the user should not know in which way an internal tree is constructed (this might even change with a new version).

The Modelica MultiBody library (Otter et al., 2003) was implemented in 2003 and since this time only minor improvements have been made. The design is made in a rigid way by defining a few part types, such as `BodyShape` or `BodyBox`, to represent a fixed setup for a part with a geometry, mass properties computed from this geometry and a fixed set of frame connectors. This design is far away from the flexibility of the Modia3D library where various geometries, including base shapes and meshes, can be defined and used in various ways for visualization, mass properties computation, collision handling. Systems with kinematic loops can be defined with the Modelica MultiBody library, but it was not possible to make this fully automatic so that the user just defines the system as it is. Instead, practically the user has to define somehow the cut-joints or assembly-joints for a kinematic loop and also has usually to explicitly define the states in a loop with the `StateSelect` attribute, because otherwise the simulation becomes much too slow due to the dynamic state selection.

[15] https://en.wikipedia.org/wiki/Scene_graph

7 Conclusion

In this article some newly developed algorithms have been described that are used by the Modia3D prototype to construct a model that can be efficiently evaluated in a simulation with a variable-step solver. Due to its architecture that is inspired by game engines, Modia3D allows a very flexible way to build-up dynamic models of 3D-mechanical systems and to model collisions. Contrary to games, the main target of the package design are variable-step solvers with step-size control. Modia3D is still an early prototype and several important parts are under development, especially the integration with Modia is missing. Furthermore, the code was currently mainly developed for its functionality and is not yet tuned for efficiency. For these reasons, benchmarks about the simulation efficiency have not yet been performed.

References

M. Arnold. DAE aspects of multibody systems. Technical report, Martin-Luther-Universität Halle-Wittenberg, Institut für Mathematik, April 2016. URL http://sim.mathematik.uni-halle.de/reports/sources/2016/01-2016.pdf.

G. Bardaro, L. Bascetta, F. Casella, and M. Matteucci. Using Modelica for advanced Multi-Body modelling in 3D graphical robotic simulators. In J. Kofranek and F. Casella, editors, *Proc. of the 12th International Modelica Conference*. LiU Electronic Press, May 2017. URL http://www.ep.liu.se/ecp/132/097/ecp17132887.pdf.

T. Bellmann. Interactive Simulations and advanced Visualization with Modelica. In Francesco Casella, editor, *Proc. of the 7th International Modelica Conference*. LiU Electronic Press, Sept. 2009. URL http://www.ep.liu.se/ecp/043/062/ecp09430056.pdf.

G.v.d. Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann Publishers, 2003.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59 (1):65–98, 2017.

H. Elmqvist, S. E. Mattsson, and C. Chapuis. Redundancies in Multibody Systems and Automatic Coupling of CATIA and Modelica. In *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, pages 551–560. Linköping University Electronic Press, 2009. URL http://www.ep.liu.se/ecp/043/063/ecp09430113.pdf.

H. Elmqvist, A. Goteman, V. Roxling, and T. Ghandriz. Generic Modelica Framework for MultiBody Contacts and Discrete Element Method. In Peter Fritzson and Hilding Elmqvist, editors, *Proc. of the 11th International Modelica Conference*. LiU Electronic Press, Sept. 2015. URL http://www.ep.liu.se/ecp/118/046/ecp15118427.pdf.

H. Elmqvist, T. Henningsson, and M. Otter. Systems Modeling and Programming in a Unified Environment based on Julia. In *Proc. of ISoLA Conference*. Springer, Oct. 2016. doi:10.1007/978-3-319-47169-3_15.

H. Elmqvist, T. Henningsson, and M. Otter. Innovations for Future Modelica. In J. Kofranek and F. Casella, editors, *Proc. of the 12th International Modelica Conference*. LiU Electronic Press, May 2017. URL http://www.ep.liu.se/ecp/132/076/ecp17132693.pdf.

C. W. Gear. Differential-algebraic equation index transformations. *SIAM J. Sci. Stat. Comput.*, 9(1):39 – 47, 1988.

C. W. Gear, G.K. Gupta, and B. Leimkuhler. Automatic integration of euler–lagrange equations with constraints. *J. Comp. Appl. Math.*, 12-13:77 – 90, 1985.

E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988. URL https://graphics.stanford.edu/courses/cs448b-00-winter/papers/gilbert.pdf.

J. Gregory. *Game engine architecture*. AK Peters/CRC Press, 2014.

M. Hellerer, T. Bellmann, and F. Schlegel. The DLR Visualization Library - Recent development and applications. In Hubertus Tummescheit and Karl-Erik Arzen, editors, *Proc. of the 10th International Modelica Conference*. LiU Electronic Press, March 2014. URL http://www.ep.liu.se/ecp/096/094/ecp14096094.pdf.

C. Höger, A. Mehlhase, C. Nytsch-Geusen, K. Isakovic, and R. Kubiak. Modelica3D - Platform Independent Simulation Visualization. In M. Otter and D. Zimmer, editors, *Proc. of the 9th International Modelica Conference*, Sept. 2012. URL http://www.ep.liu.se/ecp/076/049/ecp12076049.pdf.

A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, September 2005.

A.C. Hindmarsh, R. Serban, and A. Collier. User Documentation for IDA v2.8.2. Technical Report UCRL-SM-208112, Lawrence Livermore National Laboratory, 2015.

A. Hofmann, L. Mikelsons, I. Gubsch, and C. Schubert. Simulating Collisions within the Modelica MultiBody Library. In Hubertus Tummescheit and Karl-Erik Arzen, editors, *Proc. of the 10th International Modelica Conference*. LiU Electronic Press, March 2014. URL http://www.ep.liu.se/ecp/096/099/ecp14096099.pdf.

J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of the ACM (JACM)*, 21(4):549–568, 1974.

B. Kenwright. Generic Convex Collision Detection using Support Mapping. Technical report, 2015. URL https://www.semanticscholar.org/paper/Generic-Convex-Collision-Detection-using-Support-Kenwright/4f0f2d95375db7cfdbfaa345847418789d8cb970.

A. Neumayr and M. Otter. Collision Handling with Variable-step Integrators. In *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, EOOLT'17, pages 9–18. ACM, 2017. URL https://modiasim.github.io/Modia3D.jl/resources/documentation/CollisionHandling_Neumayr_Otter_2017.pdf.

A. Neumayr and M. Otter. Component-Based 3D Modeling of Dynamic Systems. In M. Tiller, H. Tummescheit, and L. Vanfretti, editors, *Proceedings of the American Modelica Conference*, Oct. 2018. URL https://elib.dlr.de/124126/1/2018_Modelica_Modia3D.pdf.

R. Nystrom. *Game Programming Patterns*. Genever Benning, 2014. URL http://gameprogrammingpatterns.com/.

M. Otter and H. Elmqvist. Transformation of Differential Algebraic Array Equations to Index One Form. In J. Kofranek and F. Casella, editors, *Proc. of the 12th International Modelica Conference*, May 2017. URL http://www.ep.liu.se/ecp/132/064/ecp17132565.pdf.

M. Otter, H. Elmqvist, and S. E. Mattsson. The New Modelica MultiBody Library. In P. Fritzson, editor, *Proc. of the 3rd International Modelica Conference*, Nov. 2003. URL https://www.modelica.org/events/Conference2003/papers/h37_Otter_multibody.pdf.

M. Otter, H. Elmqvist, and J. Diaz Lopez. Collision Handling for the Modelica MultiBody Library. In Gerhard Schmitz, editor, *Proc. of the 4th International Modelica Conference*, March 2005. URL https://modelica.org/events/Conference2005/online_proceedings/Session1/Session1a4.pdf.

P. Sahlin and P. Grozman. IDA Simulation Environment - a tool for Modelica based end-user application deployment. In P. Fritzson, editor, *Proc. of the 3rd International Modelica Conference*, Nov. 2003. URL https://www.modelica.org/events/Conference2003/papers/h33_Sahlin.pdf.

G. Snethen. Xenocollide: Complex collision made simple. In Scott Jacobs, editor, *Game Programming Gems 7*, pages 165–178. Charles River Media, 2008.

R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

Model visualization for e-learning Kidney simulator for medical students

Jan Šilar[1] Filip Ježek[1,2] Arnošt Mládek[1] David Polák[1] Jiří Kofránek[1]

[1]Institute of Pathological physiology, First Faculty of Medicine, Charles University, Prague, Czech republic, {jan.silar, filip.jezek, arnost.mladek, david.polak, jiri.kofranek}@lf1.cuni.cz
[2]Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Abstract

The present paper introduces a recently developed tool for building web-based simulators called *Bodylight.js*. Simulators are applications composed of a mathematical model and a graphical user interface that allows the user to easily interact with the model and visualize the results. A modelica model is first exported to FMI with sources, transcompiled into JavaScript and WebAssembly and connected to a GUI, comprised of graphical animations created in Adobe Animate and elements that allow to control the input model such as sliders, buttons, etc.

A physiological e-learning application explaining the function of a nephron – the basic functional unit of kidneys – is presented later as a use-case. The model was developed primarily as a teaching aid for use in courses of physiology for medical students at our university.

Purpose of this work is to describe the new Bodylight.js tool and to prove its usability by building the medium-complex e-learning kidney simulator. The simulator helps medical students to better understand renal function at the very basic level.

Keywords: Modelica, JavaScript, WebAssembly, web technologies, physiology, kidney, e-learning

1 Introduction

Mathematical models are powerful tools for gaining insight into the systems under study. It is sometimes feasible to experiment with a human body directly. For example a man can drink a lot of water and, due to the enhanced urine production, has to urinate sooner. But this is just an outer behavior of the system. It is not so easy to grasp the underlying mechanisms: how was the swallowed water absorbed into the intestine blood vessels? Why the blood did not get diluted? Further, how is the primary urine produced? Here models and simulation applications may help us to comprehend the underlying mechanisms. Functional models are already being used in medical education (Kofranek *et al,* 2011) (namely in physiology and pathophysiology). But we believe that the benefits are still widely underestimated and that illustrative models should be used regularly in lectures and practical classes.

Our ultimate goal is to produce simulation applications for teaching physiology that both look and behave like the simulated system so that they are as much understandable as possible. We need a modeling tool, graphical animation tool and tool to connect both the model and the animation together.

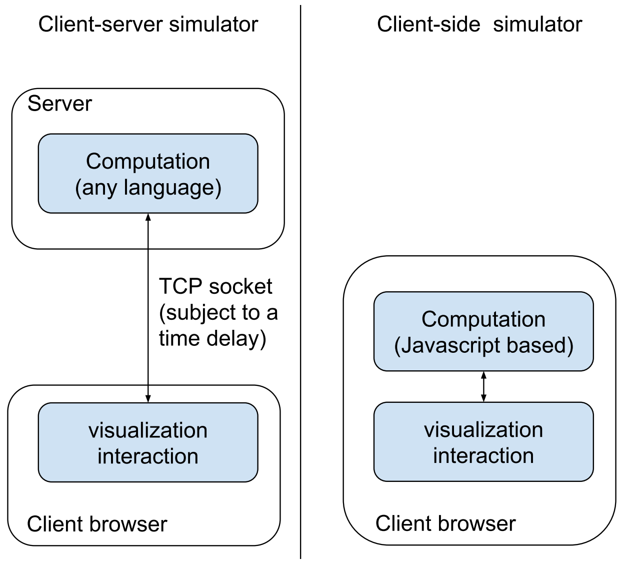
In physiological modeling, a number of modeling tools are used – e.g. Mathworks Matlab/Simulink, CellML, JSim, OpenModelica etc – and each one requires installation and at least some familiarization with the tool to be able to run the models. Some tools even require a (very expensive) commercial license. To overcome this, a standalone simulator is required, preferably without the need of installing anything. Web-based technologies do offer a convenient solution (Kofránek *et al,* 2009) and allow the simulator applications to be accessed as simply as the rest of the contemporary world-wide web.

However, development of a simulator is often a demanding task. Some effort has been put into development of web-based simulators, e.g. the proprietary Modelica.university (Tiller and Winkler, 2017) or the Bodylight framework (Ježek *et al,* 2013), based on the discontinued (Smith, 2015) Microsoft Silverlight.

Some researchers (e.g. (Christ and Thews, 2016; Kulhánek *et al,* 2013; Zhang, 2001) and a number of others) aimed at a client-server simulation. Such solution relies on a server which performs the computation and client only receives the resulting data. Based on user input, the client asks for a new set of data. The second possible approach is fully client-side, in which the client is responsible for both the computation and user interaction see Figure 1.

The client-side concept is initially more demanding task, as the whole calculation has to be performed in a web-enabled language, i.e. JavaScript, it however offers some advantages. Especially for educational purposes, the server does not have to bear entire classroom's computational load at the same time. The requirement of a smooth visualization, including continuous simulation graphing, movement of animated components and prompt interaction therefore prefers the client-side approach. Although the usage of a modern cloud technologies with scalable computational power and decentralized geographical location would reduce the client-server lag to satisfying levels, the price of the infrastructure is substantially higher and scales up with any new user. Of course, very computationally demanding simulators are not meant to be client-simulated, but those are out of scope of the discussed physiological models.

As of 2018, no open web-based simulator platform which is capable of running complex equation-based models exists. Our aim is to develop a client-side simulator technology, based on the chosen Modelica language and a simulator-producing toolchain. This technology has been named Bodylight.js



Client-server simulator
Client-side simulator
Server
Computation (any language)
TCP socket (subject to a time delay)
visualization interaction
Client browser
Computation (Javascript based)
visualization interaction
Client browser

Figure 1. Client server and Client only architectures.

2 Methods

Our group (Kofranek's group) develops medical e-learning simulators since 1996. We have focused on web-based simulators since 2012 (Ježek *et al,* 2012). After designing a set of simulators (a sample is shown on Figure 2), based on the custom Bodylight framework, built on a Microsoft Silverlight web technology (Ježek *et al,* 2013), the core Silverlight platform has been discontinued (Smith, 2015). Lessons learned – do not rely on proprietary platforms.

2.1 The Bodylight.js build process

The effort has been recently restarted, and consequently the approach has been based on open standards:

- Modelica language for modeling (Fritzson and Engelson, 1998)
- Functional Mockup Interface for model simulation
- HTML5 + JavaScript for model presentation and interaction

Driven by industrial needs to share and co-simulate models of various languages and tools, the Functional Mockup Interface (FMI) (Blochwitz *et al,* 2012) emerged as an open standard. Developed and maintained by the Modelica association (Lund, Sweden), it quickly gained wide support from tool vendors.

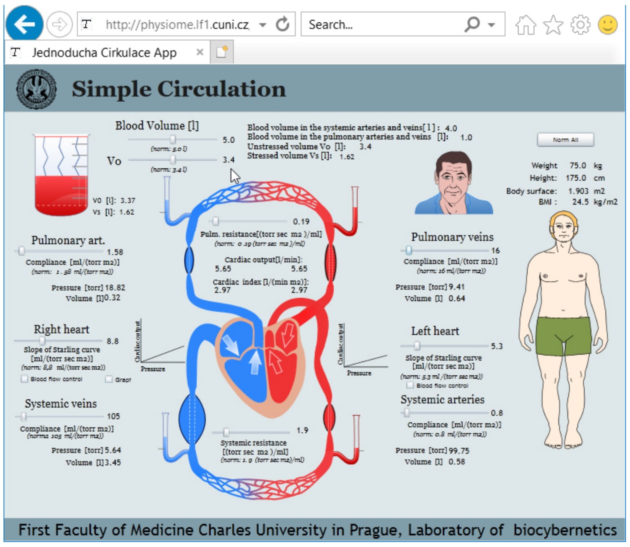


Figure 2. The simulator of simple circulation, built using the Silverlight technology (Tribula *et al,* 2013).

As of September 2018, 110 tools are capable of either FMI export, import or both (Modelica Association, 2017).

In first stage of our work-flow the model is exported as FMU for Co-simulation version 2.0 including source code. The advantage of using FMI is the standardization, which ensures further compatibility of export from multiple tools and their future versions.

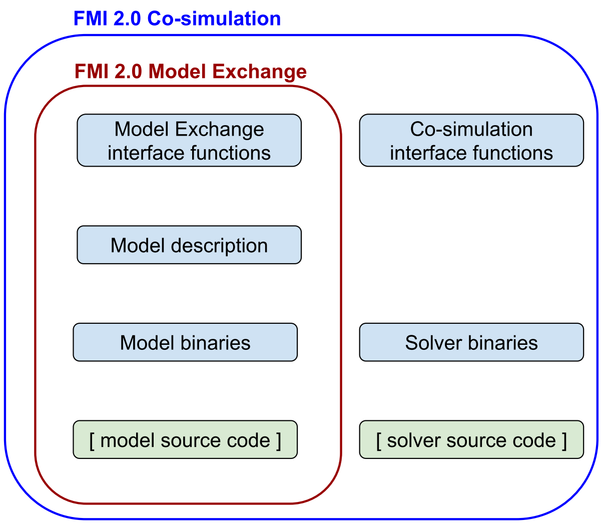


Figure 3. Content of FMU can vary, depending on usage (simplified).

The task is to get the FMU into JavaScript, so it can run in the browser. As shown in Figure 3, the FMU can contain source code of both the model and the solver. The C code could be then translated to JavaScript using Emscripten (Zakai, 2011). The Emscripten translation offers two targets: ASM.JS and WebAssembly (or WASM). Asm.js is a turing-complete subset of the JavaScript language, used as a compilation target. WebAssembly is currently a more effective binary version of Asm.js, but it is designed to "*Define* *a portable, size- and load-time-efficient binary* format *to serve as a compilation target which can* be compiled to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms, including mobile and IoT" (WebAssembly High-Level Goals – WebAssembly, n.d.). The model compilation to a binary format effectively obfuscates the model code, so this method is suitable for proprietary or undisclosable models as well. The translated FMU code is then linked to model controls (such as start, stop, parameters input etc), graphs and animated components.

For educational purposes especially in non-technical fields, the value of a graph alone is usually not enough. Simulators should provide rich content, including images and animations controlled by the model's output. Thus, the animation components are designed and animated in Adobe Animate and then exported as an HTML component, exposing their animation time-lines as Javascript functions, which are linked to the model. The animations time-lines could be nested, so it is possible to animate e.g. width and height of a component independently, but the animations have to be stackable, that is they are not truly independent. The whole web-simulator build process is visualised in Figure 4.

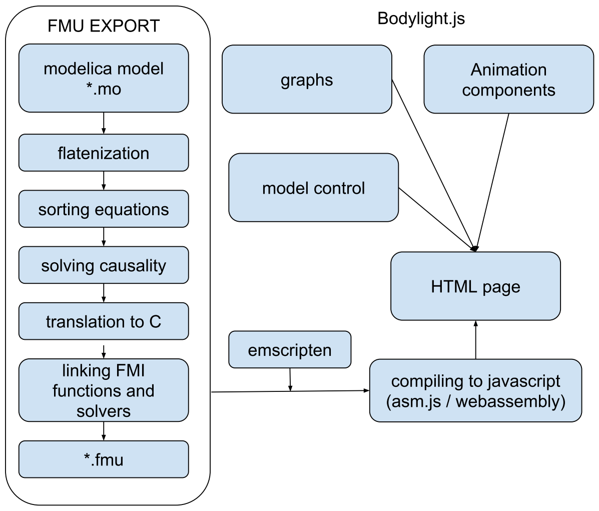


Figure 4. FMU build process. The FMU is exported and the packaged sources are translated into the JavaScript to enable web-simulation.

2.2 The Bodylight.js Composer

As illustrated by Figure 5, the development of an educational simulator is a multi-disciplinary task (Kofránek *et al,* 2009):

- The domain expert (teacher) sets the simulator objectives and designs a simulation scenario
- The modeler develops and implements the mathematical model
- The graphic designer draws and animates the components and prepares the layout
- The integrator composes the simulator together.

In fact, the simulator integration could be simplified to a bare minimum – all inputs are already known and prepared, thus it is only necessary to interconnect the controls and graphical components with the model inputs and outputs. And that could be mostly automatized. Therefore, to make the simulator development more efficient, a special helper composer tool has been developed.

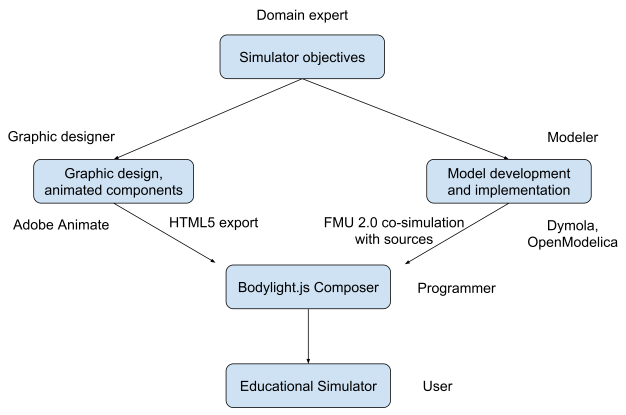


Figure 5. Simulator design and build process scheme.

The composer allows to upload the FMU, manage the model settings, upload animations and other graphical components, insert graphs and HTML controls, and interconnect it all together and then export a standalone HTML5 application.

3 Applications

The Simple Circulation simulator implemented earlier in Silverlight (Figure 2) was already reimplemented[1] using the new technology.

Another new simulator on iron regulation[2] was implemented, see Figure 6. Here, we have implemented a mathematical model of systemic iron regulation based on the work of Enculescu et al. (Enculescu *et al,* 2017). The model incorporates dynamics of organ iron pools as well as regulation by the hepcidin/ferroportin system. The model was calibrated and validated with time-resolved measurements of iron responses in mice challenged with dietary iron overload and/or inflammation. The model demonstrates that inflammation mainly reduces the amount of iron in the bloodstream by reducing intracellular ferroportin transcription, and not by hepcidin-dependent ferroportin protein destabilization. In contrast, ferroportin regulation by hepcidin is the predominant mechanism of iron homeostasis in response to changing iron diets for a big range of dietary iron contents.

3.1 Nephron simulator

A nephron simulator[3] was implemented recently. Nephron is the structural and functional unit of the

[1] at www.physiome.cz/apps/SimpleCirculation/
[2] at www.physiome.cz/apps/IronMetabolism
3 at www.physiome.cz/apps/Nephron/

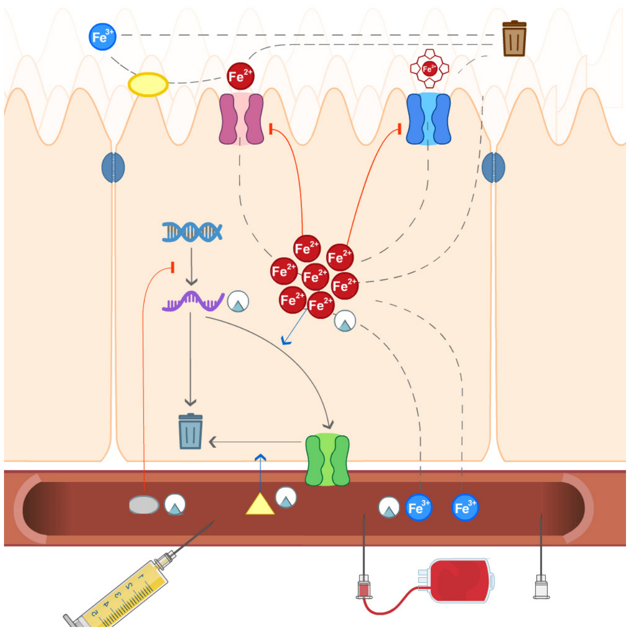


Figure 6. Iron regulation. The present screen shows the basic iron metabolism in duodenal cells. The dashed lines symbolize iron transfer between cell compartments and blood stream. It is possible to regulate food iron income as well as blood transfusion and loss. Further it is possible to initiate an inflammation process via injection of lipopolysacharide (LPS) into to blood vessels.

kidney. In the following text we explain some of the kidney's main functions and present the model behind the simulation application. Finally we present the application itself, which is composed of several consecutive simulation screens, and discuss how it is used to clarify physiological processes.

3.1.1 Basic kidney functions

The urinary system is comprised of two kidneys connected via ureters to the urinary bladder, and an urethra. The kidneys produce urine containing excess water, electrolytes and body waste products. The urine then flows down the ureter into the bladder where it is temporarily stored. The bladder is then reflexively emptied via the urethra.

The kidney has many important homeostatic, hormonal, and metabolic functions making its (patho) physiology very complex and difficult for medical students to comprehend. To mention several of these:

1. The water balance and electrolyte homeostasis.
2. The regulation of acid-base balance.
3. Excretion of metabolic waste products, especially the toxic nitrogenous compounds and xenobiotics.
4. Production of renin enzyme for arterial blood pressure control and erythropoietin, which stimulates red blood cell production in the red bone marrow.
5. Conversion of vitamin D into an active form for the regulation of calcium balance.

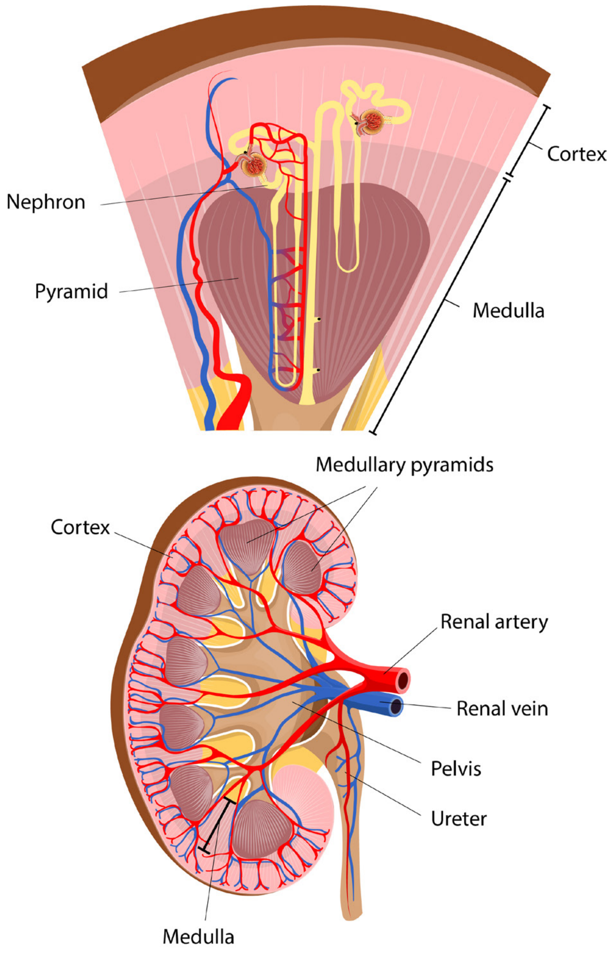


Figure 7. Kidney anatomy

Anatomically a kidney is composed of two layers: an outer layer named cortex and an inner layer called medulla. The medulla contains multiple cone-shaped lobes, known as medullary pyramids. The urine drains into the renal pelvis, which is the initial part of the ureter. The hilum of the kidney is the site of entry and exit for renal artery, renal vein, and ureter.

There are two types of nephrons (Figure 7) differing in length and urine concentration capacity: short cortical (70 – 80%) and long juxtamedullary (20 – 30%) nephrons, in total there is about one million nephrons in each kidney. The nephrons begin in the cortex; the tubules descend down to the medulla, then make a U-turn and return to the cortex before draining into the collecting duct. The collecting ducts then descend towards the renal pelvis and empty the final urine into the ureter.

Each nephron has the following parts (Figure 8):

1. Glomerulus and Bowman's capsule (Figure 9).
2. Proximal tubule.
3. Loop of Henle (descending and ascending parts).
4. Distal convoluted tubule.
5. Collecting duct.

Throughout the length of the nephron, peritubular capillaries lie adjacent to all segments of the tubule and help to maintain the dynamic stationary state. The capillaries originate from the efferent glomerular arteriole and remove the water and solutes excreted by the tubules.

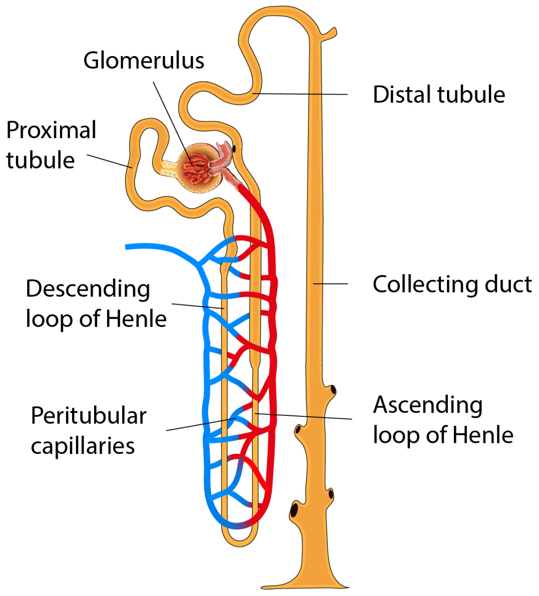


Figure 8. Nephron

The present application focuses on the urine production in terms of water and sodium ion (Na^+) only. Prospectively we intend to extend the model by including more solutes and kidney processes in future.

3.1.2 Physiology and models

There are two separate models used in the application. One for the glomerulus simulator (Figure 10) and one for the other (nephron tubules) simulators (Figure 11). Both glomerulus and nephron tubule models utilize the PhysioLibrary (Matejak and Kofranek, 2015), which was also developed within our group. The models are considered to be in the steady state and the model does not take temporal evolution into account.

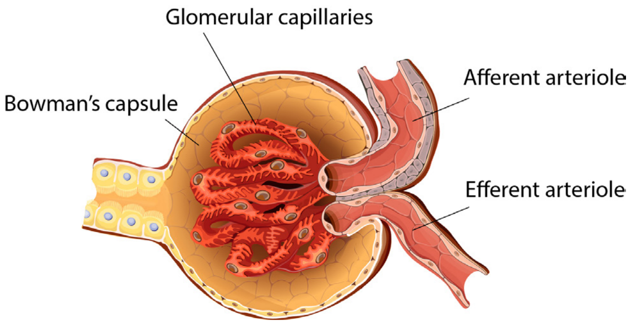


Figure 9. Glomerulus

Glomerulus

Blood enters the afferent arteriole and flows into the glomerular cluster of intertwined capillaries buried within the Bowman's capsule. The blood leaves the glomerulus through the efferent arteriole.

The wall of the glomerular capillaries is penetrated with many microscopic slits through which the fluid and small solutes passes into the Bowman's capsule. The size of the filtration slits restricts the large molecules from being filtered out the plasma (such as protein albumin/ globuline) and cells (such as erytrocytes or leucocytes).

The process of the glomerular filtration is often called renal ultrafiltration. The force of the hydrostatic pressure in the glomerulus (the force of the pressure exerted from the pressure of the blood vessel itself) is the driving force that pushes the filtrate out of the capillaries.

The osmotic pressure (the pulling force exerted by the albumins) works against the greater force of the hydrostatic pressure, and the difference between the two determines the effective filtration pressure and the glomerular filtration rate (GFR), along with a few other factors.

GFR is physiologically kept constant for a wide interval of arterial pressure. The hydrostatic pressure can also be controlled by widening (vasodilation) or narrowing (vasoconstriction) the afferent and efferent arterioles.. The glomerulus model was implemented from scratch. It utilizes the hydraulic domain (connector composed of *pressure* and flow *VolumeFlowRate* variables) of PhysioLibrary. The model is analogy of electrical voltage divider. The model represents all the glomeruli contained in pair of kidneys, e.g. the flows in the model are summed up over all nephrons.

Resistance of the glomerular capillary wall is modeled with the *filterResistance* component. Afferent and efferent arterioles are modelled with the *afferentResistance* and *efferentResistance* components. The two resistances are variable and affect the pressure on input of *filterResistance* and thus flow through it (which is GFR). The osmotic pressure is included simply by adding two extra pressure columns (*osmoticBlood* and *osmoticUrine*) around *filterResistance* component.

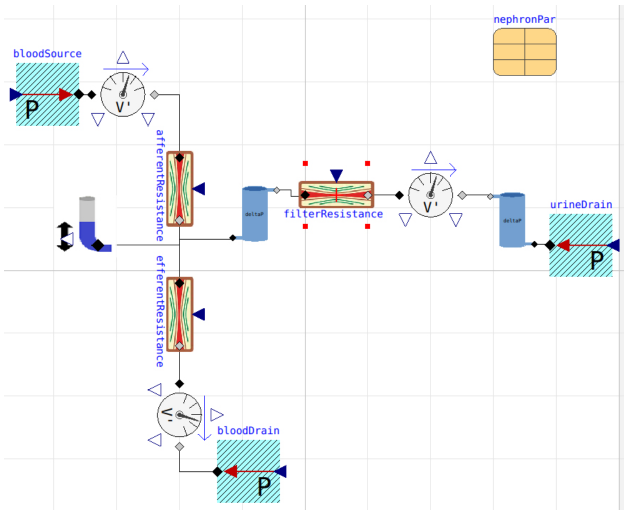


Figure 10. Glomerulus model, analogy of electrical voltage divider: afferent and efferent resistances control pressure on the left connector of filter resistance. Osmotic pressure is modeled by pressure columns.

Tubules

The nephron tubule model utilizes the osmotic domain (connector composed of *Concentration* and flow *VolumeFlowRate* variables) of the PhysioLibrary. It is implemented according to (Hoppensteadt and Peskin, 1992), but there are several changes and extensions. It is basically a system of ODE in a space coordinate. This coordinate is manually discretized so that fields are replaced with arrays and derivatives with forward differences. The model is composed of several tubules components/classes. All tubule models extend from common general tubule with this equations:

$$\begin{aligned} \frac{dQ}{dx} + f_{H_2O} &= 0 \\ \frac{d(Qo)}{dx} + f_{Na} &= 0 \end{aligned} \tag{1}$$

where Q is filtrate volumetric flow [m^3s^{-1}] through the tubule, o is osmolarity of the filtrate [mOsm L^{-1}], f_{H2O} water volumetric flow through the tubule wall in the outer direction per unit length [m^2s^{-1}] and f_{Na} similarly the Na molar flow through the tubule wall in the outer direction per unit length.[mmol $s^{-1}m^{-1}$]. The particular tubules differ with the equations for f_{H2O} and f_{Na} according to the tubule function.

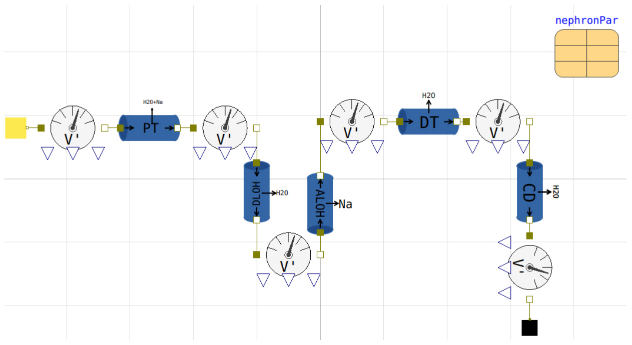


Figure 11. Nephron model is composed of UnlimitedVolume source as a simplified glomerulus and a sequence of tubules (proximal tubule, descending loop of Henle, ascending loop of Henle, distal tubule, collecting duct)

Proximal tubule

The proximal tubule is the major resorptive segment of the nephron and accounts for resorption of nearly two-thirds of all filtered water and sodium. The water is reabsorbed along with all the dissolved sodium, so that the filtrate osmolarity is preserved. The additional equations in this component are

$$\begin{aligned} f_{H_20} &= k_{H_2O} \\ o &= o_{in} \end{aligned} \tag{2}$$

where k_{H2O} is chosen so that ⅔ of water is reabsorbed in the proximal tubule under the normal GFR. oin is input osmolarity of the proximal tubule.

Descending Loop of Henle

The descending Loop of Henle displays a high permeability to water but is virtually impermeable for sodium. The osmolarity of medulla surrounding the tubule rises from 300 near the cortical layer down to 1200 mOsm/l deep in medullary layer. The water leaves passively the tubule so that the osmolarity in the tubule equalizes with osmolarity of the ambient medulla. Approximately 20% of water is reabsorbed here.
Additional equations are

$$\begin{aligned} o &= o_{med} \\ f_{Na} &= 0 \end{aligned} \tag{3}$$

where omed is an array. Its value rises linearly with the index to model the medulla osmolarity gradient.

Ascending Loop of Henle

The ascending loop of Henle accounts for resorption of nearly a quarter of the filtered load of sodium. It is virtually impermeable to water. Given the large amount of solute resorption that occurs in the absence of water resorption, the tubular fluid becomes progressively dilute as it travels through the ascending loop. This feature is why this segment is frequently referred to as the “Diluting Segment” of the nephron. The resorption is active and consumes energy in form of ATP. This enables lower osmolarity in the duct compared to surrounding medula, but the difference can’t be higher than 200 mOsm/l. The osmolarity drops down to 100 mOsm/l in the duct. The equations are

$$\begin{aligned} f_{H_20} &= 0 \\ f_{Na} &= \text{limit}(k_{Na}, o_{med}) \end{aligned} \tag{4}$$

where kNa is chosen to meet the osmolarity 100 mOsm/L at the outflow of the loop of Henle under normal condition. The limiter function ensures that the osmolarity difference does not exceed 200 mOsm/L namely at decreased GFR.

Distal tubule and collecting duct

The distal tubule and the collecting ducts represent the final functional segment of the nephron after which any remaining tubular fluid is excreted as the final urine. By this segment, the vast majority of solutes and water is resorbed and thus the late distal tubule and collecting ducts are responsible only for a small fraction of total resorption. However, this represents the major locus of regulated tubular resorption and given the enormous quantities of glomerular filtration that occur per minute, even small changes in resorption rates at this segment can have enormous impacts on the composition of the body’s extracellular fluid.

The distal tubules of several nephrons empties into one shared collecting duct.

The distal tubule and collecting duct system is under the control of antidiuretic hormone (*ADH*). When *ADH* is present, the tubules becomes permeable to water. The high osmotic pressure in the medulla (generated by the counter-current multiplier system/loop of Henle) then passively draws out water from the tubules to medulla and blood vessels drain it away. Than the final urine osmolarity is about 1200 mOsm/l and as much as possible water is retained in body.

With no *ADH*, tubule walls are impermeable to water, no water is reabsorbed. The urine osmolarity is 100 mOsm/l (as it leaves ascending loop of Henle) and the body loses water rapidly. (4)

The equations of both distal tubule and collecting duct are

$$\begin{aligned} f_{H_2O} &= ADH \cdot q_{H_2O}(o_{med} - o) \\ f_{Na} &= 0 \end{aligned} \qquad (5)$$

Where *ADH* ϵ (0,1) and q_{H2O} is constant.

3.1.3 The simulators

The simulator is composed of several screens, each for a section of the nephron. All screens contain besides others sliders to control some model parameters and plots visualising usually flow and osmolarity along tubules. But these are not shown on the following screenshots.

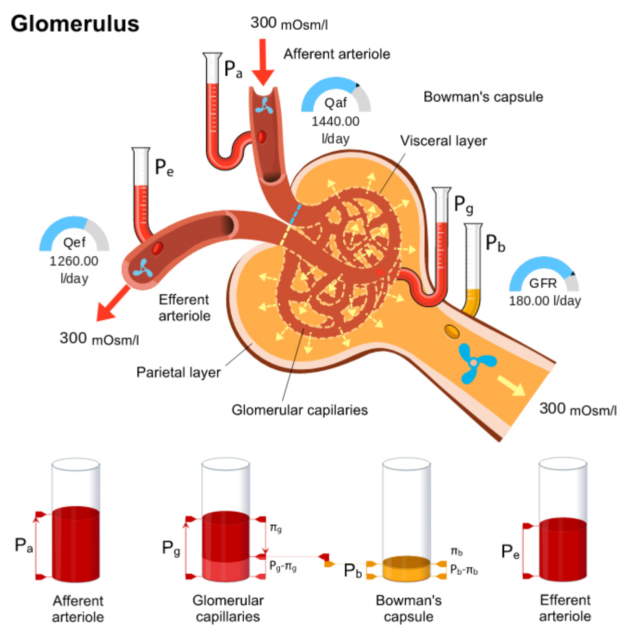


Figure 12. Top: Bowman's capsule (yellow), afferent and efferent arterioles. Red arrows symbolize the blood flow direction, yellow arrow represents urine flow direction. For each part of glomerulus both the hydrostatic and osmotic pressure is calculated. The difference between the total effective pressure in capillaries and in the Bowman's capsule drives the net filtration flow.

Glomerulus

Figure 12 depicts the glomerulus screen. Pressures are depicted with the liquid-column gauge, flows through the tubules with the half-circle measures and turbines, flow through the vessel walls with width of the dashed moving arrows. This visualized sensors are used also in all other parts of the application.

Student can change resistance of afferent and efferent arteriole and thus control the hydrostatic pressure in glomerular capillaries and GFR. Mean arterial pressure (pressure at afferent arteriole entry) may be also modified. The filtration resistance may be controlled as well to simulate certain renal pathologies. The goal is to explain how the glomerulus maintains constant GFR despite changing arterial pressure by means of changing afferent and efferent arteriole resistance.

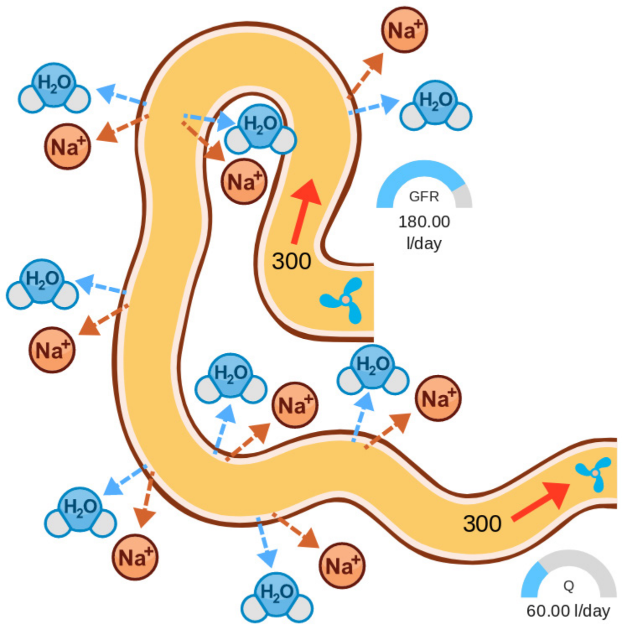


Figure 13. Diagram of the proximal loop. The red arrows show the direction of the urine flow. Along the proximal tubule sodium ions and water molecules are reabsorbed across the cellular boundary.

Proximal tubule

Proximal tubule is shown on Figure 13. The number inside the tubule depicts the osmolarity of the filtrate. GFR may be controlled. Decrease of flow may be observed on the flow measures whereas the osmolarity is maintained.

Loop of Henle

Loop of Henle is shown on Figure 14. Ascending and descending tubules of Loop of henle are together on one screen. Student can control the GFR and observe changes in reabsorption rates, flow and osmolarity. The sodium transport limiter is applied in the ascending section so the osmolarity never drops below 100mOsm/l at the outflow.

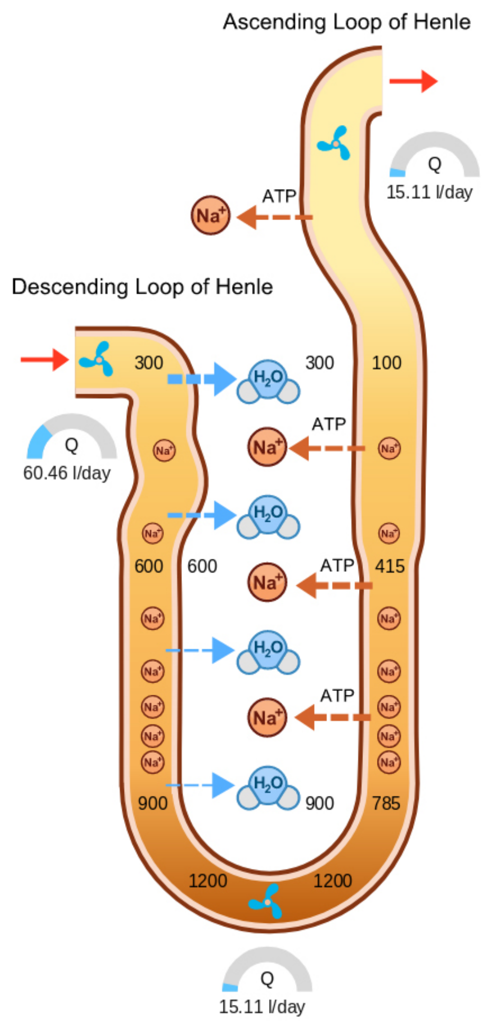


Figure 14. Loop of Henle. Red arrows show the urine flow direction. Water molecules passively leave the urine making it more concentrated from physiological 300 mosm/l down to 1200 mosm/l in the descending tubule. The blue dashed arrows indicate the amount of water molecules transfer from tubulus along its osmotic gradient. The ascending tubule is virtually impermeable to water molecules while sodium ions are actively pumped outside the tubulus. The width of the purple arrows indicates the amount of the sodium reabsorption.

Distal tubule and collecting duct

Figure 15 depicts the distal tubule and collecting duct. The amount of ADH may be controlled by the student. Higher permeability of the tubule wall for water is represented by widening of the blue water channels. The goal is to explain how ADH affects the reabsorption: With no ADH, there is no reabsorption. Filtrate goes through unaffected. Urine production rate is high and its osmolarity is low. With full ADH so much water is reabsorbed that the urine osmolarity equalizes with osmolarity of the surrounding medulla. Only small amount of highly osmotic urine is produced.

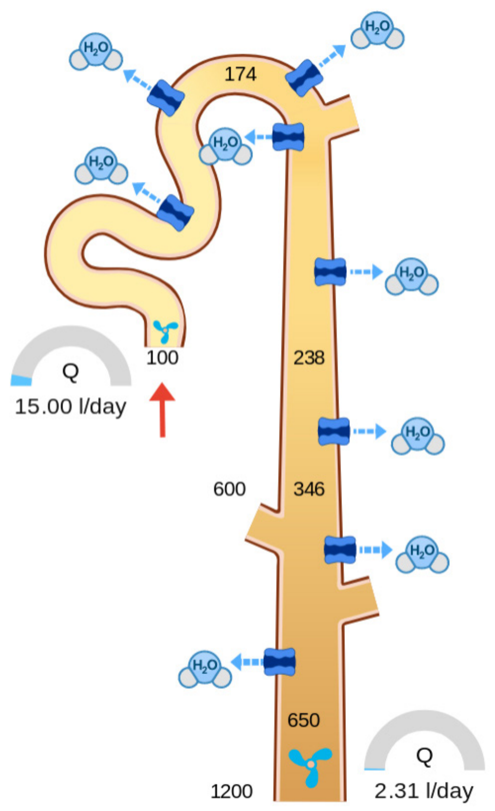


Figure 15. Distal tubule and collecting duct represent the last two segments of a nephron. The distal tubule and the collecting duct are permeable for water molecules, however, the net flow across the tubule wall is endocrinologically regulated via ADH. As a result the body is able to fine tune the urine osmolarity according to the circumstances.

Complete nephron

Figure 16 shows a complete nephron. All the information from previous screens is recapitulated here. For simplicity the GFR is controlled directly instead of controlling afferent and efferent arteriole resistance as it was in the glomerulus section. ADH is controlled as well.

4 Discussion and conclusion

New framework Bodylight.js for building web-based simulators was presented. Bodylight.js will be available for public use, but it is still under heavy development and not ready for widespread deployment. We encourage interested parties to contact us and we will gladly provide access and documentation. We welcome any feedback and code contributions others can provide. The Bodylight.js was already used to compose three teaching simulators and proved to be really useful. One of them, the Nephron simulator was presented within this paper. This simulator will be used in physiology lectures at our faculty and will be updated according to the feedback from teachers and students alike. Physiological model for this simulator was developed as

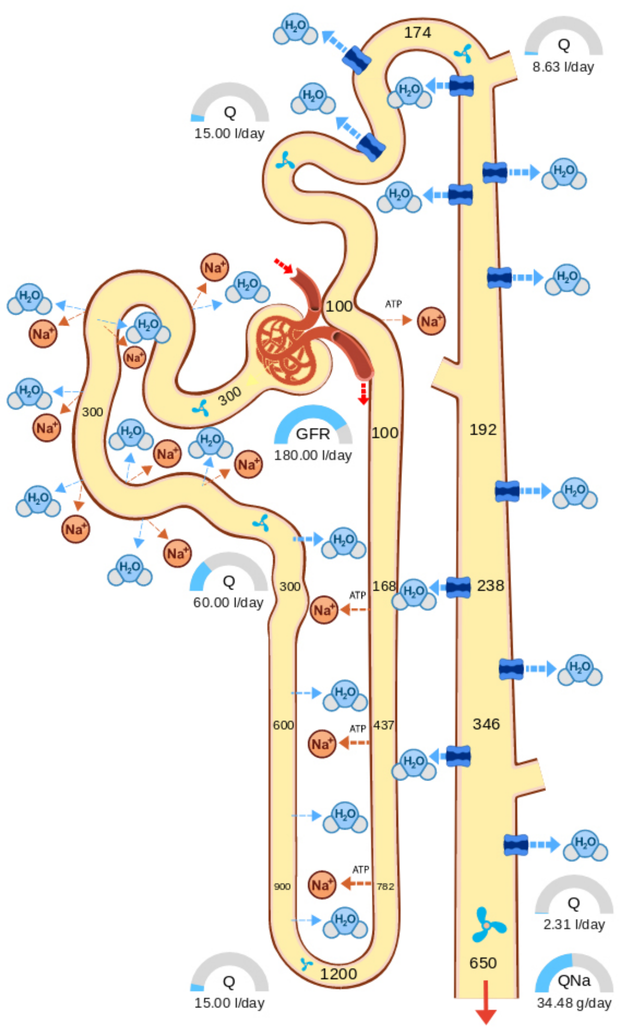


Figure 16. Screen of the whole nephron model, i.e. glomerulus, proximal tubule, descending and ascending loop of Henle, distal tubule and collecting duct. Besides others, the mass flow rate of excreted Na is displayed.

well. Results of the model were checked by physiologists and are approximately correct, enough for the teaching purposes. We plan to add a follow-up simulator including more solutes and additional regulation mechanisms.
We have built an extensive library of simulators with the previous, now defunct, Silverlight technology, covering large portions of physiological systems. We hope that by relying on standardized web technologies we can provide a plethora of new and future-proof web based teaching applications.

Acknowledgement

This work was supported by the TRIO MPO FV20628, SVV260371, PROGRESS Q26 grants and the Creative Connections s.r.o.. We thank Martin Brož for the artwork.

References

Blochwitz T, Otter M, Akesson J, 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*; September 3–5; 2012; Munich; Germany, 2012, pp. 173–184. Linköping University Electronic Press. Available at: http://www.ep.liu.se/ecp_article/index.en.aspx?issue=076%20;article=017.

Christ A and Thews O (2016) Using numeric simulation in an online e-learning environment to teach functional physiological contexts. *Computer methods and programs in biomedicine* 127: 15–23. DOI: 10.1016/j.cmpb.2016.01.012.

Enculescu M, Metzendorf C, Sparla R, et al. (2017) Modelling Systemic Iron Regulation during Dietary Iron Overload and Acute Inflammation: Role of Hepcidin-Independent Mechanisms. *PLoS computational biology* 13(1): e1005322. DOI: 10.1371/journal.pcbi.1005322.

Fritzson P and Engelson V (1998) Modelica—A unified object-oriented language for system modeling and simulation. In: *European Conference on Object-Oriented Programming*, 1998, pp. 67–90. Springer. Available at: https://link.springer.com/chapter/10.1007/BFb0054087.

Hoppensteadt FC and Peskin CS (1992) *Mathematics in Medicine and the Life Sciences*. DOI: 10.1007/978-1-4757-4131-5.

Ježek F, Privitzer P, Mateják M, et al. (2012) Demonstration of the Risk of Fixed Ejection Volume in Ventricular Assist Devices in Small Patients Using Web Simulator. In: *5th European Conference of the International Federation for Medical and Biological Engineering*, 2012, pp. 489–492. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-23508-5_127.

Ježek F, Tribula M, Kolman J, et al. (2013) Sada výukových simulátorů – výsledky vývoje frameworku bodylight. *MEDSOFT 2013: sborník příspěvků*: 38–48. Available at: http://www.medvik.cz/link/bmc13015203.

Kofránek J, Privitzer P, Matoušek S, et al. (2009) Schola Ludus in Modern Garment: Use of Web Multimedia Simulation in Biomedical Teaching. *IFAC Proceedings Volumes* 42(12). Elsevier: 413–418. DOI: 10.3182/20090812-3-DK-2006.0087.

Kofranek J, Matousek S, Rusz J, et al. (2011) The Atlas of Physiology and Pathophysiology: Web-based multimedia enabled interactive simulations. *Computer methods and programs in biomedicine* 104(2): 143–153. DOI: 10.1016/j.cmpb.2010.12.007.

Kulhánek T, Mateják M, Šilar J, et al. (2013) Hybridní architektura pro webové simulátory. *MEDSOFT 2013: sborník příspěvků*: 115–121. Available at: http://www.medvik.cz/link/bmc13015212.

Matejak M and Kofranek J (2015) Physiomodel – an integrative physiology in Modelica. *Conference proceedings: ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference* 2015: 1464–1467. DOI: 10.1109/EMBC.2015.7318646.

Modelica Association (2017) Tools | Functional Mock-up Interface. Available at: http://fmi-standard.org/tools/ (accessed 21 July 2017).

Smith J (2015) Moving to HTML5 Premium Media – Microsoft Edge Dev Blog. Available at: https://blogs.windows.com/msedgedev/2015/07/02/moving-to-html5-premium-media/ (accessed 27 August 2018).

Tiller MM and Winkler D (2017) modelica.university: A Platform for Interactive Modelica Content. In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic*, May 15–17, 2017, 4 July 2017, pp. 725–734. Linköping Electronic Conference Proceedings. Linköping University Electronic Press. DOI: 10.3384/ecp17132725.

Tribula M, Ježek F, Privitzer P, et al. (2013) Webový výukový simulátor krevního oběhu. *MEDSOFT 2013: sborník příspěvků*: 197–204. Available at: http://www.medvik.cz/link/bmc13015231.

WebAssembly High-Level Goals – WebAssembly (n.d.). Available at: https://webassembly.org/docs/high-level-goals/ (accessed 24 September 2018).

Zakai A (2011) Emscripten: An LLVM-to-JavaScript Compiler. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, New York, NY, USA, 2011, pp. 301–312. OOPSLA '11. ACM. DOI: 10.1145/2048147.2048224.

Zhang S (2001) *An IIOP architecture for Web-enabled physiological models*. Massachusetts Institute of Technology.

Session 4A: Power & Energy 3

Platform for Microgrid Design and Operation

Johan Windahl[1] Håkan Runvik[1] Stéphane Velut[1]

[1]Modelon, Sweden, {johan.windahl,hakan.runvik,stephane.velut}@modelon.com

Abstract

This paper describes the development and requirement specification of a platform for design and operation of microgrids.

The goal is to have a flexible platform based on open standards that can be used to efficiently solve current and future engineering problems for distributed energy sources and storage systems. By basing it on a unified architecture, collaboration and efficient work flows are enabled.

In this work we investigate the requirements on the model and on the tool side. We also demonstrate how an energy storage system can be designed to reduce the maximum peak power and how it can be operated in the most economic efficient way, taking into consideration constraints and limitations of the system.

This work is based on Modelon's web-based modeling and simulation platform and its Modelica library Microgrid.

Keywords: simulation, optimization, peak shaving, battery storage, energy management, economic dispatch

1 Introduction

Environmental considerations and increasing awareness of infrastructure sensitivity have led to reconsiderations of how energy systems should be best configured. The current highly centralized systems which were developed for large production units such as nuclear and fossil power plants are not suitable for renewable, intermittent and distributed energy sources such as wind and solar (Fathima and Palanisamy, 2015). This motivates the use of microgrids, which are specifically developed for this kind of heterogenous energy production.

A microgrid is a group of interconnected energy sources, loads and storage devices that can operate both connected with the surrounding electricity grid and disconnected in islanding mode. It has the potential to offer increased self-sufficiency and reliability at low cost and reduced environmental impact (Eto *et al*, 2018). Microgrids typically include smaller production units such as photovoltaic arrays, wind turbines, microturbines and generators (combustion engines) as well as storage devices such as flywheels and batteries. Their comparably smaller investment cost makes them attractive to install in remote areas and their capacity for reducing transmission congestion makes them interesting for energy suppliers (Venkatraman and Khaitan, 2015).

2 Background

The challenges of microgrid design and operation are attracting considerable research interest. A survey of these, including islanding transitions, power quality improvements and economic optimization can be found in (Venkatraman and Khaitan, 2015). Optimization is an important tool in this regard. (Fathima and Palanisamy, 2015) contains an overview of the different optimization problems considered in the current research, and the methods and tools used to solve them.

A review of previous work reveals that although there are plenty of methods and tools developed to solve specific problems in this domain, there is no unified tool-chain capable of handling all the relevant problems in both design and operation. Common tools used for microgrid optimization and simulation today include HOMER (HOMER Energy, 2018) and PSCAD (Manitoba Hydro International 2018). HOMER (Hybrid Optimization of Multiple Energy Resources) is a commercial tool originally developed by the National Renewable Energy Laboratory (NREL), that can be used for optimization of the system configuration, component sizing and grid operation. PSCAD is a time-domain based power system simulation tool, which can be applied for microgrid configurations. Tools such as these are quite specialized, lacking the needed flexibility of a complete solution, such as

- changing fidelity levels in the system representation
- introducing highly customized component models in specialized systems
- customizing optimization formulations for specific needs
- combining the physical domains such as thermal, electrical, mechanical to describe future heterogeneous cyber-physical systems

Alternative, highly customized implementations based on more generic tools such as GAMS (General Algebraic Modeling System) (GAMS, 2018) also exist, but the work required to setup such systems makes them impractical for the common user.

A Modelica-based solution is a good candidate for filling this gap, enabling a flexible and user-friendly framework for the design and operation task. Previous microgrid solutions based on Modelica include (Du *et al*, 2014), where two optimization problems for a Modelica microgrid model are solved using Matlab and FMI-toolbox. In (Kehl *et al*, 2017), a heterogeneous modeling process is proposed, where Modelica models are coupled with Simulink for control design. Additional work focusing only on the Modelica modeling also exist, such as (Roy *et al*, 2014) and (Enerbäck and Nalin Nilsson, 2013). None of these present a platform for modeling, simulation and optimization, enabling the user to solve many of the problems of microgrid design and operation with one tool. Presenting the development and requirements of such a tool is the goal of this paper.

2.1 Design and operation

Design and operation are different types of engineering tasks that typically occur at different places in time using different types of tools of various fidelity levels and time scales. But the design and operation are tightly coupled and would benefit from a unified tool-chain. As an example, a design that requires an aggressive use of the energy storage system may shorten the life-time and lead to a higher total cost compared to a different design. Another example is peak reduction, a design that cannot reduce the peak power may lead to higher costs. In (Fathima and Palanisamy, 2015), the challenges encountered in this regard are divided into generation, control and distribution side optimization problems. Generation side problems consider system design and component sizing, control side problems describe phenomena such as voltage and frequency control and the distribution side considers scheduling and dispatching. The performance indicators and optimization objectives are typically based on economic, environmental or reliability considerations (Luna-Rubio *et al*, 2012). Typical problem formulations are:

1. Which system configuration has the lowest capex and opex cost, for a given load profile and ambient conditions.
2. What is the optimal component sizing under certain reliability requirements.
3. What is the optimal economic dispatch, taking electricity, fuel, maintenance and aging costs into account.

Only a subset of the possible questions can be answered by one single tool, but by providing a flexible framework that can be used by multiple user types, the potential impact is maximized.

3 Requirements

A challenge with designing a model-based framework that should empower different user personas performing various tasks and analyses is the variety of aspects and requirements to consider. Requirements cover the range from model fidelity level, interface design and numerical robustness to user-friendly work-flows, application interface and integration of input data such as solar irradiation and economy parameters.

Here we will look closer into the technical aspects of analysis that are required and the kind of model fidelity that needs to be supported.

3.1 Analyses

A framework that should support both design and operation needs to cover various types of analyses.

System design focuses on the overall system behavior. It requires support of:

- Sizing of components
- Configuration evaluation
- System evaluation

Operation covers a wide area from energy management where energy producers and storage should be used in an optimal way, to grid robustness assuring a stable grid that meets grid code requirements.

For operation we identity following basic analysis types that a tool should support:

- Economic dispatch
- Control design

"Economic dispatch is the short-term determination of the optimal output of a number of electricity generation facilities, to meet the system load, at the lowest possible cost, subject to transmission and operational constraints" (Wikipedia – Economic Dispatch, 2018).

3.2 Technology

The technology needs to be able to support the analyses described in chapter 3.1. In the center there is a model with the right fidelity level to accurately describe the system.

The different analysis types correspond to different computational execution tasks which also set requirements on the tool. The following tasks needs to be supported:

- Simulation
- Multi-simulation
- Optimization

Besides simulation, which is a basic requirement for analyzing the system behavior and to support control design, multi-simulation is required to quickly evaluate configurations. Parallelization of simulations is not a hard requirement but will improve the performance and user experience. Multi-simulation is also needed to perform sensitivity analysis in the presence of uncertainty related to weather or load forecast.

Optimization is required for component sizing and to solve the economic dispatch problem.

3.2.1 Model

Component sizing and control design have different requirements on complexity and time scale of interest. See Figure 1 for an overview of dynamic phenomena and their corresponding time range in an electrical system.

A disadvantage of using a higher complexity level than required is that it results in larger simulation times and potentially also numerical robustness issues. Another issue is that a higher model complexity often has a more detailed parameterization that increases the barrier to get started.

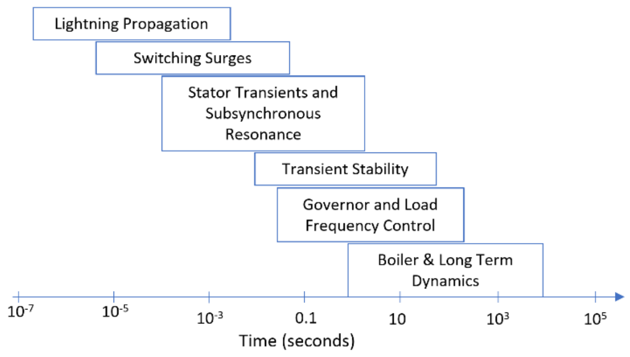


Figure 1 Time ranges of dynamic phenomena. (Sauer et. al, 1997)

One strength of Modelica is the support of generic data types that makes it possible to define a replaceable architecture, where a user can switch the complexity level. Examples of work that take advantage of this in the electrical domain are a multi-level library for electrical machines (Giangrande, *et al*, 2014) and an electric power library that covers arbitrary phases and transforms in one generic framework (Franke and Wiesmann, 2014).

In the different analyses, we categorize model requirements in following two groups:

1. System design and energy management. Time scale of interest from minutes to hours where a simulation case can cover a whole year. Here the main interest is energy flows including production, consumption and storage. Detailed electrical behavior is not of interest with an assumption of a balanced grid that neglects fast electrical transients.
2. Grid robustness. Time scale of interest from milli-second to minutes where a typical simulation may be a few seconds to a few minutes. The main interest is transient electrical behavior with a focus on grid frequency and voltage stability.

For use case 1, it should be possible to efficiently apply optimization. This requires that the model equations are at least twice continuously differentiable (Nocedel and Wright, 2006)

3.2.2 Interface

Modelica models are typically implemented to describe physical properties such as geometry and its corresponding fundamental physical equations, see. e.g. Modelica Standard Library (Modelica Libraries, 2018).

But for models to be used for system design decisions which often include economic aspects, economic information also needs to be included. Examples of economic data are nominal lifetime, capital cost and maintenance cost and interval. This information could be provided in e.g. a Modelica record.

Boundary conditions and input data of importance are:

- Weather data for renewable producers
- Electric load defining the consumption
- Fuel and electricity prices

It would be beneficial to have an interface that handles this type of location-dependent data in a user-friendly way, e.g. by integration with a web-map such as Google maps or Bing.

3.3 Collaboration

A general requirement for model-based development that is also true for micro-grid application is the ability to easily share models and results. Different tasks are typically done by people with different roles. Examples include managers that want to supervise an ongoing new design, and sales personnel that want to find a suitable configuration for a customer by simulating various configurations and present the most appropriate one.

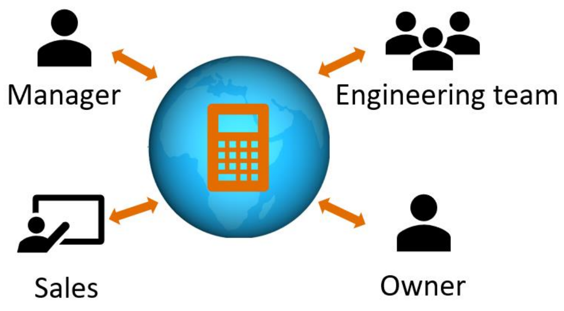


Figure 2 Illustration of a model centric collaboration approach.

Various tasks require different levels of model abstraction and exposure of model information. A sale person may not need to know about the details of Modelica or numerical integration algorithms. Instead he or she may only require an accessible and easy to use interface where a sub-set of all parameters and outputs is shown.

4 Computational Platform

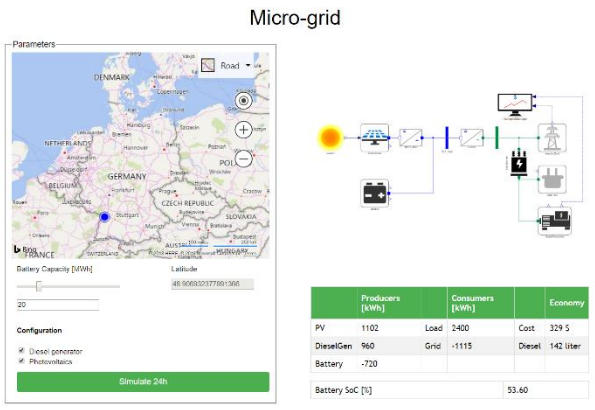
In this chapter we present our framework for the design and operation of micro-grids.
The solution consists of several parts to meet the wide variety of needs and requirements. In the center of the framework is a Microgrid Modelica library and a web modeling and simulation platform from Modelon, including the Optimica Compiler Toolkit for model simulation and optimization.

The following artefacts define the micro-grid framework:

- Modelica models are used to describe the behavior of the components and include data structures with economic data
- Optimica, an extension of Modelica (Åkesson, 2008), is used to formulate optimization problems
- Python scripts are used for optimization workflows and post processing
- HTML and JavaScript are used to create a customizable end-user interface
- FMI is used for deployment.

4.1 Software and Tools

A web-platform matches well with the requirements in chapter 3. It offers an accessible solution that supports a model centric collaboration approach. The web interface makes it possible to create various abstraction levels for a model. This is used to fulfill the different simulation needs of e.g. engineering teams and sales organization.



	Producers [kWh]		Consumers [kWh]		Economy
PV	1102	Load	2400	Cost	329 $
DieselGen	960	Grid	-1115	Diesel	142 liter
Battery	-720				

Battery SoC [%]	53.60

Figure 3 Simulation web-interface with integration of Bing map into Modelon's web-based modeling and simulation platform web-API.

Another aspect of a web simulation platform is scalability, where simulations can be distributed and executed on the cloud.

Modelon's new web-based modeling and simulation platform builds on open standards. An overview of the architecture is seen in Figure 4. The platform has support for various open formats and programming languages that have been used to create a framework that empowers the user. The following concepts have been used:

1. **View -** with views it is possible to add a visualization layer above the model. The separation makes it possible to create custom views targeting different users using a single model.
2. **Custom function** – using a custom function a python-based workflow can be associated with a model, see Figure 5. In this work it has been used to integrate dynamic optimization in a user-friendly way into the tool.
3. **Web-API**. A web simulation interface has been used to create a customizable end-user interface targeting sales personnel needing an easy-to use simulation tool, see Figure 3.

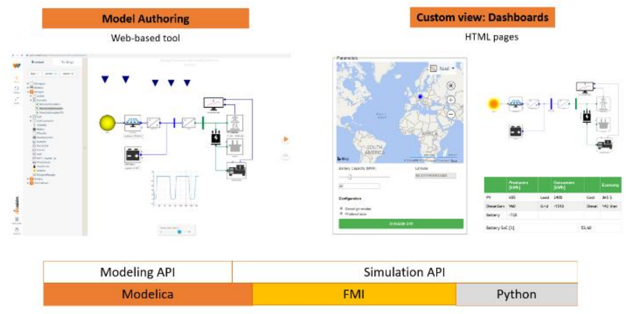
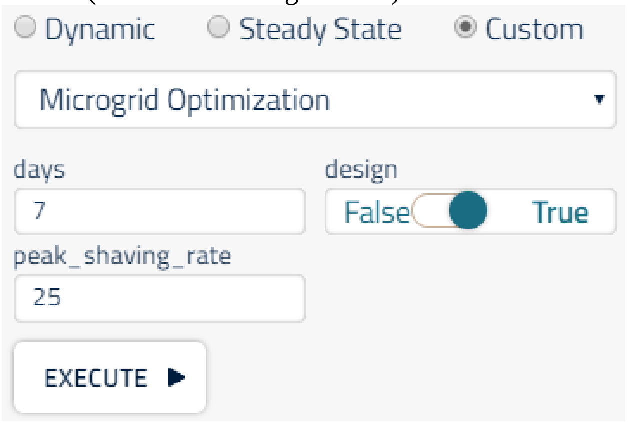


Figure 4 Simplified architecture overview of Modelon's web-based modeling and simulation platform.

4.1.1 Optimization framework

Optimization capabilities are important for many design and operation challenges, as highlighted in the previous analysis. Our solution tightly integrates the optimization formulation with the model formulation using the Modelica extension Optimica. Optimica enables the user to define the objective function, constraints and the optimization time horizon in an *optimization* class, which incorporates the Modelica models via extension or instantiation. The dynamic optimization capabilities of the Optimica Compiler Toolkit are used to solve the dynamic optimization problems defined in this way. The problems are symbolically transformed and transcribed into nonlinear programs (NLP) through direct collocation (Magnusson & Åkesson 2015). These are solved using IPOPT (Wächter and Biegler 2006).



Dynamic Steady State [✓ Custom]

Microgrid Optimization

days: 7

design: False [✓ True]

peak_shaving_rate: 25

Figure 5 Custom parameter interface in Modelon's web-based modeling and simulation platform. The user interface is automatically created based on input definitions in a Python file.

4.2 Model

Microgrid is a Modelica library developed by Modelon. It contains the basic components needed for building models aimed at evaluation and optimization of micro-grid configurations. The different components will be described in the following section.

4.2.1 Microgrid components

- **Weather:** Periodic solar insolation model.
- **Photovoltaics:** Calculates the generated power from solar insolation, based on rated capacity and efficiency.
- **Battery:** The battery model is defined by parameters for capacity, minimal and maximal state of charge and a maximal charge and discharge rate. A DC-connector is used for the charging and discharging.
- **Grid:** Ideal grid model, providing the electricity needed to balance the micro-grid through an AC-connector, at user-defined voltage.
- **Diesel generator:** Provides power to the micro-grid through an AC-connector, based on an input signal. The corresponding fuel consumption is calculated from a fuel curve defined by an intercept coefficient and a slope.
- **Load:** AC electric power consumption defined by input signal.
- **Transformer and inverters:** Efficiency based models changing voltage or current type.
- **Micro-grid manager:** Component containing replaceable control models determining the control strategy for the battery and the diesel generator. By connecting it to external sources, it supports using the micro-grid in optimization.

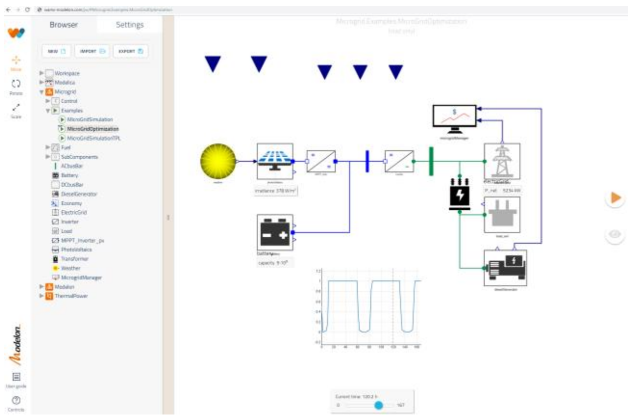


Figure 6 Microgrid Modelica library in Modelon's web-based modeling and simulation platform.

5 Use cases

Two different use cases are considered; peak shaving and economic dispatch. The same Modelica model of the micro-grid is used in both cases, a setup with all components is listed in Section 4.2.1.

The Modelica model is extended to Optimica models, which define optimization problems corresponding to the two use cases. They are determined by choosing different degrees of freedom and objectives in the optimization formulations. The optimization horizon of 7 days is divided into hourly samples in the economic dispatch and 30-minute samples in the peak shaving case.

The same electric load profile is used in both cases. It contains one week of electricity consumption data with a sampling time of 15 minutes. Simplified trajectories are used to represent the electricity price and insolation, but real data could be used for these too, by importing from a file in the same way.

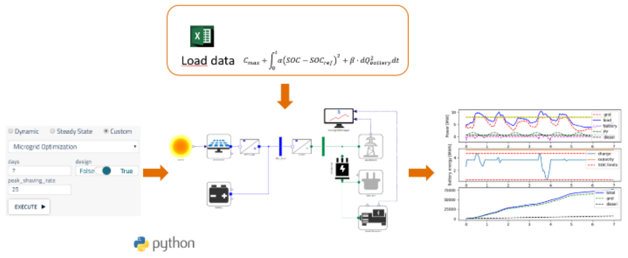


Figure 7 Illustration of optimization workflow. Running a custom function starts the execution of a Python script that compiles and optimizes a Modelica model. The result is automatically plotted and returned to the tool.

5.1 Peak shaving

Peak shaving is a method to reduce economic cost by limiting power consumption during high loads. In the modelled micro-grid, the battery is used to decrease the electricity demand from the grid. For this reason, a user-defined maximum constraint on the grid power is imposed on the system, reducing the maximum power peaks. The optimal operation of the micro-grid system is determined under this constraint, assuming constant diesel generator load and penalizing deviations from nominal state of charge for the battery. The battery size is in particular a degree of freedom in the optimization formulation and the derived size is the main result of the use case.

5.1.1 Optimization formulation

The optimization formulation is defined as the minimization of the following cost function:

$$C_{max} + \int_0^t \alpha\left(SOC - SOC_{ref}\right)^2 + \beta \cdot dQ_{battery}^2 dt$$

Where C_{max} is the battery capacity, SOC is the state of charge, SOC_{ref} the nominal state of charge and $dQ_{battery}$ is the battery charge control signal. α and β are constants determining the relative cost of the terms. The control signal is penalized to avoid unreasonably fast control action. The degrees of freedom are the battery size and the battery charge/discharge profile.

5.1.2 Results

Optimization results are displayed in Figure 8. The peak shaving of the grid load is displayed in the upper plot, for each period of high production the grid load is limited. The battery size can be seen in the middle plot, it has been chosen as small as possible without violating the prescribed state of charge limits.

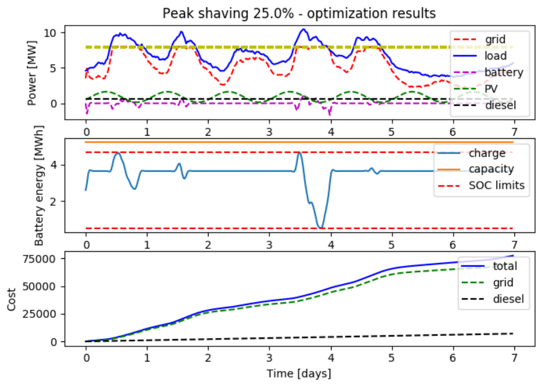


Figure 8 Results of peak shaving optimization.

5.2 Economic dispatch

The economic dispatch use case solves the problem of finding the optimal operation of the units in the micro-grid system, taking variations in load, electricity cost and solar irradiance into account. The main result is the operation mode of the battery and diesel generator and the resulting economic cost.

5.2.1 Optimization formulation

The following integral is minimized, taking economic costs and control signal changes into account:

$$\int_0^t p_{fuel} \cdot F_{fuel} + p_{el} \cdot P_{el} + \alpha \cdot dQ_{generator}^2 + \beta \cdot dQ_{battery}^2 dt$$

Where p_{fuel} and p_{el} are the diesel fuel and grid electricity prices, respectively, F_{fuel} is the diesel consumption, P_{el} is the grid electricity usage, and α and β determine the penalties for the generator load and battery charge control signals.

5.2.2 Results

The results from the economic dispatch experiment can be seen in Figure 9. The battery usage and the diesel generator load follow the variations in the electricity price, so that grid electricity consumption is minimized during the price peaks.

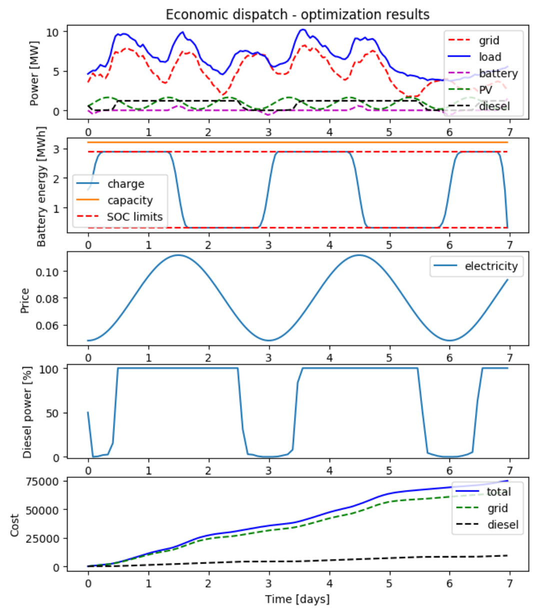


Figure 9 Result of economic dispatch.

6 Conclusions

In this paper we have presented a unified platform for microgrid design and operation with the scope to efficiently solve todays and future engineering problems for distributed energy sources and storage systems. We also investigated the requirements from a model and tool perspective with a focus on both technology and user interface. The framework is based on the open standards Modelica, Python, HTML and JavaScript and is built around Modelon's new web-based modeling and simulation platform. The use case demonstrated successfully that the framework can be used to solve peak shaving and economic dispatch optimization problems.

A major benefit with a Modelica based platform is the openness and flexibility, where it's possible to define an architecture with different fidelity levels that can be used for various types of execution and analysis types.

Future work will focus on adding additional components such as wind power and defining a flexibility architecture for 3-phase AC systems. Other focus areas are improving the fidelity level of already existing components and adding aging effects of batteries in the optimization problem.

References

Wenbo Du, Humberto E. Garcia and Christiaan J. J. Paredis. An Optimization Framework for Dynamic Hybrid Energy Systems. *10th International Modelica 2014 Conference*. Lund, Sweden.

Jonas Enerbäck and Oscar Nalin Nilsson. Modelling and Simulation of Smart grids using Dymola/Modelica. Division of Industrial Electrical Engineering and Automation Faculty of Engineering, Lund University, 2013

Joseph H. Eto, Robert Lasseter, David Klapp, Amrit Khalsa, Ben Schenkman, Mahesh Illindala and Surya Baktiono. The CERTS Microgrid Concept, as Demonstrated at the CERTS/AEP Microgrid Test Bed, Energy Analysis and Environmental Impacts Division Lawrence Berkeley National Laboratory, 2018

A. Hina Fathima and K. Palanisamy. Optimization in microgrids with hybrid energy systems – A review. *Renewable and Sustainable Energy Reviews*, 45, 431-446, 2015

Rüdiger Franke and Hansjürg Wiesmann. Flexible modeling of electrical power systems – the Modelica PowerSystems library. *10th International Modelica 2014 Conference*. Lund, Sweden.

GAMS (2018). URL https://www.gams.com/

Paulo Giangrande, Christopher Hill, Chris Gerada and Serhiy Bozhko. Multi-Level Library of Electrical Machines for Aerospace Applications. *10th International Modelica 2014 Conference*. Lund, Sweden.

Homer Energy (2018). URL https://www.homerenergy.com/

Pierre Emanuel Kehl, Raja Rehan Khalid and Georg Frey. Heterogeneous Modeling: A Need to Model Future Energy Systems. *Power and Energy Student Summit 2017*

Ricardo Luna-Rubio, Mario Trejo-Perea, Damián Vargas Vázq and José Ríos-Morena. Optimal sizing of renewable hybrid energy systems. A review of methodologies. *Solar Energy* 86, Issue 4:1077-1088, 2012

Fredrik Magnusson and Johan Åkesson. Dynamic optimization in JModelica.org. *Processes*, 3(2):471–496, 2015

Manitoba Hydro International (2018). URL https://hvdc.ca/pscad/

Modelica Association Libraries. Available at https://www.modelica.org/libraries, accessed 2018-11-19.

Jorge Nocedel and Stephen J. Wright (2006). Numerical Optimization. New York, NY: Springer New York.

Juan Van Roy, Robbe Salenbien and Johan Driesen. Modelica Library for Building and Low-Voltage Electrical AC and DC Grid Modeling. *10th International Modelica 2014 Conference*. Lund, Sweden.

Peter W. Sauer, M. A. Pai. and Joe H. Chow. (1997). Power System Dynamics and Stability, John Wiley & Sons Inc.

Ramakrishnan Venkatraman and Siddharta Kumar Khaitan. A Survey of Techniques for Designing and Managing Microgrids. *IEEE Power & Energy Society General Meeting*, 2015

Wikipedia contributors. (2018, August 28). Economic dispatch. In Wikipedia, The Free Encyclopedia. Retrieved 13:51, November 12, 2018, from https://en.wikipedia.org/w/index.php?title=Economic_dispatch&oldid=856915225

Andreas Wächter and Lorenz T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

Johan Åkesson. (2008). Optimica - An Extension of Modelica Supporting Dynamic Optimization. *Proceedings of the 8th International Modelica 2008 Conference*. Bielefeld, Germany.

Influence of Excess Power Utilization in Power-to-Heat Units on an Integrated Energy System with 100 % Renewables

Carsten Bode   Gerhard Schmitz

Institute of Engineering Thermodynamics, Hamburg University of Technology, Denickestr. 17, 21073 Hamburg, Germany, {c.bode,schmitz}@tuhh.de

Abstract

This paper presents the effect which the utilization of excess power in Power-to-Heat units has on an energy system which is fully supplied by renewables. For this, a possible future German integrated energy system consisting of the power, heat and gas sectors is modeled using the TransiEnt Library in Modelica®. The first system contains electric energy storage units, Power-to-Gas as well as Gas-to-Power plants and hot water storage units as energy storage technologies. The heat supply does not use excess power, and an option to curtail renewable power generation is added. The system costs are optimized by using simplified models in MATLAB® and designed in Modelica afterwards to include the dynamic effects. In a second system, excess power can also be used in existing electric heat pumps and in a third system as well in existing electric heating rods installed in the hot water storage tanks instead of curtailing renewable energy generation. This reduces the component sizes and thus the cost of the system because only control has to be added to enable this behavior.

Keywords: Integrated Energy System, 100% Renewables, Power-to-Heat, Energy System Analysis

1 Introduction

To reduce the effects of climate change, the United Nations have created the Paris Agreement (United Nations, 2015). This is a big challenge because CO_2 emissions have to be decreased to a minimum which can only be achieved by massive integration of renewable energies in all energy sectors. Because the most promising renewable energies produce electricity directly, e.g. photovoltaics or wind turbines (International Energy Agency, 2016), and to use good storage capacities in the other sectors, sector coupling will become inevitable.

Many studies have been conducted in recent years, examining how future integrated energy systems may be designed, e.g. Benndorf et al. (2014); Gerhardt et al. (2015); Henning and Palzer (2015); Nitsch et al. (2012); Pape et al. (2014); Teske et al. (2015). In most cases, those studies are conducted with simplified models. Most commonly, quasi-stationary models are optimized over a whole year in time steps of one hour using Mixed Integer Linear Programming (MILP) to find the most cost-efficient configuration of an energy system. Those models neglect dynamic effects, e.g. time constants of storage units or rapid changes which occur faster than the time resolution of the models.

Therefore, the search for a cost-efficient configuration of a future integrated energy system consisting of the power, heat and gas sectors was conducted using simplified quasi-stationary models first and then detailed, dynamic models for the exact system design in Bode and Schmitz (2018). This work has been extended by adding more detail to enable the examination of the influence which the utilization of excess power in Power-to-Heat (PtH) units has on the overall system.

One option to implement dynamic models is the programming language Modelica (Modelica Association, 2018) which enables equation-based and object-oriented model development. Due to this approach, physical equations can be written directly in the code and good reusability as well as simple maintenance are guaranteed.

2 Previous Work

In Bode and Schmitz (2018), renewable energy production and end use energy demand curves are generated for the future energy system of Germany including the power, heat and gas sectors for a year when those sectors will be fully supplied by renewables. End use of heat includes low temperature heat whereas the end use of gas contains high temperature heat and non-energetic use of gas. Generation and consumption are matched using different electric energy storage technologies (lithium-ion battery, pumped hydro storage or adiabatic compressed air energy storage), Power-to-Gas units (electrolyzer and methanation unit), gas storage volumes, Gas-to-Power units (gas turbine or combined cycle gas turbine), heat producers (solar thermal collector, electric heat pump, gas heat pump, gas boiler) and hot water storage units; see Table 1 and Figure 1. Energy transmission is assumed to be ideal.

The different storage and conversion technologies are charged and discharged in a given order, e.g. PHS, LIB, PtG, CCGT. So in the case of negative residual load, if the PHS unit is fully charged or is already operating at maximum power, the LIB will be charged and so on.

This integrated energy system is implemented in two different ways: First, as a dynamic model, using detailed, dynamic models from the open-source TransiEnt

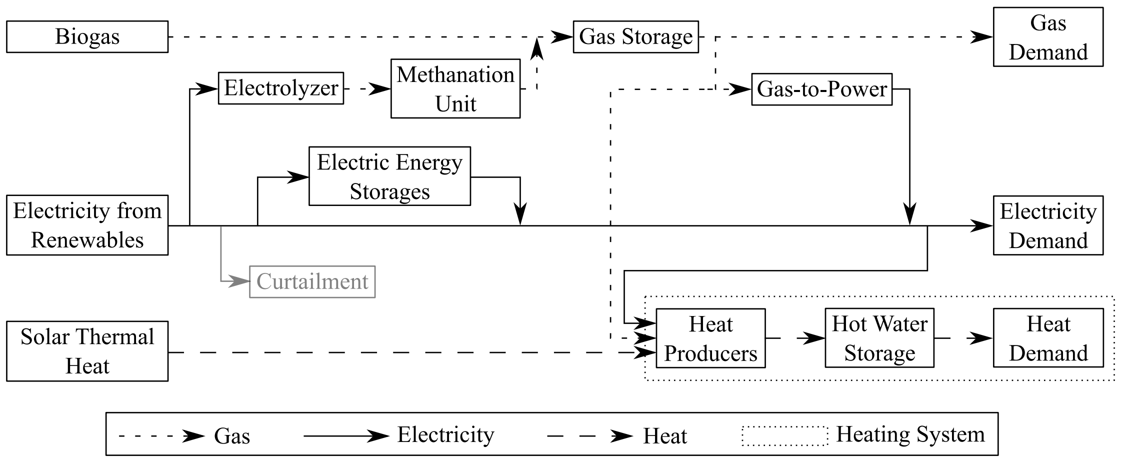


Figure 1. Overall system with energy flows between all components examined by Bode and Schmitz (2018). Curtailment (in grey) is added in this work.

Table 1. Abbreviations of components in the system

Component	Abbreviation
Lithium-ion battery	LIB
Pumped hydro storage	PHS
Adiabatic compressed air energy storage	A-CAES
Power-to-Gas unit (electrolyzer and methanation unit)	PtG
Gas-to-Power unit	GtP
Gas turbine	GT
Combined cycle gas turbine	CCGT

Library (Hamburg University of Technology, 2018; Andresen et al., 2015) and models under development, which are implemented in Modelica in the simulation environment Dymola (Dassault Systèmes, 2018). Second, as a simplified model using quasi-stationary equations in MATLAB (MathWorks, 2018a) to guarantee low computing times to be able to compare a lot of system configurations in regard to cost. However, due to the high level of detail, the heating system is included in this system using gas or electricity demand curves taken from detailed Modelica simulations.

To find the best system configuration, first, the heating system is simulated to create the electricity and gas demand curves for the heat supply. Second, the simplified models are used to simulate a wide variety of combinations of the storage and conversion technologies with different sizes. It is ensured that all storage units have the same state of charge (SOC) at the beginning and at the end of the year and that all demands are met. Third, the most promising configurations are again designed in Modelica and at last compared regarding the cost.

3 Approach

First, a cost-efficient system configuration has to be found. This is achieved by applying the optimization algorithm `patternsearch` from the MATLAB Optimization Toolbox (MathWorks, 2018b) on the simplified MATLAB model of Bode and Schmitz (2018) with 15 min time steps to which the ability to curtail renewable energy generation is added. In the former configuration of the system, all produced renewable electric energy has to be consumed by the consumers or storage units as not to waste energy. With this strategy, the PtG capacity is designed for the peak load which occurs only once per year. Adding curtailment leads to higher operation hours of the plants and thus to lower specific cost.

The heat, which is not generated by solar thermal collectors, is supplied by an electric heat pump in combination with an electric boiler and an electric heating rod installed in the hot water storage. Gas heat pumps and gas boilers were excluded because they were not cost-efficient and the potential of excess power utilization is the highest in the electrically driven heat producers. The heating system is implemented in the MATLAB model by using electricity demand curves from the heating system modeled in Modelica.

Different combinations of storage technologies and charging orders are tested based on the results of Bode and Schmitz (2018) and optimization runs are started from several starting points to find different local minima. In a multi-dimensional numerical optimization one can never be sure if the found minimum is the global minimum or just a local one. So, the best minimum found is used and assumed to be the global minimum. As Henning and Palzer (2015) state, there are a lot of different solutions with only slightly differing cost. Also, the results are highly dependent on a lot of assumptions, especially prices in the far future, so the used result here is just a possible, but likely solution.

Second, this solution is designed by simulating the detailed Modelica system model (System 1, S1) over a whole year. The renewable, PtG and GtP nominal power are changed iteratively to ensure the same SOC of all storage units at the end as at the beginning of the year while keeping the electric energy storage sizes and maximum curtailment power constant.

Third, excess power utilization in PtH units is implemented in the detailed models in Modelica. So, energy, which would otherwise be curtailed, i.e. wasted, is used directly in electric heat pumps (S2) or additionally in electric heating rods in the hot water storage tanks (S3). This way, heat generation is shifted to times with a lot of renewable generation. These systems are designed as well using the Modelica system models under the same conditions as S1 but keeping the maximum of the sum of curtailed power and used excess power in the heating system constant.

All simulations are conducted with the Dymola solver Radau IIa and a tolerance of 10^{-6}.

4 Models

The existing models of Bode and Schmitz (2018), i.e. quasi-stationary MATLAB and dynamic Modelica models, are extended by adding curtailment. In the Modelica models, the heating system is also modeled in more detail to enable a better quantification of the use of excess power in the PtH units.

4.1 Curtailment

Instead of actually reducing the power production of renewables in case of an overproduction, this excess power is consumed by a curtailment model. This way, it can be included in the existing storage technology structure in the models. The curtailment model has a maximum power, an infinite power gradient and an infinite energy capacity.

4.2 Heating System

The considered heating system, which is modeled in Modelica, is shown in Figure 2. With this model, a power consumption curve is created which is used by the simplified MATLAB model as well as S1 as a demand curve. In S2 and S3, the heating system model is included in the overall system model to enable direct coupling; see Figure 1.

4.2.1 Solar Collector

In the formerly used solar collector model, the water, which absorbs the solar radiation, is modeled using the TILMedia Library (TLK-Thermo GmbH and Institut für Thermodynamik, Technische Universität Braunschweig, 2018). In reality, usually a water-glycol mixture is used to avoid freezing but due to the unavailability of open-source media data for this medium in the TILMedia Library, pure water is used. At low ambient temperatures, low water temperatures are reached, which leads to freezing, and thus to warnings from the used media data. The freezing does not influence the model results, because at those times, the solar radiation is so low that no heat is produced, but the simulation is slowed down due to the output of those warnings. Therefore, a simple water model is implemented, using constant density $\rho = 989.9\,\mathrm{kg/m^2}$ and specific heat capacity $c_\mathrm{W} = 4184\,\mathrm{J/(kg\,K)}$ based on the fluid model TILMedia_SplineWater between 10 °C and 75 °C at 1 bar. The specific enthalpy h is calculated using the caloric state equation with the temperature T and reference point 0 at $T_0 = 273.15\,\mathrm{K}$ with $h_0 = 59.65\,\mathrm{J/kg}$.

$$h = c_\mathrm{W}(T - T_0) + h_0 \tag{1}$$

Table 2. Positions of the inlets, outlets and the electric heating rod in the hot water storage, measured from the bottom (Recknagel et al., 2017)

Component		Height in m
Electric heat pump	inlet	1.7
	oulet	1.2
Solar collector	inlet	0.8
	oulet	0.3
Electric heating rod		1.0
Space heating	inlet	1.0
	oulet	1.8
Hot water and process heat	inlet	0.0
	oulet	2.0

4.2.2 Hot Water Storage

To model the hot water storage in a more realistic way, the existing model is extended, making it possible to add more ports in different heights. This way, for instance, the solar collector can feed the hot water into the lower part of the storage while the electric heat pump charges the upper part; see Figure 2. The heights used from Recknagel et al. (2017) are given in Table 2.

An electric heating rod is also added to the storage which works like the existing electric boiler model. Because of this, the electric heat pump does not have to be designed for the maximum heating capacity which only occurs once in a few years. Due to much lower specific cost of the electric heating rod, which is assumed to be the same as for the electric boiler (70 €/kW$_\mathrm{th}$ (Elsner et al., 2015)), the cost of the whole system can be reduced.

4.2.3 Consumer Side

To simplify the control structure and speed up the simulations, the consumer models are simplified.

First, the consumers model is combined with the pump model. The necessary mass flow rate is calculated using the specific enthalpy calculated by the TILMedia fluid model at the outlet using the desired temperature, the specific enthalpy at the inlet and the consumed heat flow rate. With this model, the controller for the pump can be left out. Additionally, the back-mixing valve with its controller can be omitted as long as it is ensured that the temperature in the

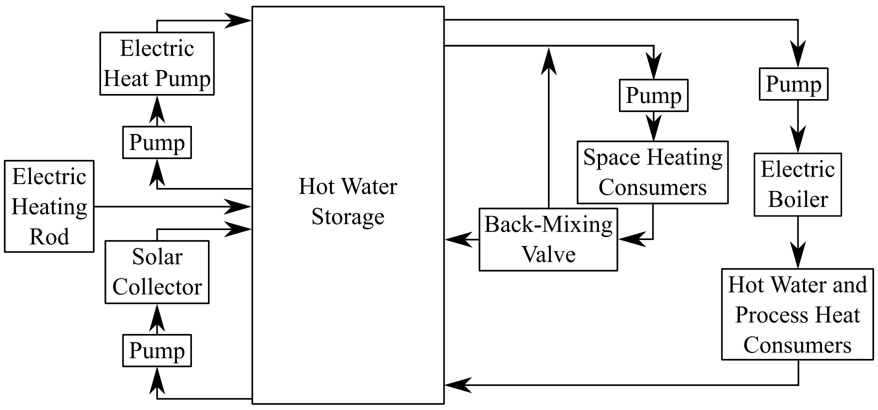


Figure 2. Considered heating system with modeled water flows.

corresponding control volume of the hot water storage is always above the current supply temperature defined by the heating curve. This is guaranteed by the control of the heat producer.

Second, on the process heat and hot water consumer's side, the electric boiler can be added to the combined pump-consumer model using the same principle as described above with a desired electric boiler outlet temperature of 60 °C. The temperature after the consumer is assumed to be 10 °C.

All these simplifications have been validated against the more detailed models to ensure correct behavior.

4.2.4 Electric Heat Pump

The electric heat pump model is also combined with the pump model as it is done for the consumers but more equations have to be added to ensure useful behavior. If the specific enthalpy at the inlet overshoots the desired specific enthalpy at the outlet, the mass flow is set to zero and the specific enthalpy is just passed through to the outlet. This way, the electric heat pump does not overheat. In addition, the mass flow rate can be limited to avoid high peaks.

For this model, a validation against the detailed models has been conducted as well.

4.2.5 Heating System Control

Proportional (P) controllers are favored over PI or PID controllers in general for control due to higher simulation speed and negligible control errors. The model LimPID from the ClaRa library (Hamburg University of Technology et al., 2018; Brunnemann et al., 2012) is modified and used, leaving out the smooth activation feature because it was found to be unsuitable in some cases.

The control of the pump for the solar collector is changed to make it more realistic to work in matched-flow operation. Volume flows between 8 and 40 $\mathrm{l/(m^2\,h)}$ with temperatures between 75 °C and 90 °C are used (Späte and Ladener, 2011). To implement this behavior, a P controller is used. To avoid chattering, a hysteresis is implemented, which turns the pump on if the inlet temperature is 5 K above the temperature in the solar collector and turns it off again if the temperature difference is below 2 K. Minimum on and off times of 1 h each are also added to limit the switching events.

The control of the electric heat pump is modified to enable the use of excess power. The existing P controller, which is responsible for the normal operation of the heat pump, has a set value of 0.5 K above the maximum supply temperature for space heating of 45 °C (the minimum supply temperature is 35 °C) and measures the top temperature of the storage. For the excess power utilization, a second P controller is added with the same set value as the first controller but the measured value is the temperature of the fifth out of ten control volumes (counted from the top). The output is limited to the available excess power and added to the output of the first controller and again limited by the nominal power of the heat pump.

To control the electric heating rod in the hot water storage in normal operation, i.e. without the use of excess power, a controller is turned on when the ambient temperature is below the bivalence point or if the temperature at the top of the storage is 0.5 K below the desired value. Its set and measured values are the same as for the heat pump controller.

If excess power should be used in the electric heating rod in the hot water storage as well, a second controller is added here, similar to the heat pump controller. The set temperature is 75 °C to increase the storage capacity. The output is limited by the remaining excess power, added to the output of the first controller and limited to the maximum power of the heating rod.

The required electric base load for the heat supply is calculated by the sum of the outputs of the first controller of the electric heat pump and the electric heating rod, respectively. The additional power used is the used excess power.

To avoid a direct influence of the base load on the excess power which is fed to all the storage technologies, a first order block is set after the heating system. The use of excess power directly decreases the base load which would instantly increase the excess power. This would

Table 3. Mathematical and numerical properties of the different systems.

	S1	S2	S3
No. of equations	2645	4294	4395
No. of nontrivial equations	1946	3253	3348
No. of differentiated variables	73	86	86
No. of time states	49	62	62
Biggest nonlinear system of equations before manipulation	10	23	27
Biggest nonlinear system of equations after manipulation	1	3	3
No. of event iterations	6015	20550	23809
CPU time in h	0.47	1.72	12.44

influence all storage technologies and slow down the simulation. A small time constant of 60 s is sufficient and does not distort the results.

4.2.6 Design of Components

The electric heat pump and electric heating rod in the hot water storage tank were sized for a bivalent operation (Recknagel et al., 2017). This means that the electric heat pump should be able to supply all the heat at the bivalence point which was chosen to be $-5\,°\mathrm{C}$ according to Recknagel et al. (2017). The standard ambient temperature was calculated using a weighted average of the standard ambient temperatures from DIN EN 12831 (DIN Deutsches Institut für Normung e.V., 2008) of the six biggest German metropolitan regions according to IKM (2018) which results in $-12.40\,°\mathrm{C}$. Using the standard load profile approach from BDEW et al. (2016), daily mean values of the heat demand at standard ambient temperature and at the bivalence temperature are calculated. With an assumed efficiency of the hot water storage of 99 %, the electric heat pump and the electric heating rod should have nominal heat flow rates of $137.6\,\mathrm{GW_{th}}$ and $31.04\,\mathrm{GW_{th}}$ respectively. The maximum mass flow of the pump for the electric heat pump is assumed to be $3 \cdot 10^6\,\mathrm{kg/s}$.

The electric boiler, which increases the temperature of the water for the hot water and process heat demand, is designed so that the highest occurring heat flow rate in that year can be supplied.

The area of the solar collector is chosen to produce the desired heat according to Bode and Schmitz (2018) and the volume of the hot water storage is calculated using a specific value of $0.075\,\mathrm{m^3/m^2}$ (Recknagel et al., 2017).

4.3 Mathematical and Numerical Properties

The different systems vary strongly in their mathematical and numerical properties as is shown in Table 3. The CPU times were measured on a 64 bit cluster with Intel® Xeon® E5-2650 v3 CPUs with 2.30 GHz. Even though the number of equations and number of event iterations only increase slightly from S2 to S3, the CPU time rises by factor 7.2.

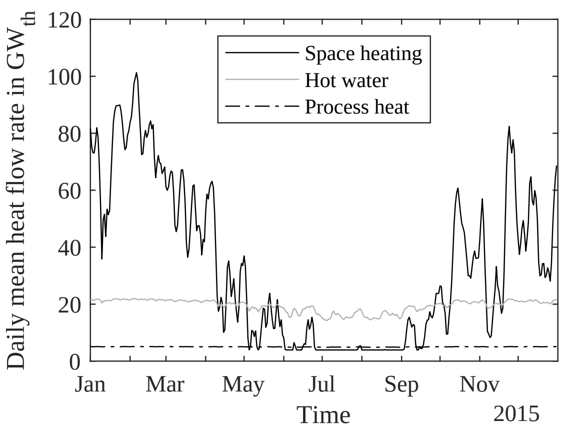


Figure 3. Heat demand curves of the different applications (daily mean values).

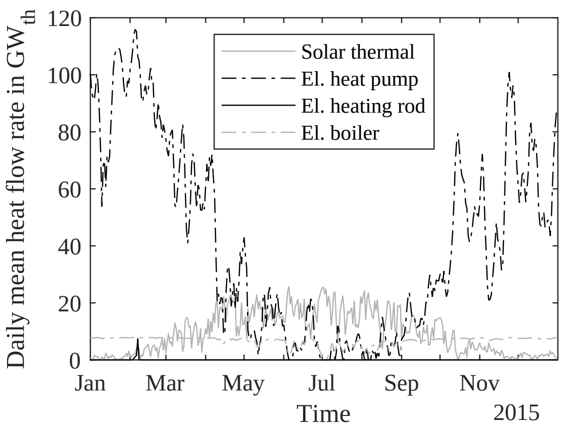


Figure 4. Heat supply curves of the different heat producers (daily mean values).

5 Results

5.1 Design Results without Excess Power Utilization (S1)

For the heating system, the design process returned a nominal power of the electric boiler of $10.62\,\mathrm{GW_{th}}$, a solar thermal collector area of $200.6 \cdot 10^6\,\mathrm{m^2}$ and a hot water storage volume of $15.04 \cdot 10^6\,\mathrm{m^3}$ which leads to a specific solar heat production of $423.8\,\mathrm{kWh/m^2}$. In Figures 3 and 4, the heat demand and heat supply curves are shown. The electric heating rod is used only from February 3rd to 7th because then low ambient temperatures are reached.

The best configuration that was found by the optimizer in MATLAB is where a pumped hydro storage ($187.6\,\mathrm{TJ}$, $8.60\,\mathrm{GW_{el}}$) and a lithium-ion battery ($284.3\,\mathrm{TJ}$, $54.58\,\mathrm{GW_{el}}$) are charged in this order and Power-to-Gas in combination with combined cycle gas turbines is used. The remaining power is curtailed. The results are listed in Table 4.

Table 4. Results of the different systems.

	S1	S2	S3
Renewable electric power in GW_{el}	513.7	510.5	505.4
Electrolyzer power in GW_{el}	136.5	135.3	135.2
Gas-to-Power power in GW_{el}	120.4	119.8	119.8
Gas storage in 10^6 kg	4315	4285	4315
Maximum curtailed power in GW_{el}	152.3	152.3	121.0
Curtailed energy in TWh_{el}	30.51	28.76	12.61
Electricity consumption of the heating system in TWh_{el}			
- Electric heat pump	119.87	120.18	117.05
- Electric heating rod	0.25	0.27	17.99
- Electric boiler	59.24	58.69	55.13
Seasonal performance factor of the heating system	2.804	2.807	2.644
Solar thermal generation in TWh_{th}	85.00	84.79	82.70
Annuity in bil. €	111.03	110.67	110.37

5.2 Proof of Concept of Excess Power Utilization

In Figure 5, curves from a winter day are shown to visualize the function of the system with excess power utilization in the electric heat pump and the electric heating rod (S3). Over the day, only low solar thermal generation occurs and the electric heat pump supplies most of the heat demand. At 05:45 AM, excess power is available and completely utilized by the electric heat pump whereas from 11:30 AM the residual load cannot be consumed completely because the storage top temperature has reached the supply temperature of the electric heat pump and the electric heating rod is at maximum capacity which results in a noticeable temperature increase in the middle of the storage.

In Figure 6, a summer day with a significant solar heat production is shown. Because of the high temperatures in the storage due to previous sunny days, the excess power is only partly used and the rest is curtailed: In the beginning (8:00 AM), it is used in the electric heat pump and from 8:25 AM on in the electric heating rod because the temperature is too high for the heat pump. The short minimum in the available excess power at 11:30 AM results from a charging process in the lithium-ion battery.

5.3 Results with Excess Power Utilization (S2 and S3)

When excess power is used in the electric heat pump (S2), the system becomes more efficient; see Table 4. The electric heat pump consumes just slightly more electric energy (0.31 TWh_{el}, i.e. 0.26 %) but this small shift in the demand leads to a noticeable decrease in required renewable electric power (-3.23 GW_{el}) as well as the Power-to-Gas, Gas-to-Power and gas storage units and thus a cost decrease of 0.33 %. Because the electric heat pump slightly increases the average temperature at the top of the storage, the electric boiler produces less heat which results in a small increase in the seasonal performance factor of the whole heating system from 2.804 to 2.807 but the solar thermal generation also drops by 0.25 %.

The decrease in the gas-related component sizes and cost continues when excess power is also used in the electric heating rod in the hot water storage (S3). In this case, the electric heat pump consumes less electric energy because the heat from the electric heating rod partly replaces heat which would otherwise have been produced by the electric heat pump. This results in a strong decrease in the seasonal performance factor to 2.644 as well as the solar thermal generation by 2.70 % compared to S1.

The maximum curtailed power and energy both sink significantly by 20.58 % and 58.65 %, respectively, compared to the reference case (S1), which leads to the observed increase in system efficiency and decrease in cost by 0.60 %. Of course, this value depends highly on the cost assumptions for the components but there can be no case in which the cost would rise because, due to the excess power utilization, only certain component sizes decrease but none increase significantly.

6 Conclusion

The existing models from Bode and Schmitz (2018) are extended, adding curtailment and more detail in the heating system. The combination of quasi-stationary MATLAB models and dynamic Modelica models allows for a good balance of speed and accuracy to investigate the influence of excess power utilization in electric heat pumps and electric heating rods. The reusability of the Modelica models enables the user to quickly build complex energy system models for dynamic simulation.

Starting at a cost-optimal point (S1), excess power usage in the electric heat pump (S2) and additionally in the electric heating rod (S3) is implemented. The results show that the cost can be reduced from S1 to S2 by 0.33 % and from S1 to S3 by 0.60 %. Those are small values but only the control structure of the heating system has to be changed to enable the system to move the heat generation to times of high renewable energy generation. This way, the system becomes more cost and energy efficient.

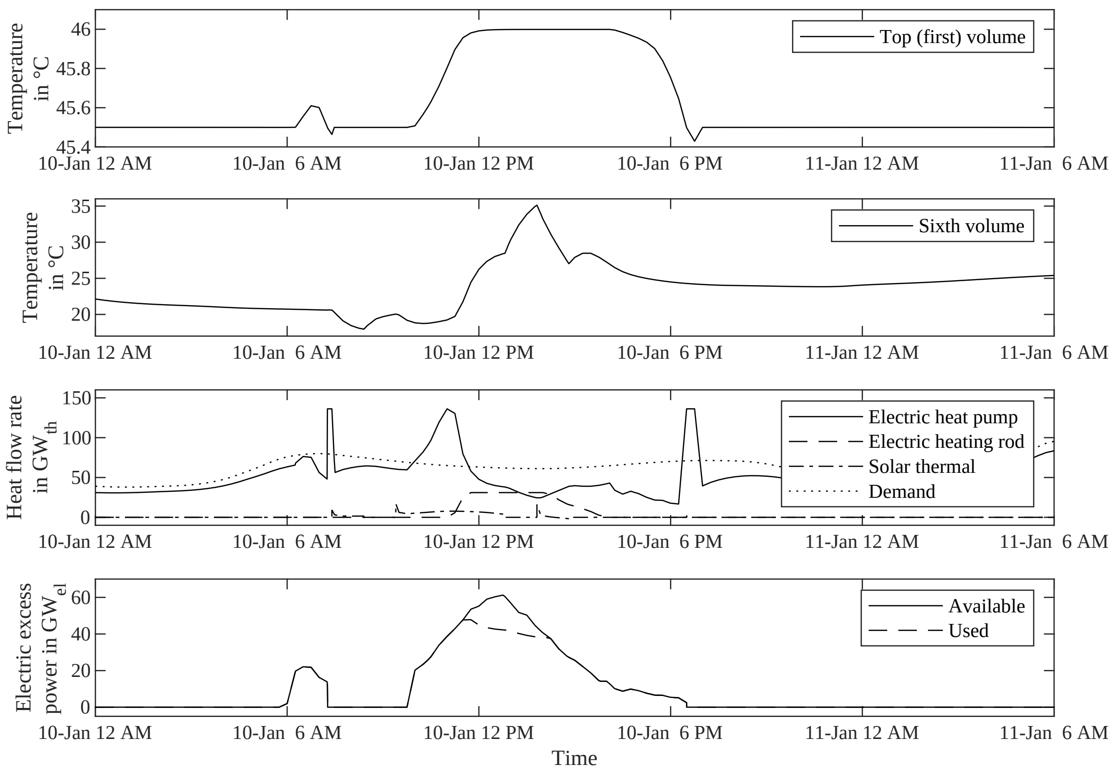


Figure 5. Curves of storage temperatures, heat flow rates and electric excess power in the heating system on a winter day in S3.

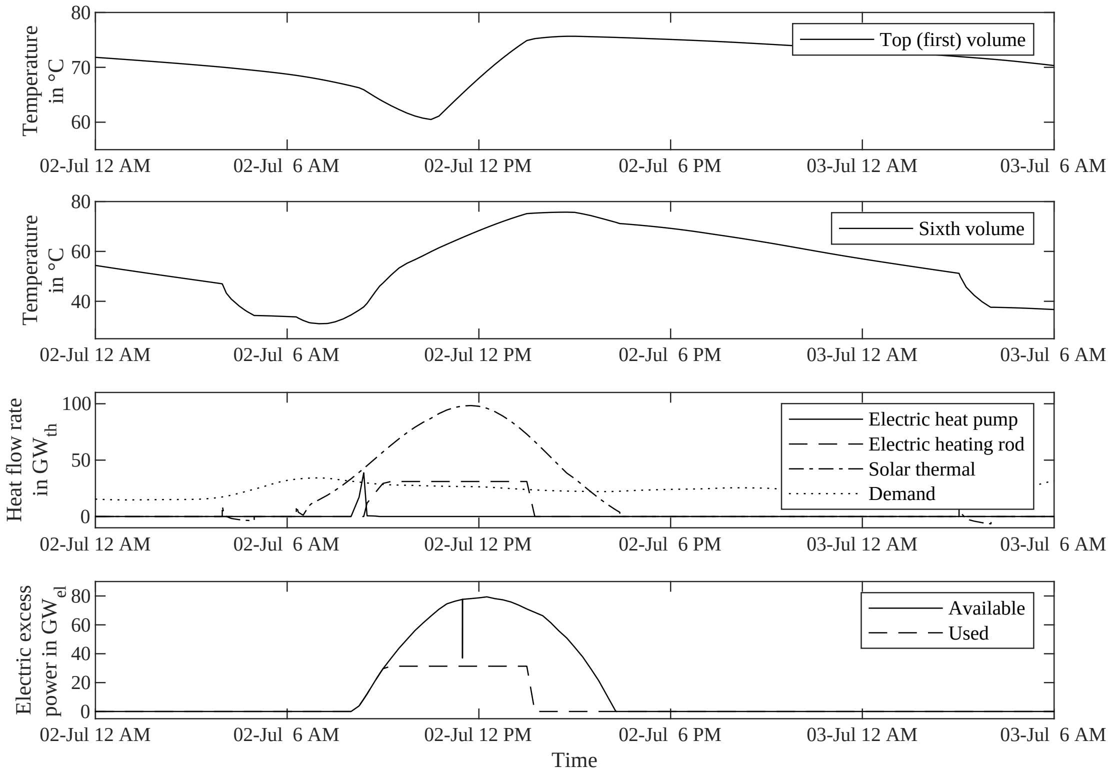


Figure 6. Curves of storage temperatures, heat flow rates and electric excess power in the heating system on a summer day in S3.

Acknowledgements

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

Lisa Andresen, Pascal Dubucq, Ricardo Peniche, Günter Ackermann, Alfons Kather, and Gerhard Schmitz. Status of the TransiEnt Library: Transient simulation of coupled energy networks with high share of renewable energy. In *Proceedings of the 11th International Modelica Conference*, pages 695–705, Versailles, 2015. doi:10.3384/ecp15118695.

BDEW, VKU, and GEODE. BDEW / VKU / GEODE- Leitfaden. Abwicklung von Standardlastprofilen Gas. Technical report, Berlin, 2016.

Rosemarie Benndorf, Maja Bernicke, Andreas Bertram, Wolfgang Butz, Folke Dettling, Johannes Drotleff, Corinna Elsner, Eric Fee, Christopher Gabler, Christine Galander, Yvonne Hargita, Reinhard Herbener, Tim Hermann, Fabian Jäger, Judith Kanthak, Hermann Kessler, Yvonne Koch, David Kuntze, Martin Lambrecht, Christian Lehmann, Harry Lehmann, Sandra Leuthold, Benjamin Lünenbürger, Insa Lütkehus, Kerstin Martens, Felix Müller, Klaus Müschen, Diana Nissler, Sebastian Plickert, Katja Purr, Almut Reichart, Jens Reichel, Hanno Salecker, Sven Schneider, Jens Schuberth, Dietrich Schulz, Marlene Sieck, Ulla Strenge, Bärbel Westermann, Kathrin Werner, Christine Winde, Dietmar Wunderlich, and Brigitte Zietlow. Treibhausgasneutrales Deutschland im Jahr 2050 (Climate Change 07/2014). Technical report, Dessau-Roßlau, 2014.

Carsten Bode and Gerhard Schmitz. Dynamic Simulation and Comparison of Different Configurations for a Coupled Energy System with 100 % Renewables. *Energy Procedia*, 155:412–430, 2018.

Johannes Brunnemann, Friedrich Gottelt, Kai Wellner, Ala Renz, André Thüring, Volker Roeder, Christoph Hasenbein, Christian Schulze, Gerhard Schmitz, and Jörg Eiden. Status of ClaRaCCS : Modelling and Simulation of Coal-Fired Power Plants with CO2 Capture. In *Proceedings of the 9th International Modelica Conference*, pages 609–618, Munich, 2012. doi:10.3384/ecp12076609.

Dassault Systèmes. Dymola 2019 - Dynamic Modeling Laboratory, 2018. URL `https://www.3ds.com/products-services/catia/products/dymola/`.

DIN Deutsches Institut für Normung e.V. DIN EN 12831 Beiblatt 1:2008-07, Heizsysteme in Gebäuden - Verfahren zur Berechnung der Norm-Heizlast - Nationaler Anhang NA, 2008.

Peter Elsner, Manfred Fischedick, Dirk Uwe Sauer, Berit Erlach, and Benedikt Lunz. Flexibilitätskonzepte für die Stromversorgung 2050. Technologien - Szenarien - Systemzusammenhänge (Analyse aus der Schriftenreihe Energiesysteme der Zukunft). Technical report, München, 2015.

Norman Gerhardt, Fabian Sandau, Angela Scholz, Henning Hahn, Patrick Schumacher, Christina Sager, Fabian Bergk, Claudia Kämper, Wolfram Knörr, Jan Kräck, Udo Lambrecht, Oliver Antoni, Johannes Hilpert, Katharina Merkel, and Thorsten Müller. Interaktion EE-Strom, Wärme und Verkehr. Technical report, Kassel, Heidelberg, Würzburg, 2015.

Hamburg University of Technology. TransiEnt Library, 2018. URL `https://www.tuhh.de/transient-ee/en/`.

Hamburg University of Technology, TLK-Thermo GmbH, and XRG Simulation GmbH. ClaRa 1.3.0, 2018. URL `https://claralib.com/`.

Hans-Martin Henning and Andreas Palzer. Was kostet die Energiewende? Wege zur Transformation des deutschen Energiesystems bis 2050. Technical report, Freiburg, 2015.

IKM. Initiativkreis Europäische Metropolregionen in Deutschland, 2018. URL `http://www.deutsche-metropolregionen.org/`.

International Energy Agency. World Energy Outlook 2016. Technical report, Paris, 2016.

MathWorks. MATLAB 2018b, 2018a. URL `https://www.mathworks.com/products/matlab.html`.

MathWorks. MATLAB Optimization Toolbox 8.1, 2018b. URL `https://www.mathworks.com/products/optimization.html`.

Modelica Association. Modelica and the Modelica Association, 2018. URL `https://www.modelica.org/`.

Joachim Nitsch, Thomas Pregger, Tobias Naegler, Dominik Heide, Diego Luca de Tena, Franz Trieb, Yvonne Scholz, Norman Gerhardt, Michael Sterner, Tobias Trost, Amany von Oehsen, Rainer Schwinn, Carsten Pape, Henning Hahn, and Bernd Wenzel. Langfristszenarien und Strategien für den Ausbau der Erneuerbaren Energien in Deutschland bei Berücksichtigung der Entwicklung in Europa und global. Schlussbericht. Technical report, Stuttgart, Kassel, Teltow, 2012.

Carsten Pape, Norman Gerhardt, Philipp Härtel, Angela Scholz, Rainer Schwinn, Tim Drees, Andreas Maaz, Jens Sprey, Christopher Breuer, Albert Moser, Frank Sailer, Simon Reuter, and Thorsten Müller. Roadmap Speicher. Speicherbedarf für erneuerbare Energien - Speicheralternativen - Speicheranreiz - Überwindung rechtlicher Hemmnisse. Endbericht. Technical report, Kassel, Aachen, Würzburg, 2014.

Hermann Recknagel, Otto Ginsberg, Kurt Gehrenbeck, Eberhard Sprenger, Winfried Hönmann, Ernst-Rudolf Schramek, and Karl-Josef Albers (editors). *Taschenbuch für Heizung und Klimatechnik*. DIV Deutscher Industrieverlag GmbH, München, 78th edition, 2017.

Frank Späte and Heinz Ladener. *Solaranlagen. Handbuch der thermischen Solarenergienutzung*. ökobuch Verlag, Staufen bei Freiburg, 11th edition, 2011.

Sven Teske, Steve Sawyer, Oliver Schäfer, Thomas Pregger, Sonja Simon, Tobias Naegler, Stephan Schmid, Doruk Özdemir, Johannes Pagenkopf, Florian Kleiner, Jay Rutovitz, Elsa Dominish, Jenni Downes, Thomas Ackermann, Tom Brown, Simon Boxer, Ricardo Baitelo, and Larissa A. Rodrigues. energy [R]evolution. A Sustainable World Energy Outlook 2015. Technical report, Amsterdam, Brussels, 2015.

TLK-Thermo GmbH and Institut für Thermodynamik, Technische Universität Braunschweig. TILMedia 1.3.0 ClaRa, 2018. URL `https://www.tlk-thermo.com/index.php/en/software-products/tilmedia-suite`.

United Nations. Paris Agreement, 2015.

Model-Based Controls Development and Implementation for a Hydroelectric Power System

Anh Nguyen[1]   John Batteh[2]

[1]Modelon Inc., USA, {anh.nguyen,john.batteh}@modelon.com

Abstract

This paper describes the model-based control system development for a hydroelectric power plant to ensure water level control and mitigate spillage risk. The modeling of both the flume system and prototype controls is described. The integrated model is run over a suite of tests to verify the calibration of the control strategy. Results from the plant commissioning are compared with the virtual tests. The model proved capable of accurate predictions of the waterway dynamics, and the model-based calibration was successfully verified on the actual plant.

Keywords: hydro power, control systems; hydraulics

1 Introduction

Hydroelectric power plants can experience long lifecycles with plants often operating for decades. With years of historical data, it is not uncommon for plants to operate based on manual operator control. In an effort to improve power dispatch and uptime, optimize revenue, extend plant life, and improve reliability and safety, modern control solutions can be deployed and retrofit to existing plants at significant benefit. Model-based controls development is a critical element for any plant control modernization effort.

Hydro Power Library (Modelon AB, 2018) provides a framework for modeling and simulation of hydro power plant operation and control. The library provides a complete environment for modeling the plant system including the hydraulics, waterway dynamics, plant turbine and electrical, and associated controls including both dynamic and steady state operation. A model of the Sundsbarm hydro power plant in Seljorn, Norway was built using a previous version of the library to simulate and identify the reasons for power production variation at the plant including the development of a linearized model and a model predictive control (MPC) approach to optimize plant operation (Winkler *et al*, 2011). The library has been used to simulate a number of on and off-design operating conditions for the Fossárvirkjun power station in northern Iceland (Magnúsdóttir and Winkler, 2017) including a detailed electrical system modeled with Electric Power Library.

This paper describes the model-based development of a control strategy for a hydro power plant to ensure appropriate water level control for environmental impact due to spillage risk. Due to US security concerns for critical power infrastructure, the plant must remain anonymous with some sensitive data obscured. The paper will provide an overview of the waterway system model including model calibration. The control strategy is prototyped and integrated with the waterway system model. A virtual test suite is executed for model-based calibration and verification of the control strategy. Results from the model are then compared with data obtained from commissioning and testing on the actual plant. The model-based approach proved capable of predicting the waterway dynamics and for model-based calibration and verification of the control strategy.

2 Flume System Modeling

This section describes the physical model of the flume system. The model is assembled using Hydro Power Library (Modelon AB, 2018) along with some custom components. The full hydraulic system model is introduced and then individual components are described in more detail along with the calibration performed based on available data from the waterway prior to commissioning.

2.1 Full System Model

Figure 1 shows the full hydraulic system model starting from the intake of the flume system and ending at the turbine inlet. The model includes the following components:

- Intake gate with control
- Upper flume system with rectangular geometry
- Lower flume system with trapezoidal geometry
- Flume at spillway for increased resolution at critical area for spillage
- Basin and forebay reservoirs
- Penstock with valve control
- Visualizers for flow rate, water elevation, and volume at various locations in the flume system
- Visualizer component to show invert elevation, water elevation, and max elevation throughout the flume system

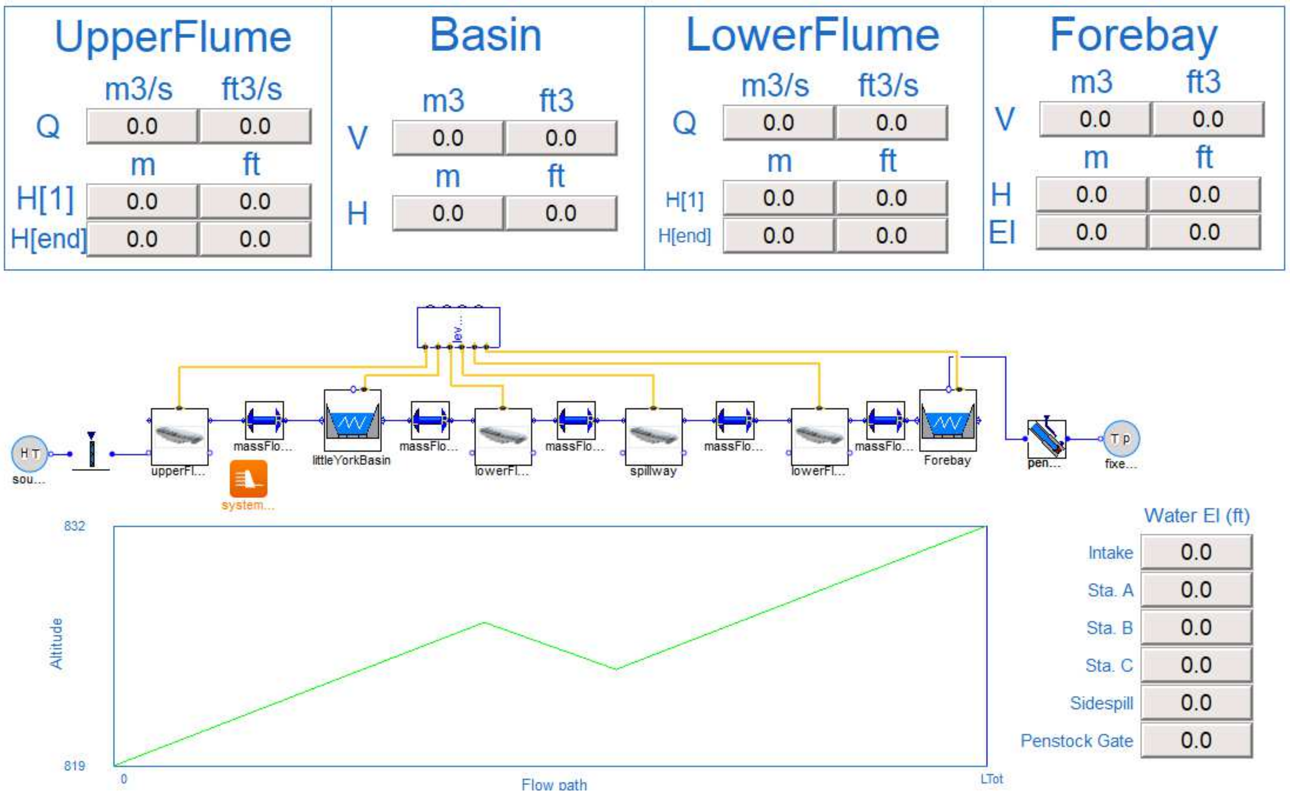


Figure 1. Flume system model including visualizers

Individual components are discussed in more detail in the following sections.

2.2 Intake Gate

The intake gate is modeled as a linear valve with the gate velocity as an input as shown in Figure 2. The valve velocity is integrated to provide a valve lift. The valve is characterized based on an estimate of the gate position at maximum flow.

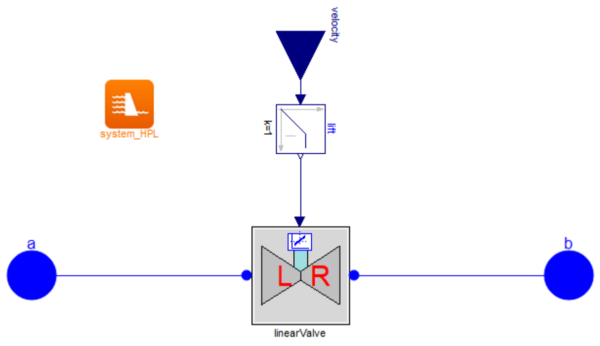


Figure 2. Intake gate model

2.3 Flumes and Spillway

The upper and lower flumes and spillway are modeled as reservoirs with varying elevation, width, cross section shape, and maximum height along the discretized length of the flume. The flumes are discretized models with combinations of volumes with open channel flow between adjacent volumes. The upper flume is rectangular shape and the lower flume is trapezoidal. The spillway is modeled as a separate component to allow increased resolution in the area that is most important for spill control. The upper flume is discretized into 80 segments while the lower flume and spillway include roughly 10 segments as they are significantly shorter.

Hydro Power Library provides models for friction in open and closed channel flow but allows flexibility for custom models to be implemented. For this work, a custom friction model was implemented and integrated into the reservoir model. The friction model is based on the standard Manning equation with the Kutter roughness coefficient (Sellin, 1970). The roughness coefficient for concrete ranges from 0.01 to 0.015.

Since the focus of the controls development is avoidance of spill, it is critical that the flume system flows correctly at different depths. Unfortunately no detailed information was available prior to commissioning for the flow of the overall system. However, operator setpoint data was available for the upper flume depth as a function of flowrate. A model of the upper flume was used to calibrate the roughness coefficient for the friction model as shown in Figure 3. The results from the calibrated model are shown in Figure 4 and compared with the setpoint data. Given that no additional flow information was available, the same coefficient was then used for the lower flume and the spillway.

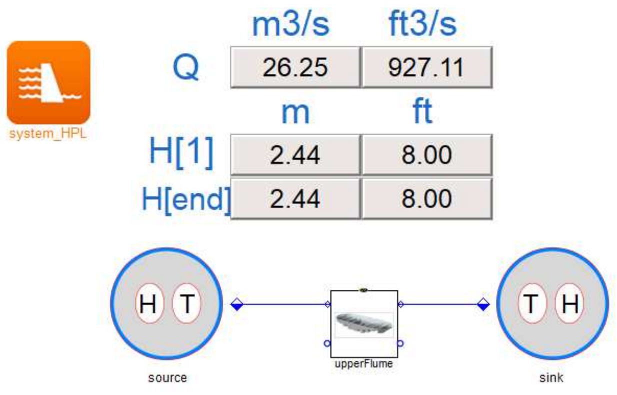


Figure 3. Calibration model for upper flume

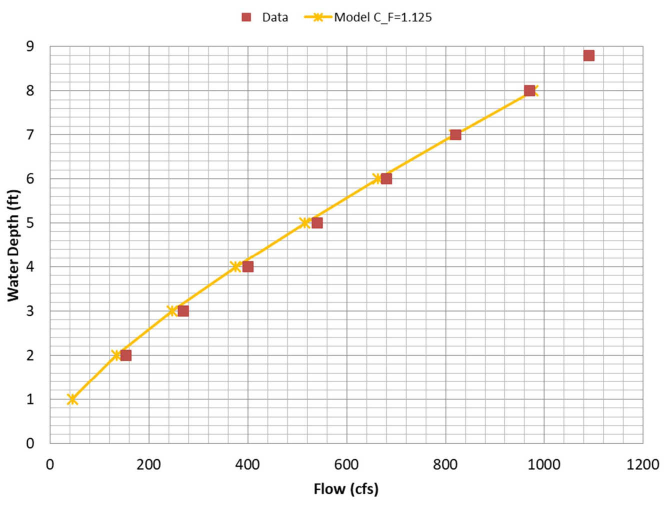


Figure 4. Calibration of upper flume friction based on operator setpoint data

2.4 Reservoirs

The reservoirs for the basin and forebay are based on the open volume component in Hydro Power Library. The open volume component contains conservation of mass and energy equations for a variable volume. The open volume component in the library allows geometry specification as shown in Figure 5. However, the reservoirs in the flume system do not map to a simple geometry specification as the reservoir geometry for the actual flume system is highly irregular due to topological variations of the geography. Thus, a custom component was created to allow a flexible specification of the depth and volume relationship as a table.

The data for the basin and forebay are shown in Figure 6. Using this table-based representation for the depth and volume relationship calculated from the actual reservoir geography, the model can accurately reflect the detailed, irregular geometry for reservoir capacity and depth without requiring a complicated geometric implementation.

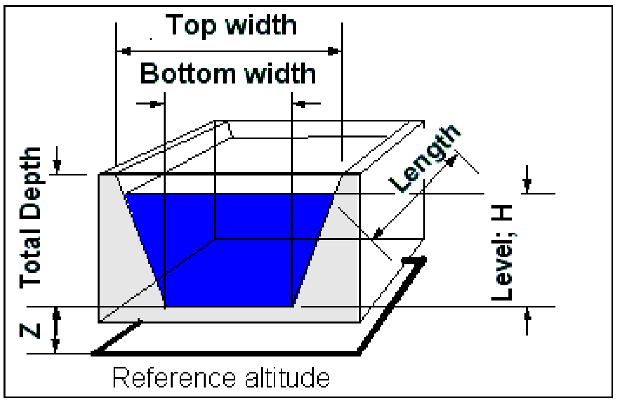


Figure 5. Base open volume component geometry specification in Hydro Power Library

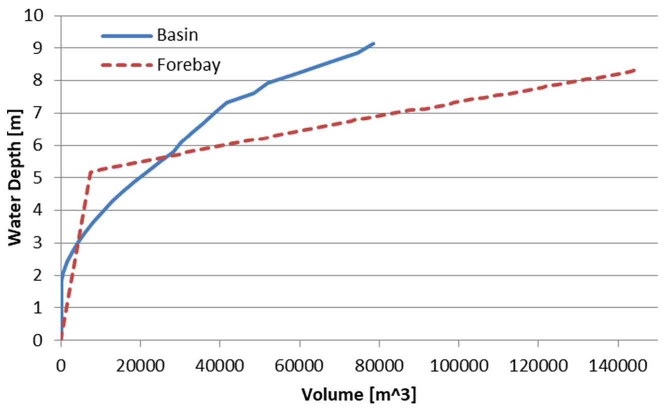


Figure 6. Reservoir depth-volume characterization

2.5 Penstock with Valve

The penstock that leads to the turbine inlet is connected to the forebay reservoir. The penstock valve is a linear valve that is characterized to deliver the maximum plant flow at a flow command of 1. Figure 7 shows the model used to characterize the penstock valve. This model includes the forebay at a specified height and then the penstock with a specified downstream pressure at the outlet. Based on the location of the penstock connection to the forebay, the hydrostatic pressure drives the flow in the penstock. The results from running the valve at the maximum flow command are shown in the visualizers in Figure 7.

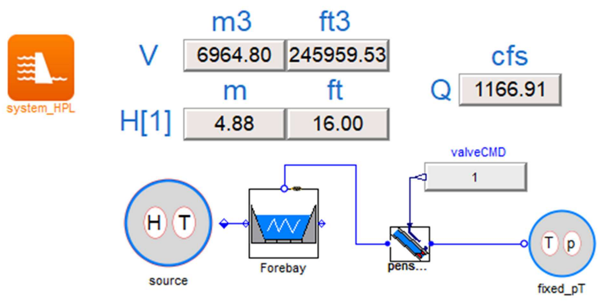


Figure 7. Test case for penstock valve characterization

2.6 Flume System Visualization

When simulating the flume system, results for water depth, flowrate, pressures, etc. at any location in the flume system are readily available. Figure 8 shows typical results from a flow rate step test. The time for a flow disturbance to travel downstream in the flume system is clearly shown by the delays in the elevation response at different locations along the waterway.

While these local results are valuable, they can be difficult to interpret to get a good overall picture of the spatial distribution of the water in the flume system, including waves. Thus, a custom component was developed to aggregate the information from the individual flume system components and visualize the entire flume system using the diagram layer animation features in Dymola (Dassault Systemes, 2018).

Figure 9 shows an example of the visualization of the flume system for a large change in inlet flow. The flow change begins from a steady state condition around t=320min. The visualization shows the invert elevation of the system (blue), maximum height of the system (green), and dynamic water elevation (red). The effect of the flow change is clearly seen progressing down the flume system, including wave behavior at different parts of the flume. When animated in Dymola, the visualization provides an animation of the flume system as a function of time and is critical for understanding the dynamics of the system and assessing the water depths relative to the spill limits. The combination of the detailed traces at a specific location as shown in Figure 8 and the overall visualization of the entire flume system as shown in Figure 9 provide a more complete view of the waterway dynamics.

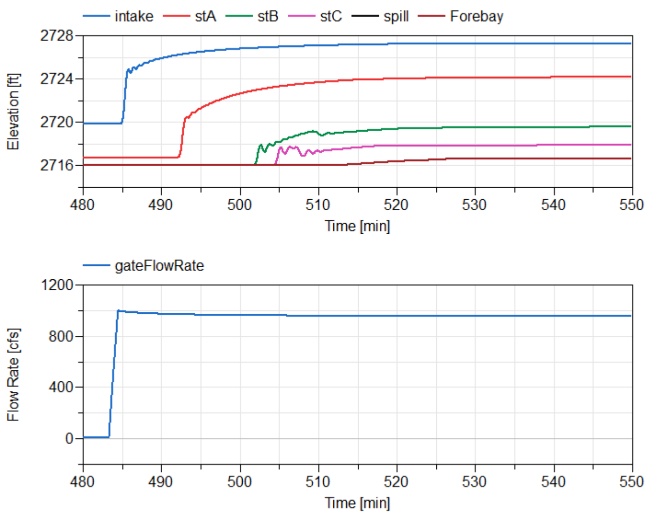


Figure 8. Results from flow rate step test

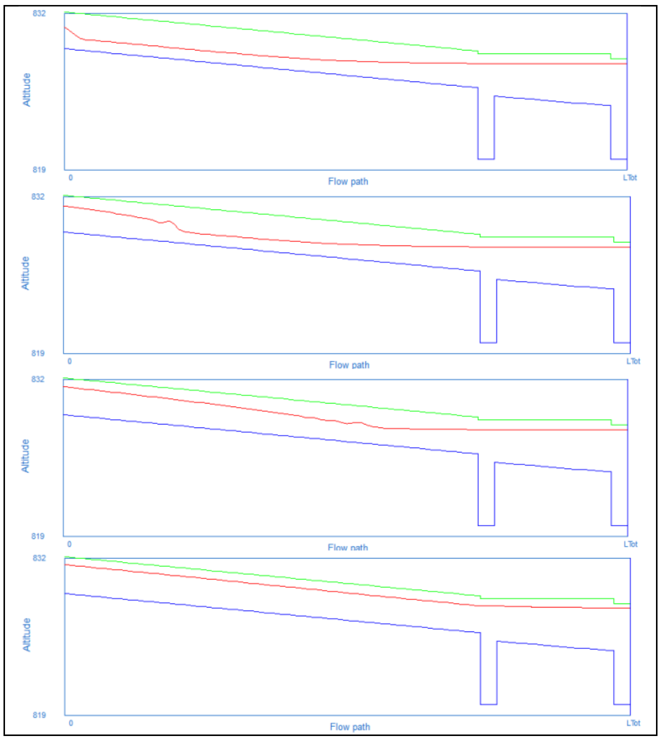


Figure 9. Flume system visualization for a large change in inlet flow at t=321 min, t=326.67 min, t=335 min, and t=350 min

3 Controls Development and Integration

For this plant system, there is concern for the environmental impact of any potential spill. Thus, the focus of the controls modernization is system control for spill. The control algorithm receives the forebay level as input and actuates the penstock valve. The actual discrete control algorithm is implemented in the model as in the hardware PLC logic and with the same calibration values. Due to confidentiality reasons, the actual control algorithm cannot be shown nor can the calibration values.

Figure 10 shows the aggregate penstock control block. This block includes several different control modes including open loop control, continuous control, and the actual discrete control implemented in the block *controller1*. The reason for this controller structure is to allow different operating modes for the system including:

- Open loop for manual operation
- Continuous control for computational efficiency in establishing test conditions for the simulation
- Steady state detection
- Discrete control algorithm under development

Since the controller can operate in various modes, it is important that the transfer to the actual discrete controller occurs without disturbance to the penstock command. Thus, bumpless transfer is implemented as in the actual controller to ensure smooth transitions.

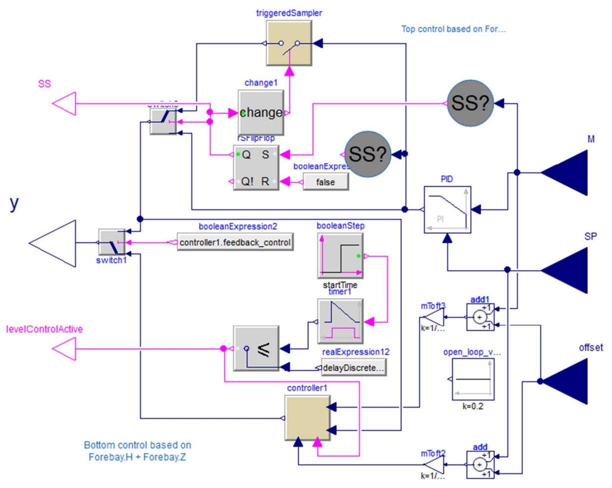


Figure 10. Penstock control including open loop, continuous control, and discrete control with switching

Figure 11 shows the model of the flume system with the integrated penstock control and gate control elements. This model is extended from the base flume system model in Figure 1 and simply adds the control elements. The gate control is simply for actuation of the intake system for testing purposes. The gate controller specifies the maximum gate opening velocity until the desired flowrate is achieved and then holds the gate position.

4 Virtual Controls Verification

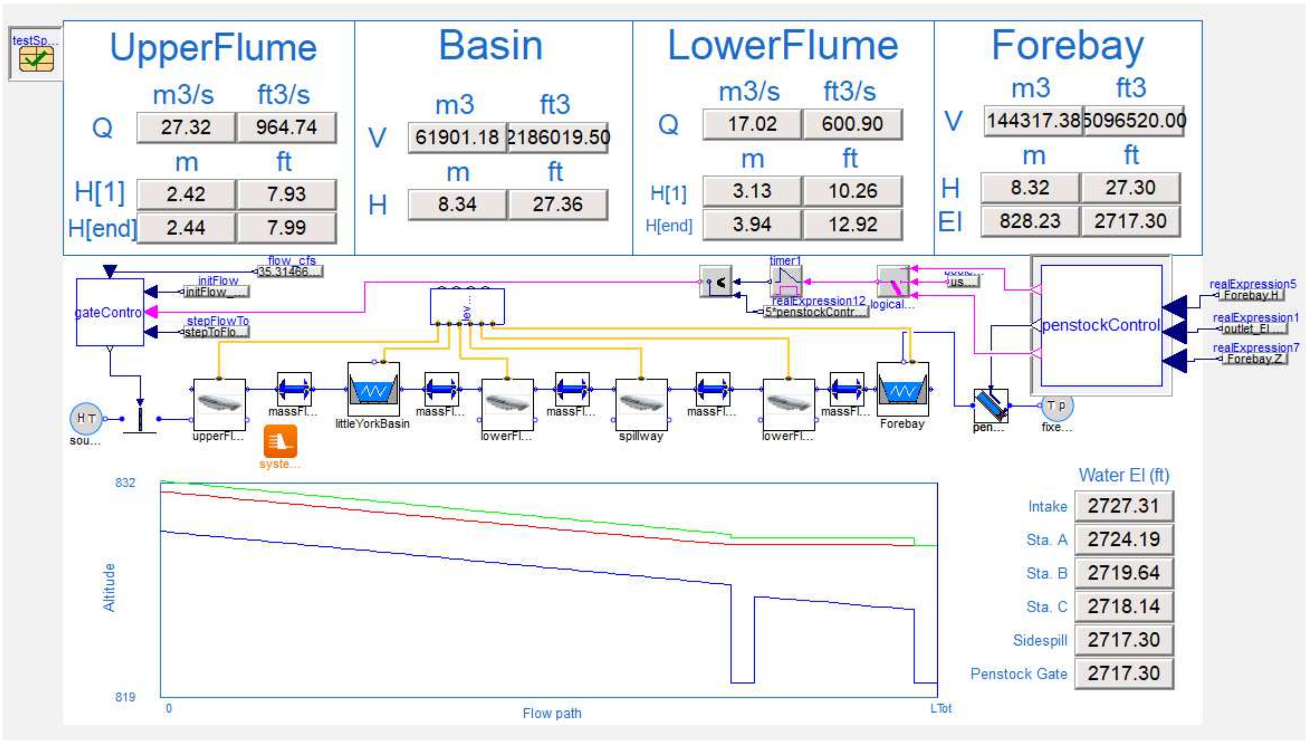
The integrated model in Figure 11 serves as the test bench for the virtual controls development, calibration, and verification. This section describes and gives results from a virtual test plan that was conducted prior to the commissioning of the controls integration on the plant hardware. The virtual test plan was developed to prove out the physical model and calibrate the proprietary control algorithm. Though formal controls methods are certainly applicable to ensure controller stability, the calibration of the control algorithm was performed via execution of the virtual test plan with a focus on operating conditions that pose the most severe spillage risk and were also planned for execution in the plant commissioning.

4.1 Flow Steps with Fixed Valve

To test the overall system response, a series of flow step tests were conducted as follows:

- Run to steady state at initial flowrate and system level using continuous control for efficiency
- Initiate flow step but with fixed penstock command from initial flowrate (i.e. no controller)
- Observe system response

Figure 12 shows the system response to a flow step change but with relatively low flows. As the flowrate increases, the elevation increases since the valve command is fixed at the steady state value from the initial flowrate conditions. Figure 13 shows the system response to a flow step change but with a step from low to high flow. Under this condition, the system elevation increases rapidly because the valve command is fixed at the value for a much lower flow. The simulation is stopped due to the excessive level.



	UpperFlume m3/s	UpperFlume ft3/s
Q	27.32	964.74
	m	ft
H[1]	2.42	7.93
H[end]	2.44	7.99

	Basin m3	Basin ft3
V	61901.18	2186019.50
	m	ft
H	8.34	27.36

	LowerFlume m3/s	LowerFlume ft3/s
Q	17.02	600.90
	m	ft
H[1]	3.13	10.26
H[end]	3.94	12.92

	Forebay m3	Forebay ft3
V	144317.38	5096520.00
	m	ft
H	8.32	27.30
El	828.23	2717.30

	Water El (ft)
Intake	2727.31
Sta. A	2724.19
Sta. B	2719.64
Sta. C	2718.14
Sidespill	2717.30
Penstock Gate	2717.30

Figure 11. Flume system model with integrated penstock control and gate control

For these tests and many of the subsequent tests, continuous level control is used to run the system to steady state at specified initial conditions. This control is simply to get the system to a desired initial state, and continuous control is used for computational efficiency. It should be noted that this control is completely separate from the actual discrete controls being prototyped and is only used to set the initial system state for testing purposes. Furthermore, no significant effort was spent calibrating the continuous control given its purpose to help initialize the system for dynamic testing. The continuous control can be used to simulate the system to a steady state and then hold the valve command to simulate manual control of the system.

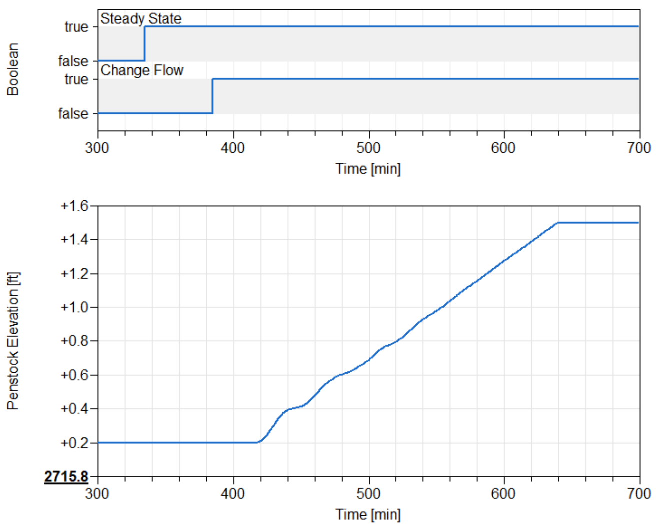


Figure 12. System response to flow step change, low flows

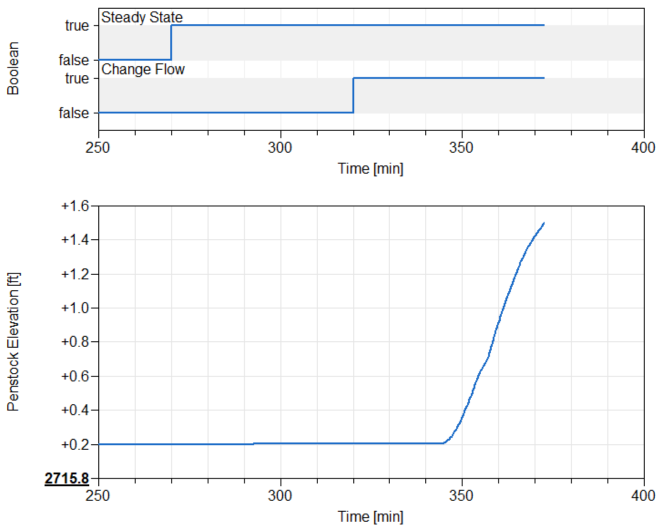


Figure 13. System response to flow step change, step to high flow

4.2 Level Setpoint Steps

A series of tests for controller stability were conducted as follows:

- At a specified flowrate, run to steady state at a specified system level using continuous control for efficiency
- Engage discrete controller and initiate a change in the system level setpoint
- Observe system and controller response

These tests were conducted at different flowrates and for both step up and step down in system level. Standard controller design metrics such as overshoot, undershoot, and settling time were used to calibrate the controller. Figure 14 shows results from a test with a step down in level control setpoint at both high and low flowrates. When the level setpoint change is initiated, the controller smoothly opens the penstock valve until the desired level is achieved and then ultimately returns to the initial opening since the flowrate is held constant in these tests. At the higher flowrate, the response is slower as the penstock command is saturated at maximum opening.

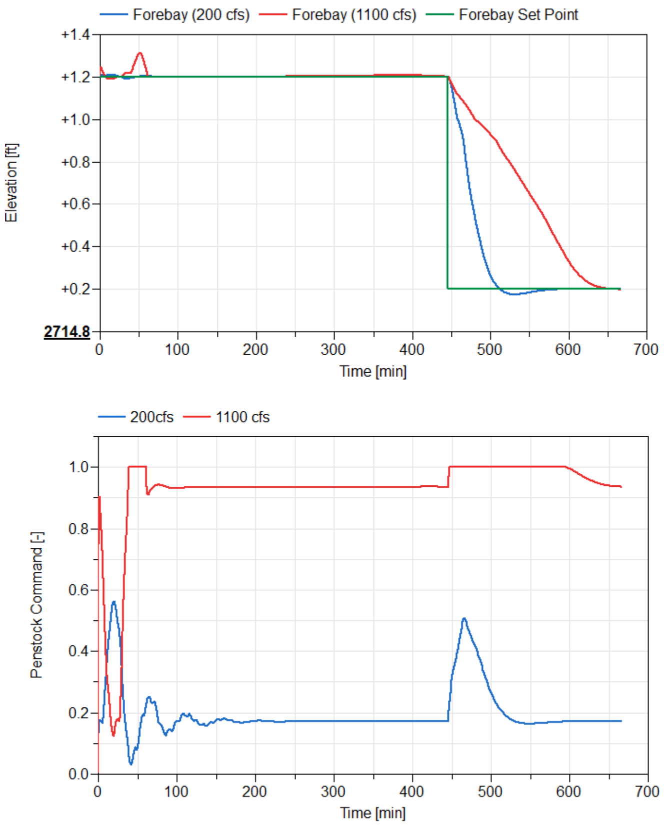


Figure 14. System response to level step down at low and high flows

Figure 15 shows results from a test with a step up in level control setpoint at low, medium, and high flowrates. When the level setpoint changes, the penstock valve closes to increase the system level.

Once the desired level is achieved, the valve command opens again to allow the system to flow at the new system level. At the lowest flow command, the penstock valve actually closes completely before opening again.

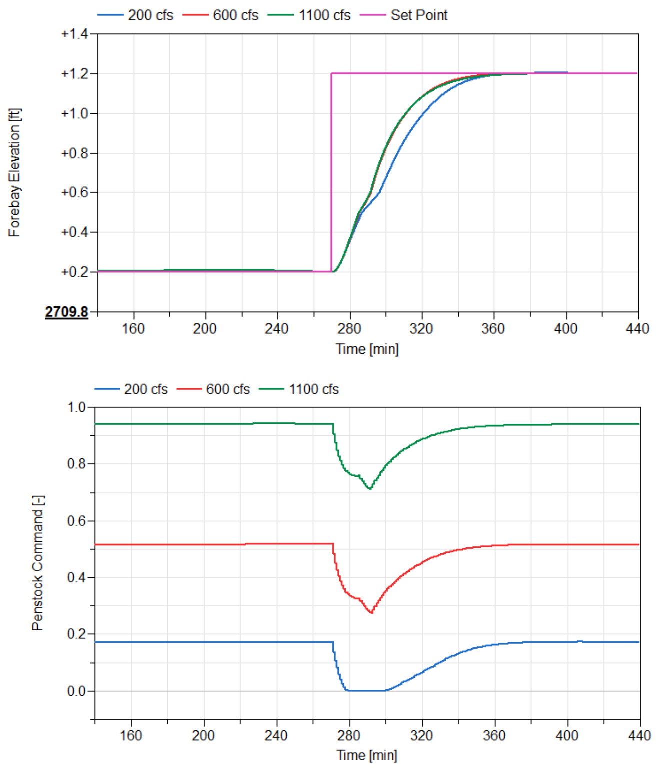


Figure 15. System response to level step up at low, medium, and high flows

4.3 Oscillation Tests

When there is a significant elevation difference between reservoirs, there is the potential for flow to oscillate between the reservoirs. A series of oscillation tests were performed to ensure that the controller did not cause the system to become unstable under this scenario. The tests were conducted as follows:

- At a high flowrate, run to steady state at a specified system level
- Switch off flow at intake to induce a level difference between reservoirs
- Engage active level control to observe interaction between active level control and level oscillations

Figure 16 shows results from the tests. When the flow at the flume intake turns off, the basin reservoir sees a drop in water elevation first. There is oscillating flow that exists between the two reservoirs as evidenced by the level oscillations. With active level control, the controller quickly closes the penstock valve to maintain the system level. The oscillating flow between the reservoirs eventually damps out, and there does not seem to be any adverse impact of the active level control on the oscillations. Though stability considerations are best evaluated via formal controls methods, these time domain test are useful for validating the controls performance.

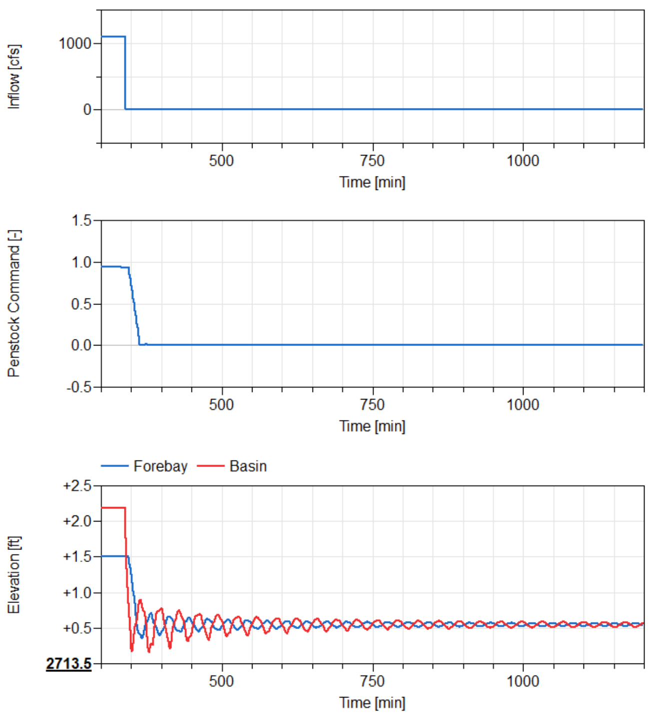


Figure 16. System response to oscillation test with active level control

4.4 Max Flow Step Tests

The highest risk of spilling occurs when the system experiences the maximum step from lowest flow to highest flow when the system level is high. To simulate this worst case condition, the following test was conducted:

- Run system to steady condition at very low flow and high system level with manual control
- Step to maximum inlet flow keeping manual control
- Observe controller and system response to flow increase

Figure 17 shows the response to the max flow step test. In this scenario, the controller as calibrated is able to arrest the system level and ensure that no spilling occurs. Figure 18 shows some sensitivity results to limits in the controller for the same flow step. With a low limit on the controller, the maximum elevation increases as expected.

These max flow tests were critical for the evaluation of the controller calibration and provided model-based verification of the controller under extreme conditions. As described in Section 5, plant commissioning conducted with the calibration developed from the virtual tests showed similar results.

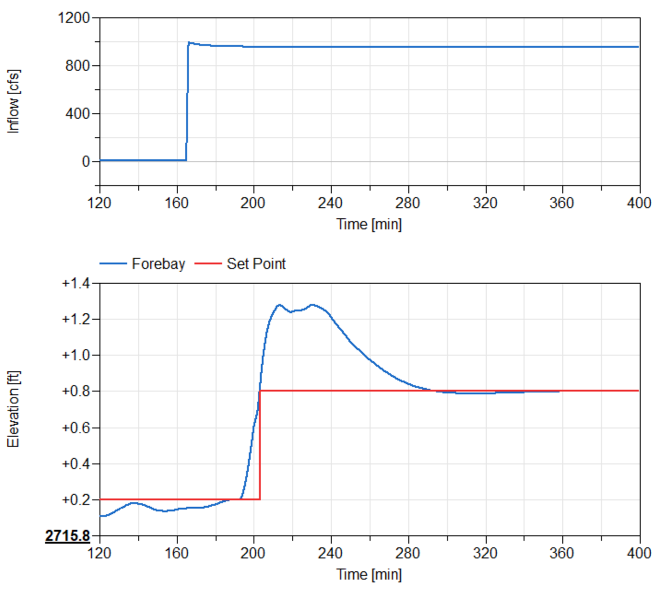


Figure 17. System response to max flow step test at high level

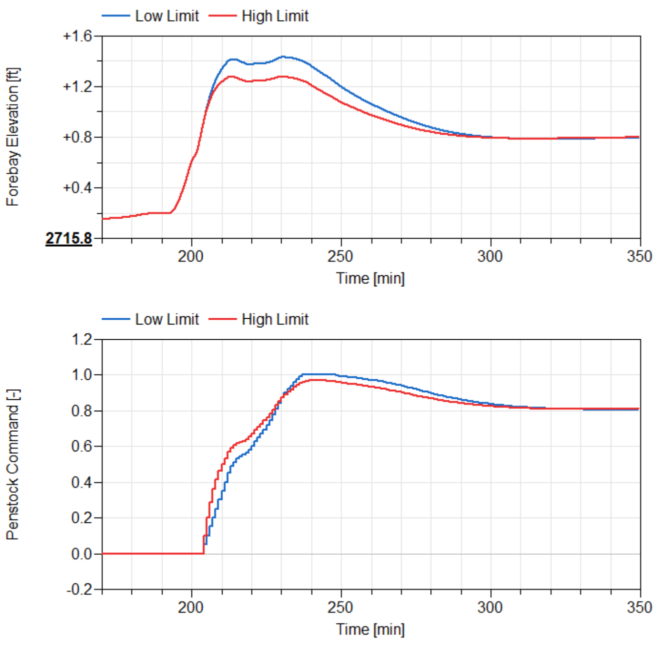


Figure 18. System response to max flow step test at high level, varying controller limits

Similar tests were also conducted at low system levels. The test scenario is as follows:

- Run system to steady condition at very high flow and low system level with manual control
- Step to very low inlet flow keeping manual control
- Observe controller and system response to flow decrease

Figure 19 shows results from this test. The controller is able to control the system level appropriately. Notice the long time constant required to increase the system level due to the very low inlet flow. Even with the penstock valve completely closed, it takes time to increase the system level due to the low intake flow. As expected, the system exhibits different time constants when subjected to flow steps up and down given the underlying volume dynamics required to change the system elevation.

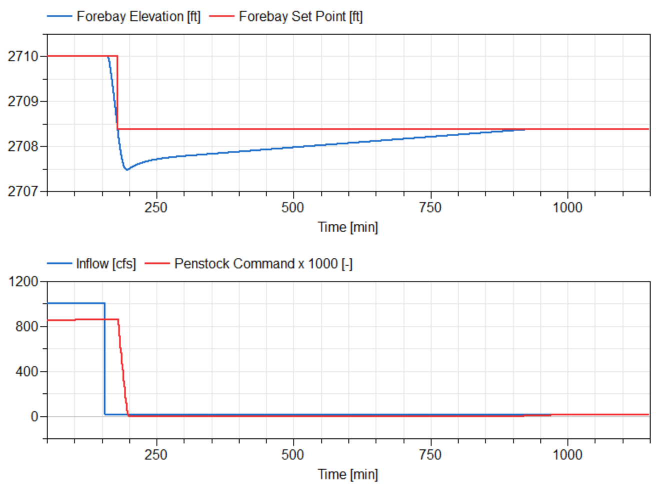


Figure 19. System response to min flow step test at low level

5 Plant Commissioning

After the model-based calibration and verification of the controller on the virtual test suite, the controller was commissioned on the plant. The controller as implemented in PLC logic for the plant level control is identical to that implemented in Modelica. A comprehensive test plan was executed as part of the commissioning to ensure signal integrity, controller response, actuator bandwidth and response, etc. over a wide range of operating conditions.

Selected results from the commissioning are shown below. The model-based calibration developed during the analytic work was successfully validated in the commissioning work. Only minor changes were required to handle some signal conditioning issues which were not anticipated and not simulated. Otherwise, the results predicted by the simulations were confirmed in the commissioning.

Figure 20 shows results from a step up test during the commissioning. The commissioning data is provided at 5 minute intervals. The intake setpoint is representative of the intake system flow. The system level response compares favorably to the similar test shown in Figure 17-Figure 18. The tests are not identical as they start at different initial levels. However, the response of the system level in terms of time constants and profile compare well. The calibration developed in the model proved capable of managing the spillage risk as commissioned on the plant.

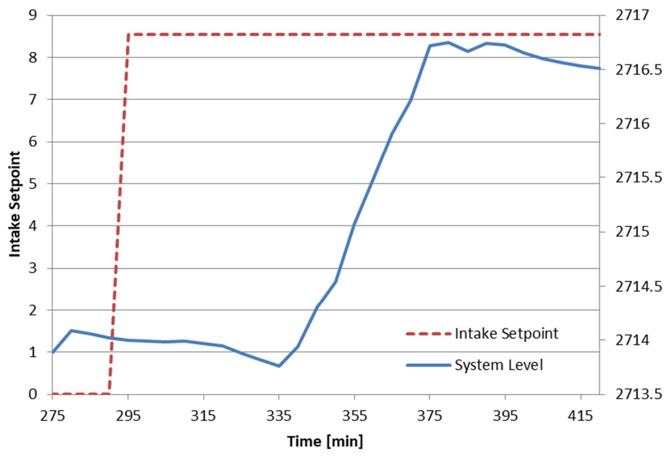


Figure 20. System response to max flow step test during commissioning

Figure 21 shows results from a step down test during the commissioning. Similar characteristics are seen when compared with the step down test in Figure 19 though the tests are not identical since the commissioning step down test starts from a high level while the simulations started from a low level. The simulations take much longer to respond due to the different starting level as the penstock command saturates.

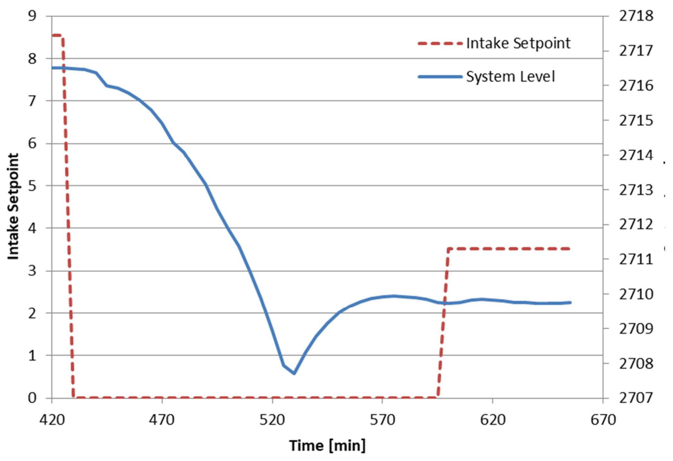


Figure 21. System response to min flow step test during commissioning

Based on the results from the commissioning, several different actions to improve the overall model-based controls development process were identified:

- Any flow data available when building the model is critical as it allows verification of the system response early in the development process
- Simulating not just the overall system response but also any dynamics in the sensor system will provide better input signals for the model-based calibration and potentially reduce/eliminate onsite calibration work for a more robust virtual calibration process
- Steady state initialization of the system model would reduce the time spent waiting for the system to reach steady state and avoid the extra logic in the controller to control the test conditions

6 Summary

This paper describes the model-based control system development for a hydroelectric power plant to ensure water level control and mitigate spillage risk. The paper gives an overview of the work to develop a waterway system model with Hydro Power Library and associated controls. The control algorithm was prototyped in the model, and a model-based calibration process was used to verify the algorithm and calibration over a virtual test suite.

Following the analytic work, the controller was commissioned on the plant and successfully verified based on a set of commissioning tests. The algorithm prototyped in Modelica is identical to the PLC logic implementation used on the plant. Results from the model compare favorably to the commissioning tests with only minor changes required to the calibration due to unanticipated signal conditioning issues. Overall, the model-based approach proved capable of predicting the waterway dynamics and for model-based calibration of the control strategy. Future work could include additional calibration of the model based on the commissioning data and adding capability to the model to capture the sensor system dynamics and thus enable an even more robust analytic calibration process. Formal controls methods, including a linearized model, to ensure controller stability are also considered as potential future work.

References

Dassault Systemes, Velizy, France (2018) Dymola 2018. https://www.3ds.com/products-services/catia/products/dymola

Magnúsdóttir, A. and Winkler, D., “Modelling of a Hydro Power Station in an Island Operation”, *Proceedings of the 12th International Modelica Conference*, May 15-17, 2017. Prague, Czech Republic.

Modelon AB, Lund, Sweden. (2018). *Hydro Power Library*. http://www.modelon.com/products/modelon-library-suite/hydro-power-library/

Sellin, R., *Flow in Channels*, Gordan and Breach Science Publishers, New York, 1970.

Winkler, D., Thoresen, H., Andreassen, I., Perera, M., and Sharefi, B., “Modelling and Optimisation of Deviation in Hydro Power Production”, *Proceedings of the 8th International Modelica Conference*, March 20-22, 2011. Dresden, Germany.

SESSION 4B: AUTOMOTIVE 2

Fault Insertion for Controller Calibration in a Range of Engine Models

Romain Gillot[*] Alessandro Picarelli[*] Mike Dempsey[*]
[*]Claytex Services Ltd, Edmund House, Rugby Road, Leamington Spa, CV32 6EL
{romain.gillot, alessandro.picarelli, mike.dempsey}@claytex.com

Abstract

This paper investigates the response of the Engine Control Unit (ECU) in relation to abnormal engine operation. Four different faults are introduced in a physical engine model at random times during simulation to observe the effects on the model results and system behaviour. It is then shown that the sensors used throughout the engine model are capable of detecting any type of fault using signals for quantities that would be measurable in a real engine. Finally, the soft ECU switches to emergency/limpo home operation and limits the engine performance to help prevent what could be mechanical damage in a real engine.

Keywords: *engine, fault, ECU, detection*

1 Introduction

Predictive modelling is becoming increasingly popular to dimension or calibrate systems prior to the prototype stage. If current models are capable of representing the behaviour of physical systems very accurately, they often only model the system's expected behaviour. Some particular tasks like Engine Control Unit (ECU) calibration require the model to also operate in undesired states to ensure that the controller detects the faults and acts accordingly (OBD diagnostics).

The aim of this paper is to introduce a range of faults in a multidomain/multi-physics engine model to demonstrate that it is capable of detecting and identifying these faults and of taking measures to limit their effect and/or to prevent further damage to the system.

The faults modelled are a leak in the air path of a turbocharged four-cylinder engine, a clogged injector in a naturally aspirated four-cylinder engine, a stretched timing chain in a naturally aspirated four-cylinder engine and a short-circuit in the TPS (Throttle Position Sensor) of a throttle body in a naturally aspirated four-cylinder engine.

The reason why we have chosen to focus on these faults in particular is because they allow to test components from all the domains involved in an engine model. Moreover, these are all faults commonly encountered in a real engine and ones that the ECU needs to be able to detect.

2 Presentation of the engine models

2.1 Crank-angle resolved

The engines used in this study come from the VeSyMA – Engines library developed by Claytex. They are all crank-angle resolved engine models with varying features and levels of detail depending on the area of interest.

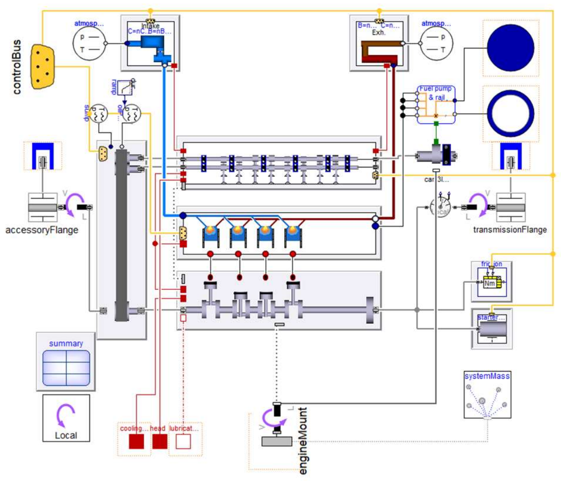


Figure 1. Crank-angle resolved engine model from VeSyMA – Engines.

The engine in Figure 1 is one of the most detailed versions available in the library [1] built referencing [2] and [3] mainly.

Some parameterisation adjustments have been made when necessary and simplifications when possible.

The mechanical systems can be reduced to 1-dimensional models (air leak, clogged injector, faulty throttle). The stretched timing chain example needs 3-dimensional mechanics since in the 1D one there is no mechanical link between the crankshaft and the cylinder head.

The surrogate model has been enabled (air leak, stretched timing belt, faulty throttle). The surrogate model is a subsystem that measures a set of physical quantities in cylinder 1 (intake air flow rate, cylinder pressure, etc.) and uses these values to drive ideal sources/actuators that are substituted to the physical fluid-based model in all the other engine cylinders. This helps to improve the model performance. The clogged injector example, however, needs all 3 cylinders to be enabled to allow for comparing their respective results and the variation in cylinder-to-cylinder performance.

2.2 ECU (Engine Control Unit)

The ECU is particularly important in this study since it is the subsystem that must be able to detect the faults and to react in relation to them.

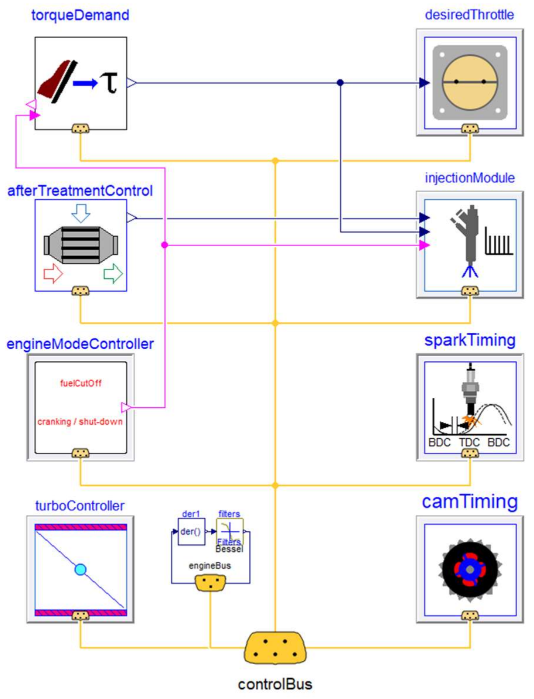


Figure 2. Engine Control Unit.Left-hand side modules from top to bottom: torque demand controller, after-treatment devices controller, engine mode controller, turbocharger controller. Right-hand side modules, from top to bottom: throttle controller, injection controller, ignition controller, valvetrain controller.

Figure 2 shows the software version of an ECU. It is used to validate the engine model before using the real ECU and doing Hardware-in-the-Loop testing. It must rely entirely on information collected by the engine sensors to detect a fault, like a real ECU would do.
It mostly uses PIDs and functions to control the engine actuators.

3 The faults

3.1 Pneumatic – air leak

A leak is introduced downstream of the intercooler and upstream of the throttle body.

It is modelled as an orifice connected to an infinite boundary at atmospheric pressure in parallel of the main air path.

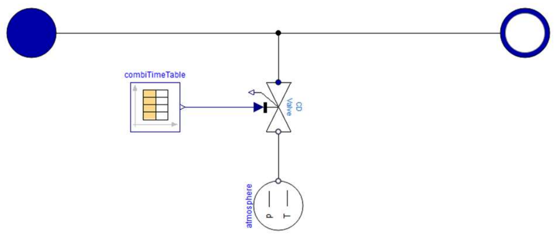


Figure 3. Air leak model with controllable orifice to ambient.

In Figure 3, port_a (blue) is connected to the intercooler outlet and port_b (white) is connected to the throttle inlet, in series. The valve discharge coefficient is entered in the time table, zero meaning no leak. This allows for the leakage to start at any moment. An external leak trigger can also be used.

3.2 Hydraulic – clogged injector

The injectors are modelled as ideal mass flow sources.

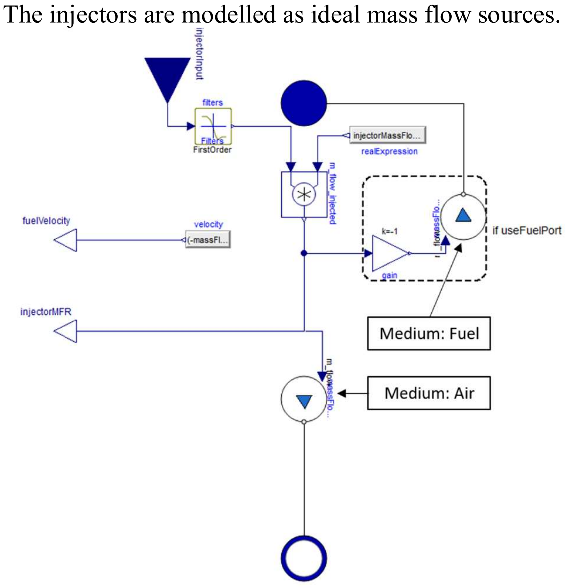


Figure 4. Injector model with optional fuel rail fluid port (filled blue round connector at top of diagram).

Two mass flow sources are used to separate the fuel path from the air path. The quantity of fuel to inject is calculated as the product of the nominal mass flow rate that the injector is capable of delivering by the injection pulse triggered by the ECU. This is the amount of fuel that the upper mass flow source in Figure 4 will remove (hence the negative gain) from the fuel line. The lower mass flow source will "inject" the same quantity into the combustion chamber.
The faulty injector is modelled in a similar way but the nominal mass flow the injector can deliver is determined by a time varying table. It is then possible to inject any fraction of what the injector should normally inject at any time.

3.3 Mechanical – stretched timing chain

In this model the timing chain is essentially an ideal gear ratio (ratio = 0.5, see Figure 5) located between the crankshaft and the camshaft flanges.

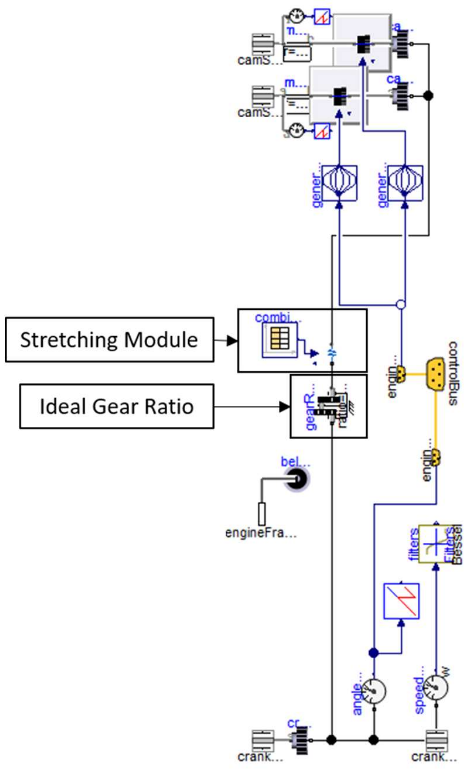


Figure 5. Timing chain module using ideal ratios with an angular offset to account for chain stretch.

The stretched timing chain module in Figure 5 is used to introduce a controlled degree of freedom between the crankshaft and the camshaft(s) angular.

In this case, we add an offset angle (between the crankshaft and the camshaft flanges) by means of a position actuator (see Figure 6). The chain is therefore still rigid but is stretched by a few millimetres.

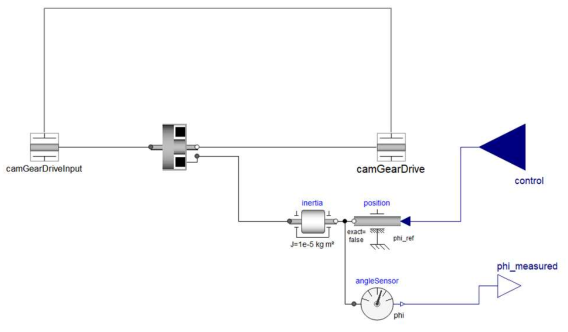


Figure 6. Compliance in the timing chain.

3.4 Electrical – short-circuit in the throttle board/wrong sensor measurement

The throttle body is a valve with varying discharge coefficient based on the plate opening angle (see 1 in Figure 7).

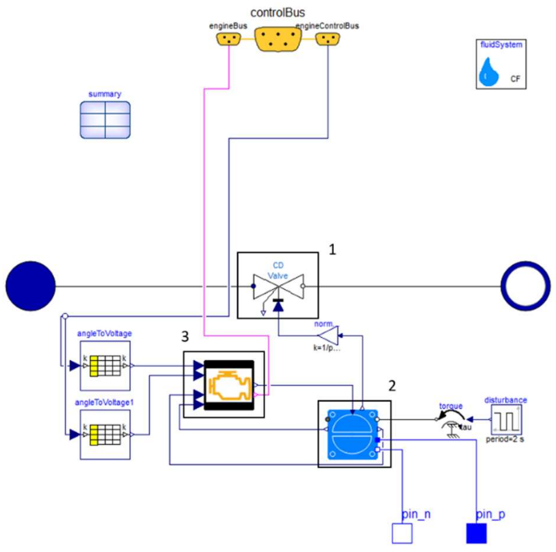


Figure 7. Throttle body with control board (1: pressure drop model, 2: mechanical system with potentiometer-based angle sensor, 3: electric board).

The throttle control board generates a current signal to drive the opening of the throttle plate (see section 4.4 for more info).

The throttle position sensor (Figure 8) uses a potentiometer to deliver a voltage as a function of the throttle opening angle.

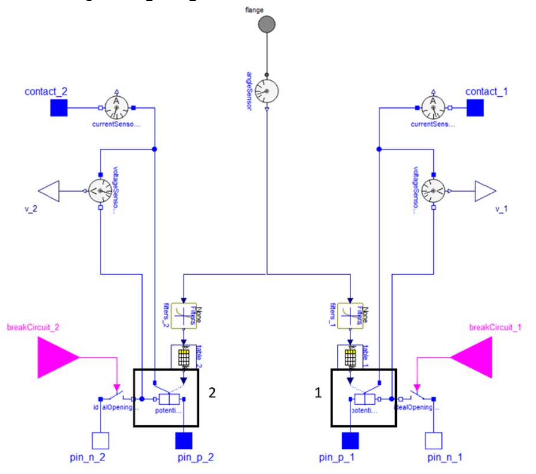


Figure 8. Throttle position sensor (1: potentiometer 1st track, 2: potentiometer 2nd track).

A second potentiometer is used to provide a valid value in case the first one fails and delivers half the voltage of the other one. Comparing these two voltages is another way of detecting a failure. The short-circuit is induced in the TPS by means of a switch breaking the electrical circuit.

4 Failure detection and simulation results

4.1 Air leak

We run our engine coupled to a fixed inertia with wide-open throttle for 10s.

A leak corresponding to an orifice with a diameter of 0.04 m is introduced at t=2.5s. The leak diameter has been chosen rather large in order to see significant and short-term effects on the engine.

We compare the boost pressure from the plenum pressure sensor to the values mapped during normal operation. If they fall below a threshold, the leak is detected, and limp mode is activated which limits throttle opening to a small value.

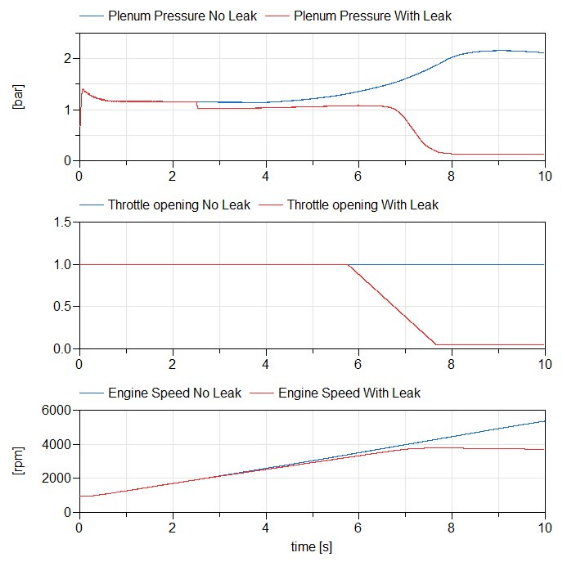


Figure 9. Plenum pressure, normalised throttle opening and engine speed under normal operation (blue) and with a leak (red).

We can see in Figure 9 that plenum pressure drops slightly when the leak starts. When it goes below the threshold at t=5.75s, limp mode gets activated and the maximum allowed normalised throttle opening is limited to 0.05 (i.e. 4.5 deg). Limiting throttle opening allows to keep engine torque below a certain threshold to prevent further damage. The value of this threshold depends on the engine manufacturer and type.

4.2 Clogged injector

The quantity of fuel injected by one of the injectors is limited for a few engine cycles to 25% of what the other injectors deliver.

The misfire detection monitor, which comprises of a set of logic blocks, compares the maximum crankshaft acceleration during every cycle to a value corresponding to normal operation based on an inferred torque for the set of given operating conditions. If a misfire is detected in the same cylinder during two consecutive cycles (can be changed to any number of consecutive cycles), it is considered a fault and the engine malfunction light goes on and limp mode gets activated. The detection monitor is not active during fast transients when the cylinder pressure varies a lot from one engine cycle to the next, like during aggressive tip-ins or tip-outs for example.

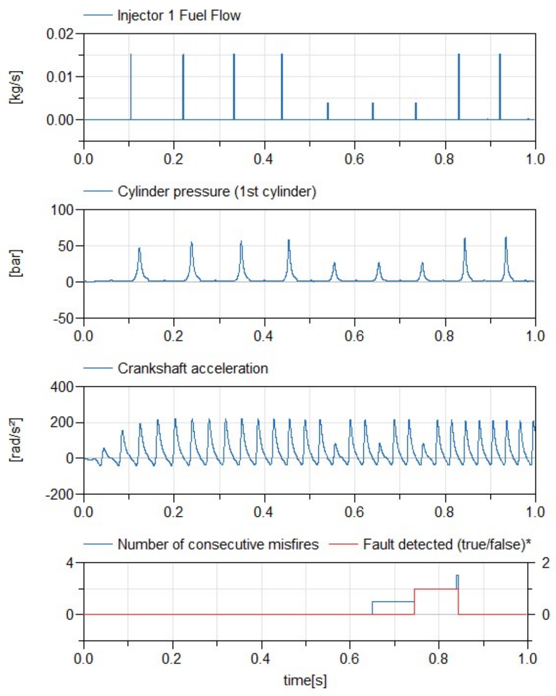


Figure 10: Injector fuel flow rate, cylinder pressure, crankshaft acceleration and number of consecutive misfires.

At about 5.5s, when the injector starts to inject less fuel than it should, the peak pressure in the cylinder decreases by more than 50%. This has a direct effect on crankshaft acceleration (see subplot 3 in Figure 10 above). During the three consecutive cycles in which cylinder one misfires, we can observe irregularities on the crankshaft acceleration curve in figure 9 (at about t=5.5s, t=6.5s and t=7.5s).

The acceleration of the crankshaft is computed from the crankshaft speed which is calculated from its angular position. The crankshaft angle is given by a crankshaft position sensor, this is then the physical quantity that will help the ECU to detect the abnormal behaviour.

4.3 Stretched timing chain

The timing chain is allowed to stretch by a few millimetres causing a slight desynchronisation between the opening of the intake and exhaust valves and the pistons positions.

The ECU detects a mismatch between the angle given by the crankshaft and the camshaft position sensors.

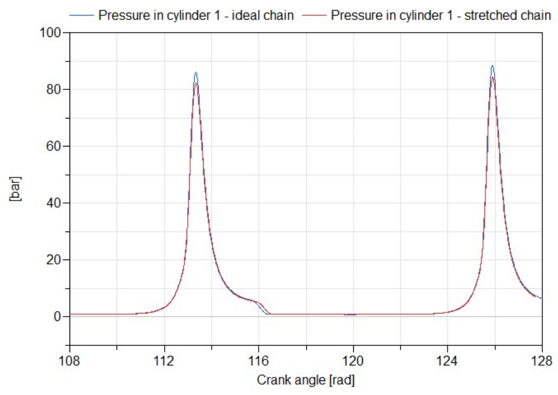


Figure 11: Pressure trace in cylinder 1 with an ideal chain (blue) and a stretched chain (red).

After a few cycles, the peak pressure reaches 88.4 bar when the chain is not stretched and the valve timing is ideal. It only reaches 84.4 bar (4.5% drop) when the chain is stretched by 5 degrees. The pressure trace for cylinder 1 is plotted in Figure 11. Cylinder 1 has been chosen arbitrarily but the results would be the same in all the other cylinders. The reduction in cylinder pressure is due to a larger valve overlap that causes the combustion chamber to not fill in with air as much as it could. The fuel injection module in the ECU maintains the air-fuel-ratio to 14.67 which means that less fuel gets injected leading to a lower power output.
The reduction in cylinder pressure compared to the expected value for a given engine speed and throttle opening is a way for the ECU to detect the fault. Combined with a mismatch in readings from the crankshaft and camshaft position sensors, it allows to identify the cause of the problem.

4.4 Faulty throttle body

A short-circuit is introduced during the simulation.

When the driver presses the accelerator pedal, the pedal position sensor (which is similar to the throttle position sensor in Figure 8) transmits a normalised angle signal to the ECU and reads a voltage using a potentiometer. Depending on the engine operating conditions, the ECU will determine the correct throttle opening. To this throttle opening angle corresponds a voltage (volts) that the throttle position sensor will compare to the value the accelerator pedal position sensor measured.

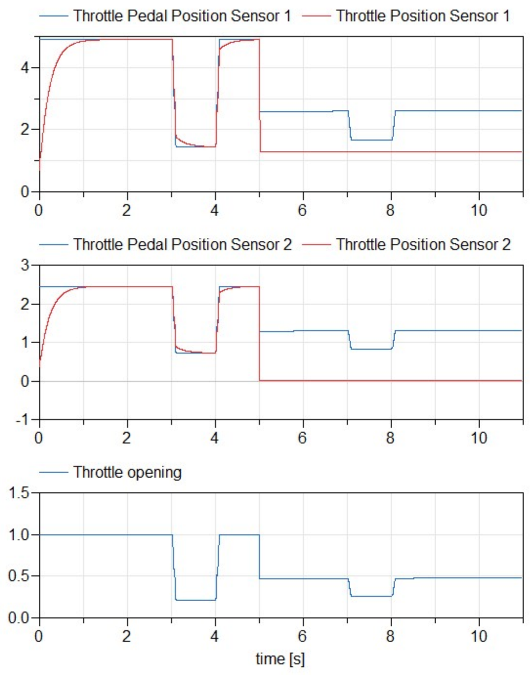


Figure 12: Comparison of the voltages from both channels of the throttle pedal position sensor and throttle position sensor. The bottom plot is throttle opening.

A short-circuit is introduced at time=5s in the second channel of the TPS which causes the voltage in the second channel to drop to 0. At this time, the ECU goes in to safety mode and the electrical current to the throttle plate spindle magnet is cut to zero which effectively corresponds to a fast idle setting for the throttle plate.

5 Conclusion and future work

Faults have been introduced in a crank-angle resolved engine model. The results before and after the faults occur have been presented to highlight their impact on the engine behaviour.

It has been demonstrated that the soft ECU can detect them relying solely on the information coming from the sensors and that the engine model can therefore be used for controller calibration tasks.

The next steps are to export this engine model as an FMU to a real-time platform and to test it with a real ECU. To do so, we would have to make further simplifications to the model for it to achieve real-time performance.

6 Bibliographic References

[1] Dempsey M. Picarelli A. Investigating the MultiBody Dynamics of the Complete Powertrain System. Como, Italy: Proceedings 7 th Modelica Conference, 2009.
[2] Heywood, B. Internal Combustion Engine Fundamentals McGraw-Hill.
[3] Robert Bosch Gmbh Gasoline Engine Management Bentley Publishers 2006.

Enhanced Motion Control of a Self-Driving Vehicle Using Modelica, FMI and ROS

Nikolas Schröder[1] Oliver Lenord[2] Ralph Lange[2]

[1]Institute of Flight Mechanics and Control, University of Stuttgart, Germany,
lrt86824@stud.uni-stuttgart.de
[2]Robert Bosch GmbH, Germany, {oliver.lenord, ralph.lange}@de.bosch.com

Abstract

This paper presents a new planar wheel model with bore friction, a control strategy to avoid locking conditions of floor vehicles with caster wheels, and the new FMI-Adapter software package, which connects the Functional Mock-up Interface (FMI) standard with the Robot Operating System (ROS). It is demonstrated how this technology enables a convenient model-based control design workflow. The approach is applied to the ActiveShuttle, a self-driving vehicle (SDV) for industrial logistics. After modeling the wheel friction characteristics of the ActiveShuttle, a feed forward controller to avoid high friction torques at the caster wheels in critical operation scenarios is designed and validated by model-in-the-loop simulations. The control function is exported as Functional Mock-up Unit (FMU) for co-simulation. With help of the FMI-Adapter package, the FMU is integrated as ROS node into the service-oriented robot control architecture, enhancing the existing motion controller. The functionality and performance is tested and successfully verified on the ActiveShuttle Dev Kit prototype.

Keywords: Modelica, FMI, ROS, Autonomous Systems, Robotics, Model-based Control, SDV, Caster Wheels

1 Introduction

A relevant application area of autonomous robotics is the industrial logistics. In the last years, a number of elaborate algorithms for task scheduling, coordination and path planning for fleets of self-driving vehicles (SDVs) in such applications have been proposed (Imlauer et al., 2016; Pecora et al., 2018). Prerequisite to apply these strategies is a reliable vehicle motion control. Trajectories commanded by the planner need to be properly executed by the drive platform to ensure that the goals of the mission are met in time and space. Safety and security margins have to be met, undesired interference due to deviations from the planned track need to be avoided, and at all times the vehicle must remain maneuverable.

Model-based control design is a well established approach to design and apply motion control strategies. Model-in-the-loop (MiL) simulations allow to validate and test the control design early on. Optimized controllers can be designed that take the physical properties and system dynamics into account (Thümmel et al., 2005).

In this work a common problem of motion platforms with differential drive and caster wheels is elaborated. By applying a model-based control design approach the reliability of the motion controller is significantly improved by the so called *Path Filter* introduced in Section 3.

The path filter module is realized as *ROS node* (see section 1.2) to allow the seamless integration into the service oriented software architecture ROS that is used on the target application *ActiveShuttle DevKit*. The widely used development environment for model-based control, Matlab Simulink, does provide a dedicated toolbox for ROS (The MathWorks). In this paper an alternative approach is applied aiming to leverage the benefits of the physical modeling language Modelica and the rich Modelica libraries such as the Modelica Standard Library (MSL) for the design, verification and validation of the path filter. For this purpose the free `PlanarMechanics` library (PML) (Zimmer, 2014) is extended and used to build up a plant model of the Active Shuttle DevKit (see section 2).

In order to enable a generic and efficient control design process, the integration of the path filter control function into ROS is facilitated through the *Functional Mock-up Interface* (FMI) (Modelica Association Project "FMI", 2014). Related approaches for such integration aim at simulation use cases only: The *Modelica-ROS Bridge* (Swaminathan) allows to integrate the Modelica language and corresponding tools with ROS by a TCP/IP-based bridge, implemented by the ROS_Bridge package for Modelica and a relay node from the ROS modelica_bridge package. A similar mechanism based on Unix IPC sockets has been proposed to integrate Modelica with the Gazebo simulator, which is used heavily by the ROS community to simulate robots in 3D environments (Bardaro et al., 2017). The *gazebo-fmi* project (Traversaro et al.) provides a plug-in to import FMUs in Gazebo.

With the new *FMI-Adapter for ROS* introduced in Section 4, a very generic mechanism is provided to integrate control functions into ROS. An export of the path filter block as FMU (Functional Mock-up Unit) allowed the straightforward integration into the ROS architecture and finally its application and test, as described in Section 5, on the Active Shuttle DevKit.

1.1 Use Case: ActiveShuttle

The *ActiveShuttle* (AS) is a self driving vehicle (SDV) for logistic services on the shop floor. A prototypical small batch series named *ActiveShuttle DevKit* operates at several Bosch plants in Germany. The missions carried out by the AS are bring and pick-up services of stacked container boxes on moving dollies.

The routes between the pick-up and drop-off locations are planned based on a map continuously updated by a simultaneous localization and mapping (SLAM) algorithm (Durrant-Whyte and Bailey, 2006). In order to make the motion of the SDVs predictable for the workers, the routes are restricted to be composed of straight segments only. Turns are restricted to be performed as turns on the spot after standstill only. Thus, the basic motion patterns are:

- Follow a straight line segment.
- Stop and turn on the spot.
- Stop and move in reverse direction.

The AS DevKit is actuated by a differential drive. Two caster wheels are positioned in the front and two in the rear. The chassis design ensures that all six wheels remain in contact to the ground while crossing sills or other uneven grounds.

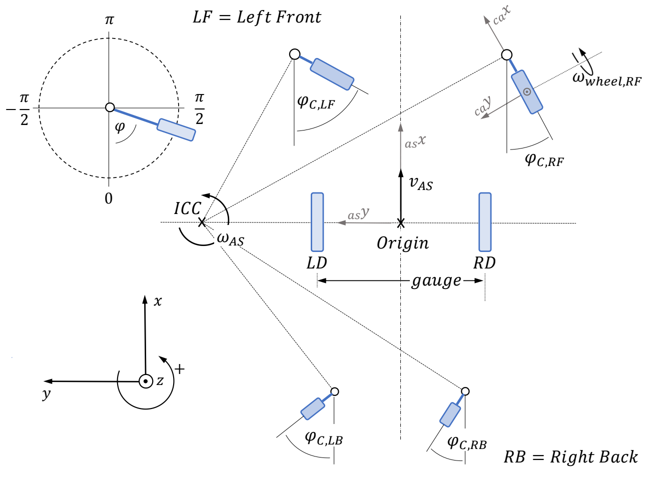


Figure 1. Active Shuttle coordinate system definition, sign convention and nomenclature.

The reference coordinate system of the SDV is fixed to the middle frame and located in between the driven wheel axes when standing on flat ground. Given that the vehicle is operated in the plane only, the motion state of the AS can be described by equation 1 with the longitudinal velocity v_{AS} and the angular velocity ω perpendicular to the moving plane.

$$\boldsymbol{x}_{AS} = \begin{pmatrix} v_{AS} \\ \omega_{AS} \end{pmatrix} = \begin{pmatrix} _{AS}v_{x,AS} \\ _{AS}\omega_{z,AS} \end{pmatrix} \tag{1}$$

During operation it has to be ensured that the above described motion patterns can be executed in an arbitrary sequence. A critical condition occurs in the transitions from moving straight to turning and vice versa. In these transitions the desired motion state of the AS is inconsistent with the actual motion state of the caster wheels given by equation 2, with the $\omega_{C,i}$ describing the angular velocity of the i-th caster wheel w.r.t. its spinning axis and $\varphi_{C,i}$ describing the orientation relative to the vehicle w.r.t. the vertical axis. The caster wheels' rotational axes are not aligned with the instantaneous center of curvature (ICC) of the SDV, located at the center of the reference frame:

$$\boldsymbol{x}_{C,i} = \begin{pmatrix} \omega_{C,i} \\ \varphi_{C,i} \end{pmatrix} = \begin{pmatrix} _{CA}\omega_{y,C,i} \\ _{AS}\varphi_{C,i} \end{pmatrix} \tag{2}$$

Due to the fact that the vehicle is at standstill when the turn is initiated, the maximum bore friction torque has to be overcome in addition to the inertial forces. Projecting the friction torque at the i-th caster wheel onto the point of contact of the driven wheels with the radii from the ICC to the driven wheel r_{DW} and caster wheel r_{CW}, reveals that due to $r_{DW} = gauge/2 < r_{CW,i}$ a significant share of the available traction force at the driven wheels is assigned to the bore torque of the caster wheels:

$$F_{DW} = \frac{r_{CW,i}}{r_{DW} \cdot r_{trail}} \cdot T_{bore} \tag{3}$$

If under full load the required driving torque exceeds the maximum motor torque, the SDV is not able to follow the commanded trajectory. Hence, avoiding the risk of this critical state by reducing the impact of the bore friction has been identified as significant contribution to make the operation of SDVs with differential drives more reliable.

1.2 Robot Operating System

The Robot Operating System (ROS) (Quigley et al., 2009) has been used for the development of the AS DevKit. ROS can be considered as a framework and middleware for robotic systems. It also provides a rich set of development tools and basic functional capabilities for perception, control, planning and manipulation. In the last ten years, a huge open-source community has grown around this project, which provides numerous software packages for all aspects of robotics (`www.ros.org/browse/`).

ROS uses a service-oriented architecture with publish-subscribe and request-response communication methods. It even allows to integrate new components dynamically at run-time. ROS supports most prevalent programming languages, particularly including C++, Python, Java, C#, JavaScript, and Ruby. These features facilitate the integration of new technologies with ROS.

2 Physical Model of the SDV

2.1 Model Requirements and Architecture

Based on the use case described in Section 1.1, the following physical effects have been identified that need to be represented by a physical model of the SDV:

- Planar motions of the SDV relative to a fixed ground and related inertial forces.

- Normal forces at the driven and caster wheels considering the chassis kinematics and mass.
- Limited traction of the driven wheels considering the maximum wheel slip.
- Bore friction torque at the caster wheels with stiction.
- Limited drive torque defined by the motor characteristics.

Due to the usage as plant model for MiL simulations with slow accelerations and limited maneuvers, the following simplifying assumptions have been applied:

- No real time requirements.
- No inertial torque along the longitudinal axis.
- Neglect the inertial torque along the lateral axis.

Aiming to keep the physical model as simple and generic as possible the `PlanarMechanics` Library (PML) (Zimmer, 2014) has been chosen as basis for the mechanical model. The library provides all elements required to describe a planar multibody system and provides a set of basic tire models referred to as `WheelJoint` that have been adopted as described in Subsection 2.2. To enable a simple reuse of these basic tire models for the described class of differential drive vehicles, additional components have been developed combining the wheel joints with components from the Modelica Standard Library (MSL).

In a separate `SelfDrivingVehicles` library the extended PLM has bee utilized to build up packages for specific applications such as the AS DevKit and others. The corresponding `System` package contains models that describe the connections between the driven wheels and caster wheels as well as a block to calculate the normal forces of the wheels dependent on the actual orientation.

The `Controller` package contains models of common control concepts applicable to any differential drive vehicle such as motion profiles described in vehicle coordinates and their mapping to command values for the left and right drive as well as implementations of the *Path Filter* introduced in Section 3.

This architecture allows to separate the application specific properties from common concepts of differential drive vehicles. Models of new applications can be created with little effort.

2.2 Wheel Models for Differential Drive

Driven Wheel. The wheel itself is modeled with a `SlipBasedWheelJoint` (PML) and a 1D-rotational `Inertia` component (MSL). The `frame_a` connector of the `SlipBasedJoint` interfaces the driven wheel subsystem with the main structure of the vehicle model. The `Inertia` component is actuated by a `Torque` signal.

Caster Wheel. Figure 2 shows a schematic illustration of a swivel caster wheel. The fork contains a bearing which allows the wheel to swivel relative to the structure it is mounted on. The bearing is modeled with a `Revolute` joint (PML). Its `frame_a` connector interfaces the subsystem with the main structure of the vehicle model. A `RelAngleSensor` (MSL) measures the current caster wheel orientation and provides it as `RealOutput`. The fork is modeled with a `FixedTranslation` component (PML) with the length $l = trail$ that connects the `Revolute` with the wheel joint. As specified in Section 2.1, the wheel joint of the caster wheel is required to consider bore friction. Bore friction is a friction torque that counteracts a wheel's rotational movement around its vertical axis. As depicted in figure 3, bore torque is denoted with T_{bore} and is opposed to the acting torque T_z and the angular velocity ω_z. In the following, we first describe a bore friction characteristic that was proposed by (Zimmer and Otter, 2010). Thereafter, we propose an alternative approach that better suits our model requirements and explain how the `IdealWheelJoint` is extended to allow its correct implementation.

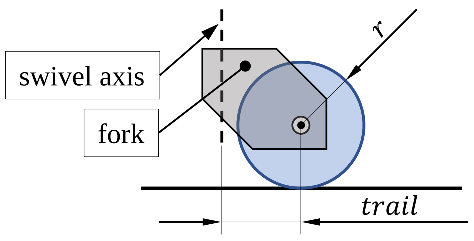


Figure 2. Nomenclature of a caster wheel.

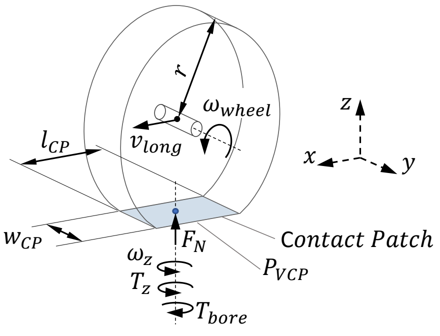


Figure 3. Tire road contact.

Equation 4 shows the proposal by (Zimmer and Otter, 2010).

$$|T_{bore}| = \begin{cases} |T_{bore,max}| \cdot \dfrac{|\lambda_{bore}|}{\lambda_{bore,lim}} \\ \quad \text{if } |\omega_z| \cdot s_{rep} < \lambda_{bore,lim} \cdot |\omega_{wheel}| \cdot r \\ |T_{bore,max}| \quad \text{else} \end{cases} \tag{4}$$

Similar to (Rill, 2007), they state that the friction torque T_{bore} is proportional to the bore slip λ_{bore}. Here, bore slip describes the ratio between the representative slip velocity $\omega_z \cdot s_{rep}$ of the tires contact patch and the wheels linear velocity $\omega_{wheel} \cdot r$.

$$|\lambda_{bore}| = \frac{|\omega_z| \cdot s_{rep}}{|\omega_{wheel}| \cdot r} \tag{5}$$

Within the contact patch, s_{rep} is the distance between the virtual contact point (P_{VCP}) and an infinitesimal element which is representative for the distances of all infinitesimal elements. It results when integrating over the contact patch.

$$s_{rep} = \frac{1}{\sqrt{12}} \sqrt{l_{cp}^2 + w_{cp}^2} \tag{6}$$

The bore torque is limited to

$$|T_{bore,max}| = F_N \cdot \mu_{bore} \cdot s_{rep}. \tag{7}$$

Here, μ_{bore} denotes the friction coefficient between tire and road, F_N the wheel contact force that acts on P_{VCP} (cf. figure 3). The limiting parameter in equation 4 is the bore slip limit $\lambda_{bore,lim} > 0$ that determines at which bore slip value the maximum bore torque is reached. Due to the fact that the actual bore slip λ_{bore} is not defined for $\omega_{wheel} = 0$ (cf. equation 5), the condition $\lambda_{bore} < \lambda_{bore,lim}$ in equation 4 is expressed such that division by zero is avoided.

Figure 4 shows a visualization of equation 4. When $|\omega_z| = |\omega_{wheel}| = 0$, the acting torque $|T_z|$ has to overcome the stiction torque $|T_{bore,stic}| = |T_{bore,max}|$. As long as $|T_z| < |T_{bore,stic}|$, the caster wheel is caught in a locking condition and cannot change its orientation around its vertical axis.

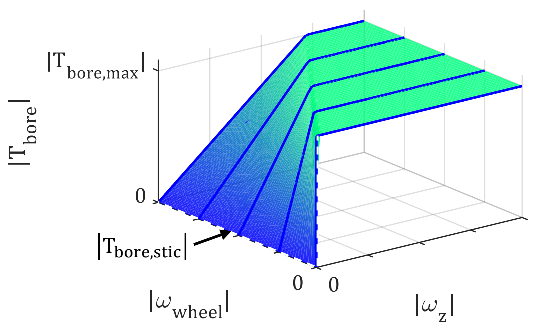


Figure 4. Bore friction characteristic (Zimmer and Otter, 2010).

In a first modeling approach equation 4 has been implemented by a regularized s-shaped characteristic as it is used in the `SlipBasedWheelJoint`. However, simulations revealed that a vehicle at standstill actuated by a constant driving torque that is considerably smaller than the expected break-off torque $|T_{bore,stic}|$ at the single caster wheels, would still start moving over the course of a few seconds. This undesired effect can be explained by using a continuous function to avoid the discontinuity which requires $|\omega_z| > 0$ rad/s when $|T_{bore}| > 0$. Even though the s-shaped characteristic can be tuned such that ω_z is comparably small when reaching $|T_{bore}| = T_{bore,max}$, still it is just a matter of time until the growing share of the acting forces pointing in the longitudinal direction of the caster wheels accelerate ω_{wheel} which leads to rapidly decreasing bore friction.

In order to properly capture the locking behavior and despite loosing real time capabilites by introducing events, the model equation 4 was implemented with help of the hybrid friction formulation used in the MSL `BearingFriction` model. However, it was found that this second approach is still not sufficient. Due to the fact that the friction characteristic (cf. figure 4) drops quickly from $T_{bore,max}$ to zero for $\omega_{wheel} \neq 0$, very small deviations of $\Delta \approx 10^{-3}$ rad/s allow the wheel to brake free. Hence, the behavior of the AS DevKit cannot be replicated with the bore friction characteristic introduced in equation 4.

In order to avoid the undesired effect described above, an alternative approach is presented in the following. In contrast to the previous model the new friction characteristic, shown in figure 5, has a linear slope w.r.t. ω_{wheel} near zero, such that small deviations do not take effect. This leads to the following formulation of the bore torque:

$$|T_{bore}| = \begin{cases} (|T_{bore,max}| - |T_{bore,stic}|) \cdot \dfrac{|\lambda_{bore}|}{\lambda_{bore,lim}} + |T_{bore,stic}| \\ \quad \text{if } |\omega_z| \cdot s_{rep} < \lambda_{bore,lim} \cdot |\omega_{wheel}| \cdot r \\ |T_{bore,max}| \quad \text{else} \end{cases} \tag{8}$$

with $T_{bore,stic}$ defined as

$$|T_{bore,stic}| = \max(0, |T_{bore,max}| - k_{stic} \cdot |\omega_{wheel}|). \tag{9}$$

For $k_{stic} \to \infty$, equation 8 tends to equation 4.

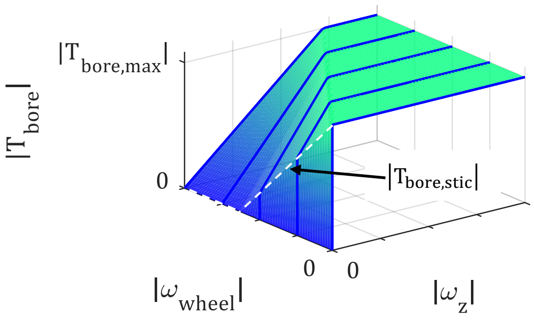


Figure 5. Alternative bore friction characteristic.

In order to implement equation 8, the new model `IdealWheelJointBore` combines concepts of the `IdealWheelJoint` (PML) and the `BearingFriction` (MSL) model. This allows to represent the locking condition explicitly as discrete state to assure that $T_{bore} = -T_z$ as long as $|T_z| < |T_{bore,stic}|$ (while $\omega_z = 0$). The `BearingFriction` component is extended with a `RealInput` to provide the current wheel contact force F_N. As the `Bearing Friction` component requires the exact torque T_z that is applied to the wheel, the wheel joint is additionally extended with a rotational `SpringDamper` component (MSL). This allows to dynamically resolve the distribution of the overall driving torque to the caster wheels despite a statically over-determined system when multiple caster wheels are connected to the same frame.

Calculation of the wheel contact forces. This subsystem provides a mathematical model for the calculation of the wheel contact forces. It is a simplified static approach that takes the current caster wheel orientations into account. However, the lateral dynamics of the AS, which have an influence on the wheel contact forces when accelerating or decelerating, are neglected due to rather small accelerations compared to gravity.

2.3 Verification of the ActiveShuttle Model

The verification of the AS simulation model focusses on the plausible replication of the observered behavior of the caster wheels and motion of the vehicle body, especially w.r.t. to the critical operation scenarios (cf. section 2.1). This is considerd sufficient in order to prove the path filter concept in Section 3.2. Therefore the model is not validated against measurements of the real system.

Turn on the spot. Figure 6 shows the simulation results for a desired clock-wise (CW) turn on the spot with two different masses. With $m_{AS} = 150$ kg, the AS is capable to initiate the turn and overcome $|T_{bore,stic}| = |T_{bore,max}|$ at all four caster wheels. However, at maximum payload (total mass $m_{AS} = 250$ kg), it is caught in a locking condition. Hence, ω_{AS} remains zero. Since $|T_{bore,max}|$ is remarkably higher in the fully loaded case, the drives of the AS are not strong enough to overcome the stiction torques at the caster wheels.

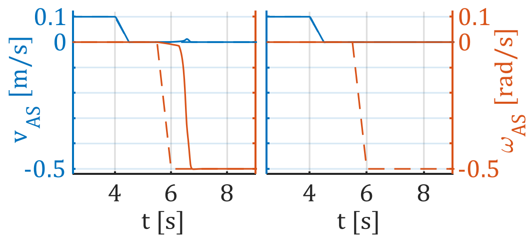


Figure 6. CW turn on the spot with $m_{AS} = 150$ kg (left) and $m_{AS} = 250$ kg (right). Dashed lines represent desired motion, solid lines the simulated motion. All caster wheels with $\mu_{bore} = 0.6$, $k_{stic} = 1$ and $\lambda_{bore,lim} = 1$.

Flipping caster wheels. Figure 7 shows the simulation results for a desired transition from driving straight forward to straight backwards. Due to the prior motion segment, all caster wheel orientations are 0 rad when $v_{AS,des}$ is reversed to -0.3 m/s. It can be noticed that the caster wheels are not changing their orientation by half a turn to π rad instantly. Instead the caster wheel's orientations start to flip randomly after about $t = 9$ s. This reflects the behavior to be observed at the real vehicle.

3 Design of the Path Filter

In this section a feed-forward controller is described that ensures that the AS continues its motion in the direction of the current caster wheel orientations before it is smoothly transferred in the direction of the desired orientations.

3.1 Control Architecture and Model

Figure 8 depicts the architecture of the path filter. Here, the index $(.)_C$ is representative for one of the four caster wheels. Moreover, ω_C is equivalent to $\omega_{C,wheel}$.

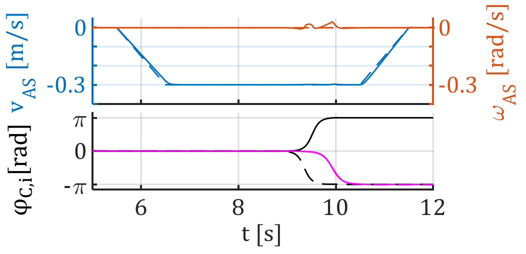


Figure 7. Random caster wheel flip with $m_{AS} = 250$ kg. Top: cf. figure 6. Bottom: Simulated caster wheel orientations. Black and magenta lines represent back and front caster wheels, resp. Dashed and solid lines represent right and left side, resp. All caster wheels with $\mu_{bore} = 0.6$, $k_{stic} = 1$ and $\lambda_{bore,lim} = 1$.

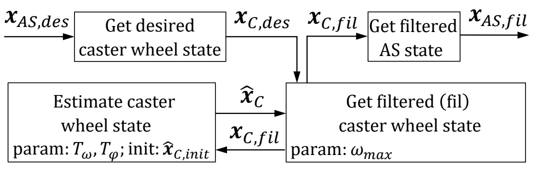

$x_{AS,des}$
Get desired caster wheel state
$x_{C,des}$
$x_{C,fil}$
Get filtered AS state
$x_{AS,fil}$
Estimate caster wheel state
param: T_ω, T_φ; init: $\hat{x}_{C,init}$
$\hat{x}_C$
$x_{C,fil}$
Get filtered (fil) caster wheel state
param: ω_{max}

Figure 8. Architecture of the path filter feed-forward controller

Get desired caster wheel state The path tracker of the motion control architecture commands the desired motion in the AS state representation. However, the path filter algorithm requires this information in the caster wheel state representation. Therefore, the geometric correlation between the two state representations is taken into account in order to provide a transformation. Hence, the subsystem requires the wheel radius and the position of the swivel axis of one representative caster wheel as parameters.

Estimate current caster wheel state We assume that all filtered states can be realized by the AS without encountering a locking condition or other external influences that keep it from reaching the commanded state. Consequently, it is taken for granted that the actual caster wheel state $\boldsymbol{x}_C$ is transferred into $\boldsymbol{x}_{C,fil}$ with a certain delay. Here, the delay is caused by the inertia of the AS. Hence, we calculate the estimate $\hat{\boldsymbol{x}}_C$ with two first order hold elements. The two time constants T_φ and T_ω can be tuned to match the AS dynamics.

Get filtered caster wheel state The actual filter algorithm calculates first the deviation between the desired and estimated caster wheel orientation. Equation 10 makes use of the modulo function in order to assure that $\Delta\varphi_C \in [-\pi;\pi]$ rad. This ensures that the AS takes the shortest path to its desired state.

$$\Delta\varphi_C = mod(\varphi_{C,des} - \hat{\varphi}_C + \pi, 2\pi) - \pi \tag{10}$$

Equation 11 states the calculation scheme for the filtered caster wheel orientation. Here, the factor k determines which proportion of $\Delta\varphi_C$ can be overcome in one calculation cycle.

$$\varphi_{C,fil} = \hat{\varphi}_C + k \cdot \Delta\varphi_C \tag{11}$$

According to Equation 12, k is defined to grow proportionally with the estimated caster wheel speed $\hat{\omega}_C$. Its definition is based on the idea that the faster a wheel is rolling, the easier it can be turned (cf. equation 4 and 5).

$$k = min(1, |\frac{\hat{\omega}_C}{\omega_{max}}|) \tag{12}$$

When the caster wheels are rolling very slowly ($\hat{\omega}_C \approx 0$ rad/s), the AS is forced to start moving in the direction of its current orientation $\hat{\varphi}_C$. Once the caster wheels have picked up some speed, their orientations change with $|\omega_{C,z}| > 0$. In other words, the path filter describes a trajectory in the bore friction characteristics of the caster wheels that avoids the peak friction torques (cf. figure 5). ω_{max} represents the slope of k. Small values reduce the time Δt that is necessary to transfer between estimated and desired state, but at the same time describe a trajectory in the bore friction characteristic that encounters higher bore torques. Hence, the choice of ω_{max} can be seen as a trade-off between transfer time and effort.

Get filtered AS state This subsystem represents the inverse of the first subsystem. It provides a transformation from the filtered caster wheel state to the filtered AS state. Here, it is important to provide the parameters of the caster wheel that was used with the first subsystem.

3.2 Path Filter Verification

In this section, we examine the path filter behavior with the help of two test setups. In both cases the path filter is set up with respect to the right front caster wheel (C,RF) and fed with a desired motion profile that includes the typical operation scenarios (cf. section 1.1). The "standalone" test case examines the input output behavior of the path filter, while the second "MIL" simulation includes the AS model as system plant and aims to show its impact on the physical behavior of the AS.

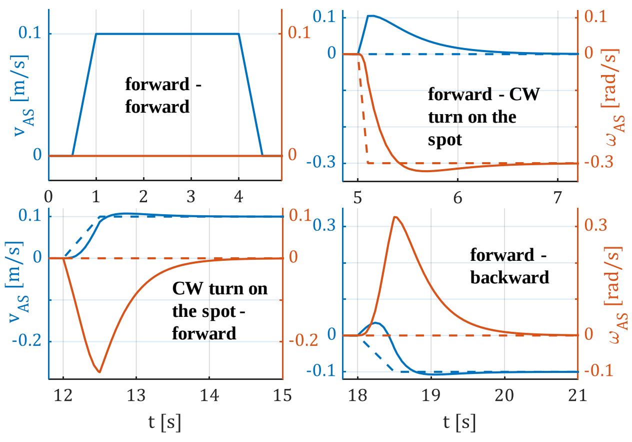


Figure 9. Standalone simulation of the path filter with ω_{max} = 100 rad/s and $T_\varphi = T_\omega = 0.01$ s. Dashed lines represent the desired motion, solid lines the filtered motion. For better scaling, the single motion scenarios are plotted separately.

The plots of the standalone simulation depicted in figure 9 show the path filter input ($\boldsymbol{x}_{AS,des}$) and output ($\boldsymbol{x}_{AS,fil}$). Here, the four motion scenarios are simulated in one sequence. Between each scenario, the AS is fully stopped. The plot on the top left reveals that $\boldsymbol{x}_{AS,des} = \boldsymbol{x}_{AS,fil}$ for the first motion segment. Here, the desired caster wheel orientation for driving straight forward is equal to the orientation that the estimation subsystem was initialized with (cf. figure 8).

$$\varphi_{C,RF,des} = \hat{\varphi}_{C,RF,init} = 0 \tag{13}$$

Consequently, $\Delta\varphi_{C,RF}$ remains zero during the first motion scenario and the filtered state is set to the desired state (cf. equation 11 and 10). For all other motion segments, the behavior of the path filter can be nicely seen. The commanded state $x_{AS,fil}$ makes the AS continue its prior motion before it smoothly passes into the desired state. When changing from driving straight forward to backward (bottom right plot), an angular velocity component is consciously induced by the path filter and avoids the random caster wheel flip.

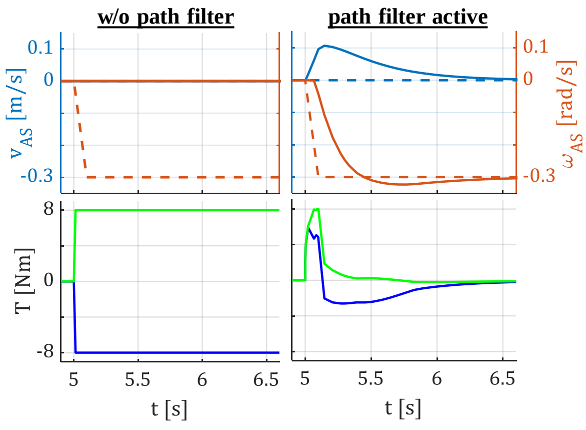


Figure 10. MIL simulation for a CW turn on the spot. The AS was driving straight forward prior to that. Top row: dashed lines represent the desired motion, solid lines the simulated motion of the AS model. Bottom row: green lines represent the left drive, blue lines the right drive. Path filter parameters: cf. figure 9. AS model parameters: m_{AS} = 250 kg, $T_{LD/RD,max} = \pm 8$ Nm, caster wheels as in figure 6.

Figure 10 shows the MIL simulation results for a CW turn on the spot. As motivated in Section 3, the path filter is able to avoid the locking condition that occurs in the setup without path filter.

4 FMI in ROS Control Architecture

In this section, we first describe relevant mechanisms and features of ROS, before we provide details on the FMI-Adapter package. Thereafter, we explain the integration of the Path Filter FMU with the ROS-based navigation architecture of the AS DevKit.

4.1 ROS Concepts

ROS uses a service-oriented architecture based on two common middleware mechanisms: publish-subscribe and request-response. Next, we briefly describe relevant ROS concepts:

Nodes. A software component is named node in ROS. Each node runs as a separate (Linux) process. Yet, a node may be instantiated multiple times, e.g., to run the same motor driver node twice for the two motors of a differential drive. To be able to distinguish two running node instances of the same executable, ROS provides a hierarchical naming scheme.

Topics. A topic is a typed and named n-to-m communication channel. Any node may open a *publisher* on a topic and *publish* (i.e. send) a *message* on it. This message is delivered to all nodes that have *subscribed* to that topic. The type of messages on a topic is defined by the first publisher. The `std_msgs` package of ROS provides message types for all primitive data types (i.e., bool, char, int, float, etc.). The packages `sensor_msgs` and `geometry_msgs` provide message types for common sensor data (e.g., laser scans, camera images, inertial measurements) and geometric primitives (e.g., points, poses, transformations), respectively. An interface definition language (IDL) allows to define application-specific messages types.

Services and actions. Using the same IDL and naming concept, services implement a typed request-response mechanism. Actions are a mechanism for long-running services, where the client may *preempt* the request.

Topics, services and actions are implemented with TCP/IP or UDP/IP. Therefore, the nodes can be distributed easily to different machines.

ROS master. The master is a dedicated process that provides a registration and lookup for nodes, topics, services and actions.

Parameter server. The parameter server provides a shared dictionary of typed key-value pairs, following the node naming scheme.

Launch files. A launch file is an XML-based specification to start a whole (sub-)system consisting of multiple nodes with corresponding parameterization. The specification language also allows to rename topics and services to connect nodes that have been developed independently.

Time and clock. ROS represents time by two 32 bit values (seconds and nanoseconds) since the epoch. In normal operation, the computer's clock is used as time source. Yet, ROS also allows a simulated clock with varying rate.

Packages. They are used to logically organize the software in ROS. A package may contain one or more nodes, a third-party library, a set of message types, launch files, etc. A package may specify dependencies to other packages, which are used for the build and installation process.

Callbacks and spin thread. Inside a node, each subscription, service server, and action server is associated with a callback function. Incoming messages are pushed to a first-in-first-out queue, which is processed sequentially by the spin thread by calling the corresponding callback function with the message data. ROS also allows to multiple callback queues and spin threads inside a node.

Timers. In addition to these communication-related events, a node may define timers to invoke certain functions periodically through the spin-thread mechanisms.

4.2 FMI-Adapter for ROS

The new fmi_adapter is a ROS package implemented in C++ for wrapping co-simulation FMUs according to the FMI standard 2.0 into ROS nodes. The package documentation is provided at `wiki.ros.org/fmi_adapter` and the source code can be downloaded at `github.com/boschresearch/fmi_adapter/`. An early version for ROS 2 can be found at `github.com/boschresearch/fmi_adapter_ros2/`.

The fmi_adapter package aims at providing the most important functions of the FMI 2.0 Co-Simulation interface (Modelica Association Project "FMI", 2014) mapped to ROS concepts/types as depicted in the following table:

FMI	ROS
input variable	subscription
output variable	publisher
state variable	*no explicit counterpart*
parameter initialization	parameter server
simulation time	ROS clock - offset
communication step-size	timer

It is intended neither to implement the whole FMI 2.0 interface nor to provide the rich set of introspection functions as for example FMI Library (JModelica.org, 2012) or FMI4cpp (SFI Offshore Mechatronics Research Centre, 2018). For advanced use-cases, ROS developers are referred to such libraries. Internally, fmi_adapter is based on the FMI Library, but the specific types are hidden from the developer.

The fmi_adapter package can be used, both, as a standalone ROS node and as a library.

Node use. The fmi_adapter package provides a ROS node, which takes the file path of an FMU as parameter `fmu_path` and creates subscribers and publishers with message type `std_msgs::Float64` for the input and output variables of the FMU, respectively. Next, it queries the ROS parameter server with the names of all variables and parameters of the FMU. For each name being found, the corresponding variable or rather parameter is initialized with the value being retrieved from the parameter server. Finally, the initialization mode of the FMU is exited and the node runs/simulates the FMU with a user-definable update period according to the ROS clock – until the node is shutdown. The following line gives an advanced example for invoking this node:

```
rosrun fmi_adapter node \
  _fmu_path:=./TransportDelay.fmu \
_step_size:=0.001 _d:=0.5 \
__name:=nodeB \
/nodeB/u:=/nodeA/angle
```

In this example, the FMU implements a simple transport delay with real-valued input u, output y and delay parameter d.[1] In this invocation, the step-size is set to 1 ms, the delay is set to 0.5 s, the node's name is set to `nodeB` and the input x is connected to the topic `/nodeA/angle`. Hence, the values of `/nodeA/angle` will be published on `/nodeB/y` with a delay of 0.5 s, sampled at 1 kHz.

Library use. The fmi_adapter package also provides a shared library which gives much more control about the integration of an FMU in a ROS node. Most important, it allows to decompose complex ROS message types and to map the individual fields to the primitive-typed input variables of an FMU. Also, it enables the use of multiple FMUs inside a ROS node. Finally, it provides some basic functions to introspect a given FMU, e.g., to query the variable names depending on their causality.

For this purpose, the fmi_adapter library provides a C++ class `fmi_adapter::FMIAdapter`, which wraps a single FMU whose file path is passed as constructor argument (cf. Figure 11). On such an instance the default experiment step-size can be queried (by `getDefaultExperimentStep`), the names of the input variables, output variables, and parameters can be retrieved (by `getInputVariableNames`, etc.), and initial values can be set (by `setInitialValue` and `initializeFromROSParameters`).

The end of the initialization phase is marked using `exitInitializationMode`. Now, inputs can be set programmatically per variable using `setInputValue` and the FMU simulation can be advanced with two functions `doStep` and `doStepsUntil`. Output values can be retrieved with `getOutputValue`. For input values, the `FMIAdapter` class allows to pass timestamped values and thus even to specify a trajectory, where the user can decide whether the input values are interpolated linearly or considered as a step function. This feature facilitates to translate between different sampling/sensor rates.

Implementation details. Several subtle details had to be considered in the mapping between FMI and ROS concepts. The most important is the representation of time. ROS represents time in seconds and nanoseconds since the epoch whereas FMI uses a floating-point-based representation. The latter loses precision for large values. Also, an FMU may specify a specific start time, typically zero. Therefore, `exitInitializationMode` expects a ROS timestamp. This timestamp is used as offset between the ROS time and FMU time in all future function calls.

Another subtle difference is that FMI supports various characters in the variable names that are not allowed in parameter or topic names in ROS. We introduced a function `rosifyName` to replace these characters by underscores.

[1] ROS does not support physical unit specifications but assumes that all values are defined in the International System of Units (SI).

Finally, the FMU has to include a binary for Linux. The export of such FMUs is supported by various commercial and open-source modeling tools.

4.3 Integration of the Path Filter FMU

Drive commands are represented as `TwistStamped` messages (from the `geometry_msgs` package) in the ROS-based navigation architecture of the AS DevKit. The `twist.linear.x` field represents the lateral velocity and the `twist.angular.z` field the rotational velocity. This is a very common representation in ROS for velocity commands for differential wheeled robots.

In the navigation stack of the AS DevKit these messages are sent from the path tracker node to the engine driver node on a topic named `/velDes`. The former node implements a controller to follow the given path from the global path planner; the latter node translates the lateral and rotational velocity to motor commands for the left drive motor and right drive motor.

We integrated the Path Filter FMU in a new node named `PathFilter` between those two nodes using the fmi_adapter library. This new node consists of one function `main` only, with just 25 lines of code. On receiving a `TwistStamped` message on the topic `/velDes`, it feeds the Path Filter FMU with the values of `twist.linear.x` and `twist.angular.z` using `setInputValue` and runs the FMU up to the current time. Then, it reads the resulting output values from the FMU, creates a new `TwistStamped` message and publishes it on a new topic `/velDesFil`.

To integrate the `PathFilter` node in the existing architecture, only two lines in the corresponding ROS launch file had to be changed: A new line for the `PathFilter` node had to be added and the input topic for the engine driver node had to be changed to `/velDesFil`.

5 Application and Test

In this section, we describe the application of the ROS architecture with the newly integrated `PathFilter` node to an AS DevKit. In a series of field tests it was the motivation to gather data that supports the results of the MIL simulations in Section 3.2 and further verifies that the path filter reaches its objectives (cf. section 3).

5.1 Test Setup

In contrast to the adaption of the launch file that was described in Section 4.3, we operated the Active Shuttle DevKit in manual mode. Here, the `PathTracker` node is deactivated and the desired motion is published to the `/velDes` topic by a node that is interfaced with a joystick. The desired motion profile included the following segments:

- straight forward → stop → CW turn on the spot → stop → straight forward → stop → straight backwards → stop → straight forward

The profile was driven several times with different path filter parameter variations including one reference case with-

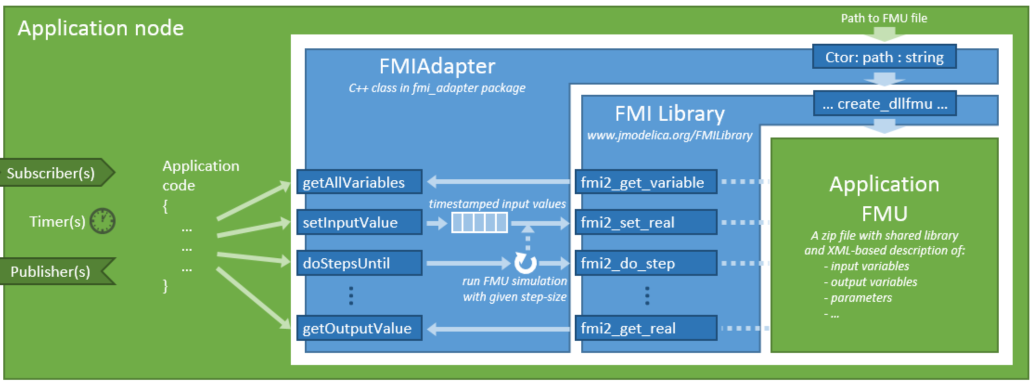


Figure 11. Architecture diagram from fmi_adapter package, illustrating library use

out path filter. The AS DevKit was loaded with 150 kg which sums up to an overall weight of $m_{AS} \approx 200$ kg. The relevant topics `/velDes` , `/velDesFil` and `/engine-data` were recorded with the `rosbag` tool. The latter topic holds the messages with the measured motor speeds and currents. The measured AS state $\boldsymbol{x}_{AS,m}$ results from $n_{LD,m}$, $n_{RD,m}$ and the AS kinematics. In order to verify that the path filter is solving the operational issue of randomly flipping caster wheels, $\boldsymbol{x}_{AS,m}$ is examined in combination with video recordings.

5.2 Test Results

Figure 12 shows a detailed view on the CW turn on the spot. Here, the measured motor currents of a reference case with no path filter are compared with a filtered case.

The time span where the motor currents of the reference case have reached a peak value of $\pm \approx 18$ A can be interpreted as the moment when the caster wheels are abruptly changing into their desired states $\boldsymbol{x}_{C,i,des}$. Here, the counteracting bore torque is abruptly vanishing and a significant drop in the motor currents can be observed. This drop results in an oscillation of the motor currents which lasts for a couple of seconds until it is damped.

The intended effect of the path filter can be nicely seen between $t = 15.4$ s and $t = 16$ s. The motor currents necessary to induce the turn on the spot are considerably reduced compared to the reference case. The path filter forces the AS to continue driving in the direction of the current caster wheel orientations first. Hence, the motor currents are first rising with a positive slope before the right motor current is dropping below zero. This accelerates the caster wheel speeds before the turn is initiated and reduces the bore friction. Moreover, we observe that the drop in the motor currents and the resulting oscillation are significantly mitigated.

Figure 13 shows a detailed view of the change from driving backward to forward. As it was elaborated in Section 3 the path filter intends to avoid the random caster wheel flip. We observe that the path filter forces the AS to keep driving backwards for a split of a second before it consciously induces an angular velocity component which triggers the orientation change of the caster wheels. Even

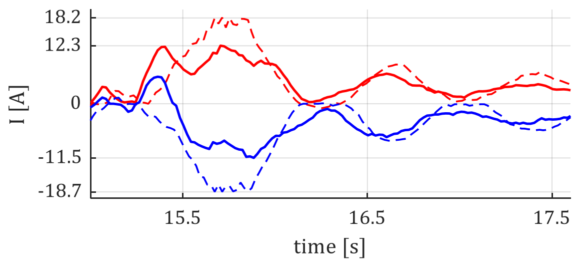


Figure 12. Measured motor currents for initiating a CW turn on the spot. The AS ($m_{AS} \approx 200$ kg) was driving forward prior to that. Dashed lines represent the results w/o path filter, solid lines the results with $\omega_{max} = 100$ rad/s and $T_\varphi = T_\omega = 0.01$ s. Red and blue lines represent the left and right drive, resp.

though the actual caster wheel orientations can not be measured we assume that the strategy works out. $\boldsymbol{x}_{AS,m}$ is following $\boldsymbol{x}_{AS,fil}$ and $\boldsymbol{x}_{AS,m}$ is significantly different from zero between $t = 46$ s and $t = 49$ s. Our assumption was verified through video recordings which clearly show that the caster wheels are turning instantaneously after the motion is started.

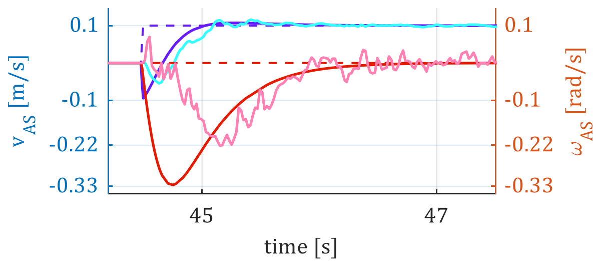


Figure 13. Desired, filtered and measured AS state for changing the driving direction. The AS was driving backwards prior to that. Dashed lines represent the desired motion, solid lines the filtered and measured motion. Path filter parameters and AS mass as in figure 12.

6 Conclusions and Outlook

In this paper a new model for wheels with bore friction has been presented. This model is suitable to describe the impact of bore stiction and allows to replicate critical locking conditions of differential drive vehicles with caster wheels. Based on the developed wheel model a vehicle model of the AS DevKit has been developed and used to design and validate the so-called *Path Filter*.

Using the new *FMI-Adapter* the PathFilter has been successfully integrated into a ROS control architecture and deployed to the AS DevKit. This way it has been demonstrated that FMI is a viable and attractive approach for an integrated end-to-end workflow from model-based control design to software integration for robotic applications and service oriented architectures. The open source *fmi_adapter* package enables users to wrap an FMU into a ROS node without deeper understanding of the FMI internals. The development process is drastically reduced with respect to time, effort and complexity.

The *Path Filter* has a significant positive impact on the handling qualities of the AS. It significantly reduces the effort necessary to perform movements that require the caster wheels to be turned on the spot. Moreover, it damps oscillations in the motor currents caused by the abrupt release of the counteracting bore torque. The path filter reduces the jerk which increases the durability of hardware components, eases the handling of fragile goods and improves the stability of shaky loads. Most important, the risk of getting stuck in a lock condition is drastically reduced. The Path Filter has been designed as self-contained function that can be retrofitted into existing control architecture between motion planer and motion controller.

The demonstrated application uses a micro processor as target and the deployed software was not subject to certified development processes. In the future work it needs to be ensured that the code within an FMU satisfies requirements of safety critical software and is optimized for real time applications. The current efforts within the publicly funded European project EMPHYSIS (ITEA3, 2017), developing the FMI for embedded systems (eFMI) standard, is addressing these challenges.

The Path Filter could be improved with respect to calibration effort and ressource demand by reducing the number of tuning parameters and the number of states. The estimation of the caster wheel angle using two first order holds could be replaced by an estimator based on the measured velocities of the driven wheels.

The plant model of the SDV could be enhanced to consider the impact of the currently neglected inertial forces.

Further studies shall reveal which constraints could be incorprated by the motion planner to determine trajectories that are compliant with the orientation of the caster wheels, which would allow to dispense the Path Filter.

References

G. Bardaro, L. Bascetta, F. Casella, and M. Matteucci. Using Modelica for advanced Multi-Body modelling in 3D graphical robotic simulators. In *Proc. of the 12th Int'l Modelica Conference*, Prague, Czech Republic, May 2017.

H. Durrant-Whyte and T. Bailey. Simultaneous Localization and Mapping (SLAM): Part I/II. *IEEE Robotics Automation Magazine*, 13(2/3):99–110/108–117, Jun/Aug 2006.

S. Imlauer, C. Mühlbacher, G. Steinbauer, S. Gspandl, and M. Reip. Hierarchical Planning with Traffic Zones for a Team of Industrial Transport Robots. In *Proc. of 4th Workshop on Distributed and Multi-Agent Planning (DMAP)*, pages 57–65, London, UK, Jun 2016.

ITEA3. EMPHYSIS – Embedded systems with physical models in the production code software, 2017. Retrieved 13 Nov 2018 from `itea3.org/project/emphysis.html`.

JModelica.org. FMI Library, 2012. Retrieved 3 Jul 2018 from `jmodelica.org`.

Modelica Association Project "FMI". Functional Mock-up Interface for Model Exchange and Co-Simulation – Version 2.0, Jul 2014.

F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov. A Loosely-Coupled Approach for Multi-Robot Coordination, Motion Planning and Control. In *Proc. of 28th ICAPS*, Delft, The Netherlands, Jun 2018.

M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *Proc. of ICRA Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

G. Rill. *Simulation von Kraftfahrzeugen*. Vieweg Verlag, Regensburg, Germany, 2007.

SFI Offshore Mechatronics Research Centre. FMI4cpp, 2018. Retrieved 25 Oct 2018 from `github.com/SFI-Mechatronics/FMI4cpp/`.

S. Swaminathan. Modelica-ROS Bridge. Retrieved 14 Jan 2019 from `github.com/ModROS`.

The MathWorks. Robot Operating System (ROS) Support from Robotics System Toolbox. Retrieved 23 Oct 2018 from `www.mathworks.com`.

M. Thümmel, G. Looye, M. Kurze, M. Otter, and J. Bals. Nonlinear Inverse Models for Control. In G. Schmitz, editor, *Proc. of the 4th Int'l Modelica Conference*, pages 267–279, Hamburg, Germany, March 2005.

S. Traversaro, P. Ramadoss, and L. Tricerri. gazebo-fmi. Retrieved 14 Jan 2019 from `github.com/robotology/gazebo-fmi`.

D. Zimmer. A free Modelica library for planar mechanical multibody systems, 2014. Retrieved 23 Oct 2018 from `github.com/dzimmer/PlanarMechanics`.

D. Zimmer and M. Otter. *Real-time models for wheels and tyres in an object-oriented modelling framework.* Vehicle System Dynamics, 2010.

Systematic Simulation of Fault Behavior by Analysis of Vehicle Dynamics

Artem Kolesnikov[1] Dzmitry Tretsiak[1] Morgan Cameron[2]
[1]ESI ITI GmbH, Dresden, Germany
[2]ESI Group, Rungis, France,
{Artem.Kolesnikov, Dzmitry.Tretsiak, Morgan.Cameron}@esi-group.com

Abstract

A new library for *System Reliability Analysis* for systematic modelling of fault effects in multi-physical systems is introduced. The motivation is outlined as well as a description of the library structure with two helper libraries and an Add-In for semi-automatic fault augmentation. The library is exemplified in the automotive domain with the fault effect simulation of a vehicle model by using a new library for *Driving Maneuvers*. The vehicle model is systematically augmented with connector and component faults for the analysis of different fault effects in vehicle dynamics.

Keywords: Fault, Reliability, Model-based Diagnosis, Driving Maneuvers, Vehicle Dynamics

1 Introduction

The decreasing development time and the rising competition in the automotive industry have led to the increased application of model-based approaches to system engineering. The applied models usually describe the designed ("nominal") behavior of a technical system and require a large amount of time and effort to be adopted because of system deviations. The reasons the physical system can deviate from its nominal behavior are manifold. First of all, the manufacturing of products is only exact to a certain point - there are variations and sometimes systematic deviations from the specified system design. Secondly, during the use of a product, its behavior can change because materials and material pairings are subject to wear, aging, abrupt failures etc. These effects include loss of lubrication, corrosion and hardening of materials. This has necessitated the development of specific models for simulation of fault effects.

Usually, models that represent fault effects are developed to describe the specific non-nominal behavior of a technical system and, hence, often require a large amount of effort to be reused for other failures or requirements. Moreover, the system simulation of multi-domain systems such as a vehicle is usually based on their nominal library components of object-oriented modelling languages. Most of them, such as Modelica®-based libraries, have seen limited attempts to extend models for model-based failure analysis. These include the *Fault Triggering* library, which allows for the insertion of faults into models of existing components (van der Linden, 2014); and the *Fault-Augmented Model Extension (FAME)* library with predefined fault augmentation (de Kleer *et al*, 2013). The latter was described in detail in (Saha *et al*, 2014) and has served as a basis for the systematic fault-augmentation approach presented in this paper.

In the following section the library developed for *System Reliability Analysis* (*SRA*) is described, whose components are the basis for the systematic fault augmentation of the vehicle model. The *SRA* library consists of a basic package containing elementary type and class definitions to parametrize faults, as described in (Gundermann *et al*, 2018). Beyond that, it is structured according to the structure for composable models in the Modelica® language (Minhas *et al*, 2014).

To demonstrate the capabilities of the *SRA* library, a vehicle model was created based on the components of the library *Driving Maneuvers* described in the section 3. As a sample, the preconfigured standard driving test defined by ISO “Severe double lane change” was chosen as described in (ISO 3888-1, 1999). Helper libraries supporting feature extraction and checking of requirements fulfilment are described in section 4. The augmentation of relevant faults in the investigated model and sample analysis of fault effects are shown in section 5.

The systematic fault augmentation methodology provides new capabilities to analyze fault behavior in multi-physical systems in *SimulationX* (www.simulationX.com), by exploiting the multi-domain SRA library and accompanying semi-automatic fault-insertion functionality. The paper motivates and demonstrates these capabilities in the context of model-based failure analyses in automotive applications. Specifically, the vehicle model was augmented with connector, component and signal fault structures of the

SRA library to simulate very different fault effects extracted and analyzed by the helper libraries.

2 The Library for System Reliability Analysis

Throughout this paper the term fault refers to any deviation from the nominal system's behavior. The Modelica® components developed for the *SRA* library in *SimulationX* simulate only the effect of a process on the system's behavior and focus on its dynamical evolution or interaction with other processes. For example, mechanical faults in joints, gears, shafts, springs or clutches because of breakage or slipping lead to the reduction of transmitted forces or torques. The current reduction because of bad or open connections in an electrical system, the mass flow decreases in a hydraulic system because of leakage or obstruction as well as changes of heat flows in a thermic system have comparable behavior and can be similarly structured, modeled and analyzed (Kolesnikov *et al*, 2018). As an example, the three oscillators from different physical domains shown in Figure 1 are similarly structured and augmented with faults in *SimulationX*. This follows from the analogies between the basic electrical, mechanical and hydraulic network elements.

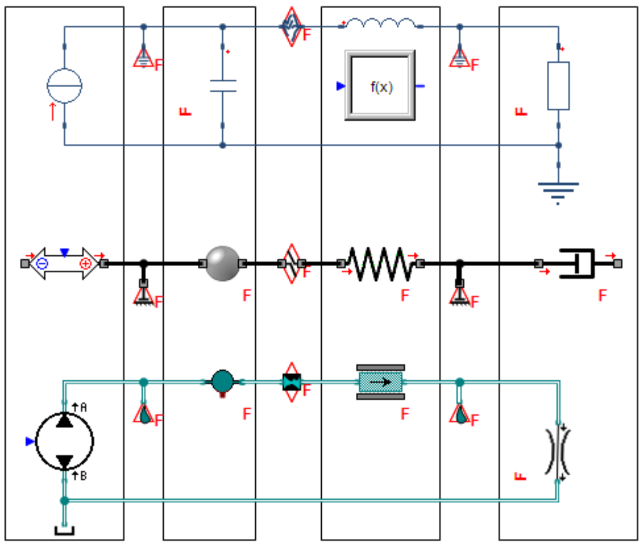


Figure 1. A model of oscillators in SimulationX with electrical, mechanical and hydraulic network elements after the augmentation with connector faults (red connectors) and parametric faults (components with the symbol F).

2.1 Fault Types

Based on the processes and effects outlined above the *SRA* library of *SimulationX* contains appropriate type definitions for two defined types of faults: *continuous* and *discrete*. Both these fault types are specialized classes – *records* containing an editable variable which is dynamic and can change its value during simulation (Fritzson, 2014). By using a type definition for the faults, it is easy to systematically read out and control the faults in a model.

Continuous faults parameterize the strength of a gradual effect. Typical examples are wear and aging phenomena which can lead to a gradually different value of a backlash parameter, or friction coefficient, etc. In the continuous record *fault* this gradual change is translated into a normalized variable named *intensity*, ranging between zero (nominal) and one (maximal effect). If the fault modifies a parameter or variable, the extended continuous record *FaultWithFunction* can be used, which allows for the definition of the functional dependency of parameter change, e.g. multiplicative

$$p = p_0(1 + c_0(scale \cdot intensity)^n) \qquad (1)$$

or exponential

$$p = p_0 e^{c_0(scale \cdot intensity)^n} \qquad (2)$$

Herein,

p_0 represents the nominal parameter value, c_0 , n represent parameters of the function. The scale parameter is needed to define a typical magnitude of the fault effect, which depends on the specific application, e.g. the mass of the surrounding components, acting forces, etc.

The *discrete* fault type can be only active or not, i.e. the fault is switched on or off. Typical examples are abrupt phenomena, such as failure of an electronic component, or breaking of a mechanical connection. The discrete record *fault* contains the *Boolean active*, which is false in the nominal case, and true if the fault is active.

If the fault modifies a parameter or variable, the extended discrete record *FaultWithFunction* can be used, which allows for the definition of the functional dependency of parameter change, e.g. proportional to the nominal parameter value including setting the prefactor d_0

$$p = p_0 \cdot d_0 \cdot scale \qquad (3)$$

or as an alternative value including setting the value p_1

$$p = p_1 \cdot scale \qquad (4)$$

2.2 Fault Classes

As shown in Figure 1 a model consists of components potentially taken from different libraries connected together. Fault modelling leads to changes in the behavior of components (*ComponentFaults*), alters the transport of quantities between components (*ConnectorFaults*) or adds new connections (*BridgeFaults*). A snapshot of a model which was augmented with *ComponentFaults*, *ConnectorFaults* and *BridgeFaults* from the SRA library, is shown in Figure 2. One of the resistors is replaced by its fault-augmented counterpart. A *ConnectorFault* modeling a short-to-ground has been added to the connection between the inductor and the capacitor. The connection between the two resistors has been cut by a *ConnectorFault* modeling a loose contact. An additional switchable connection (*BridgeFault*) models a short circuit between two junctions in circuits of consuming components.

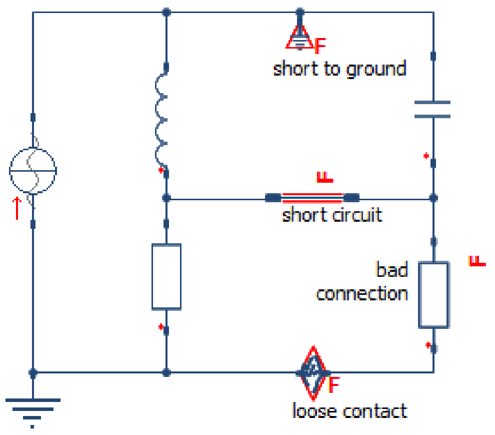


Figure 2. A model of an electrical circuit in SimulationX which has been augmented with faults.

ConnectorFaults can be inserted at (connected) connectors or into existing connections. The first type has two connectors, as shown at the bottom of Figure 2, and can be used to model, for example, the loose contact or the breaking of a mechanical component, or a mechanical obstruction. The second type has only one connection (in Figure 2, at the top), it is added to a connection, and can be used to model, for example, a contact-to-ground fault or a leakage in hydraulics. The models of these faults consist of a fault and a set of equations which depend on the intensity/active(-ity) of this fault, and, in general, on a scale.

BridgeFaults are elements with two connectors as shown in the middle of Figure 2. For each domain there exists one *BridgeFault*. They can be inserted between existing components, adding further connect-statements, if the fault is active. *BridgeFaults* can be used to model, for example, an interaction between two leaky hydraulic circuits.

Fault-augmented models with parametric *ComponentFaults* are fault-augmented counterparts to the nominal *SimulationX* library elements. As an example, Figure 2 shows a fault-augmented element named ComponentFaults.Electricity.Analog.Basic.Resistor, which is an extension of the Electricity.Analog.Basic.Resistor. It contains a fault, which changes the resistance parameter from its default value 1, as defined by intensity, scale, and function in the fault-record. The Fault triggering library models faults in a similar way to those contained within *ComponentFaults* (van der Linden, 2014).

2.3 The Fault Scale

When modeling with the connector or bridge faults, but also during the development of the fault-augmented counterparts of the library elements, the amplitude of the strongest effect (intensity=1, or active=true) differs depending on the surroundings of the system into which the fault is inserted. For example, the frictional torque in a rotational connection which prevents the latter from moving is several orders of magnitude higher in a ship's propeller than in a clockwork mechanism.

In the *SRA* library this phenomenon is captured with the definition of a positive real parameter named *scale* which can be included and defined in the definition of a fault if needed. It should be emphasized that the meaning of the *scale* differs between different faults. Furthermore, *scales* of the same faults in a model can differ in different places e.g. if the model contains both the ship's propeller and the clockwork mechanism.

For a meaningful effect of the inserted faults, it is crucial to find ways to estimate the *scale* for each fault. In the optimal case, one can calculate the *scale* from the amplitudes of variables in the surroundings of the fault as recorded during a single simulation of the nominal behavior. For example, the *scale* of the translational sticking fault is a prefactor to the friction force F_{stick} added to the connection. The latter is defined as $F_{stick} = scale$ for *intensity* 1.

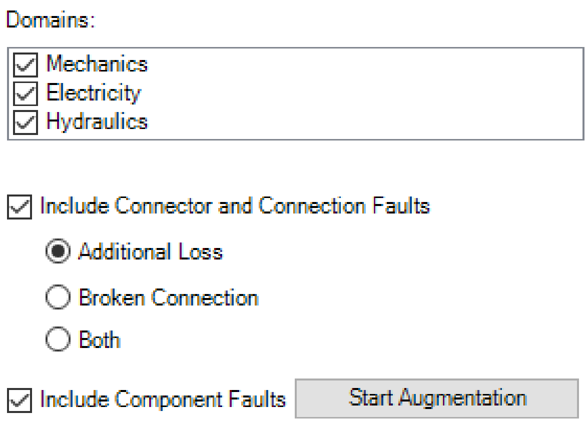
The *scale* should be chosen to be high enough to ensure that the connection stops moving when the fault intensity is equal to one. On the other hand, it should be low enough that one is not faced with numerical issues, because the system is stiff or the solver has trouble integrating when the fault is switched on during a simulation. In the optimal case, the scale value should be set to ensure that the range of the friction force over increasing intensity is such that for low intensities it already has some (recognizable) effect on the connection, but does not already prevent it from motion at all (sensitivity to fault intensity). Estimating the *scale* can be challenging even in the seemingly simple case of the friction force described above. To this end, several algorithms are currently under investigation using base unit, force, energy or power values.

2.4 Semi-Automatic Fault Augmentation

Augmenting a model of the nominal system with various faults can be time-consuming and error-prone. Moreover, when replacing the nominal type of a component by its fault-augmented counterpart, one must ensure that the parameters are kept or modified correctly.

To support the fault-augmentation process, a dedicated user interface to *SimulationX* has been developed. The tabbed interface guides the user through the whole augmentation and analysis process in an intuitive way. Once the augmentation is completed, faults of interest can be selected for subsequent analysis. The augmentation and analysis workflow can be summarized by the steps described below.

Definition of the candidate: First of all, the user must define the candidate components, i.e. all those components and connections in the model, which are to be fault-augmented. Additionally, it can be decided whether to exclude or include a certain class of faults (connector or component faults), or whether to insert faults only from specific domains (mechanical, electrical, etc.) as shown in Figure 3.


Domains:

- [✓] Mechanics
- [✓] Electricity
- [✓] Hydraulics

[✓] Include Connector and Connection Faults

[✓ Additional Loss] Broken Connection Both

[✓] Include Component Faults

Figure 3. Selection of fault classes and domains supported by the SRA Add-In.

Augmentation: The actual fault-augmentation can be executed by clicking the “Start augmentation” button. The process changes the model structure as shown in the right-hand side of Figure 4. “Nominal” components are replaced by their fault-augmented counterparts. In this example, a fault with two pins (broken) is added for each connected connector was added, and another fault with one pin (sticking) was added for the whole connection.

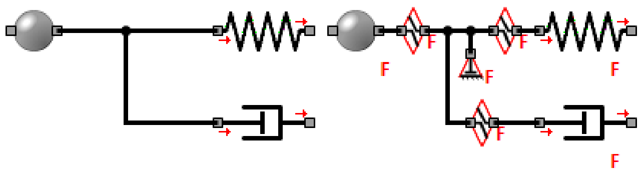


Figure 4. SimulationX model before (left) and after (right) the automatic fault augmentation for mechanical components.

3 Driving Maneuvers Library for Analysis of Vehicle Dynamics

A vehicle model for the fault effect simulation was built using a new library for *Driving Maneuvers* in *SimulationX*. The modular library contains various chassis model elements, wheel and axle suspensions, wheel elements with tire model, driver models, track and environment components and complete vehicle models. The library elements are parameterized and validated using real experimental data (Tretsiak *et al*, 2018).

The library structure has an open, user-extensible, object-oriented model architecture thanks to its compatibility with the Modelica® modelling language. The library structure, shown in Figure 5, includes following subsections described in detail below:

- Environment, Drivers and Maneuvers
- Bodies and Wheels
- Suspensions and Axles with Suspensions
- Vehicles

The real-time capable curve-based vehicle models can be used in combination with elements such as powertrains or brakes from other libraries of *SimulationX* (*Mechanics*, *Power Transmission (1D/MBS)*), and with their fault-augmented counterparts from the *SRA* library.

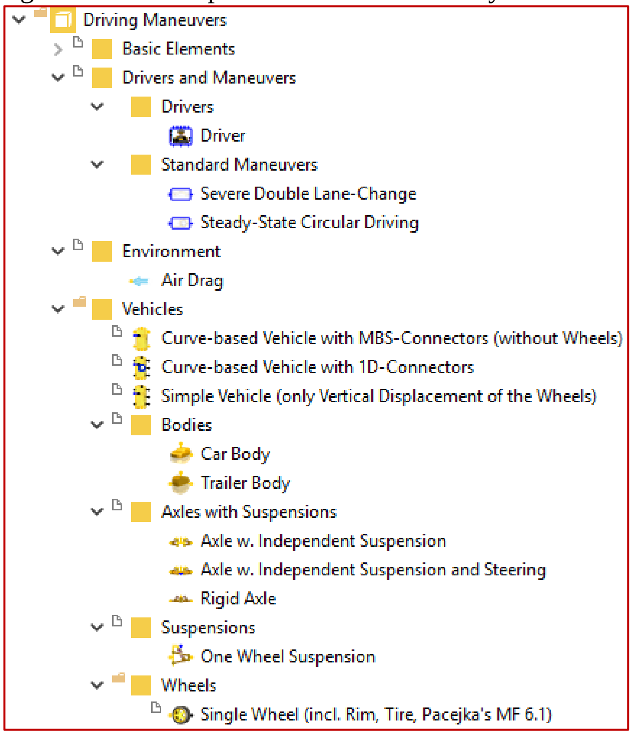

- Driving Maneuvers
  - Basic Elements
  - Drivers and Maneuvers
    - Drivers
      - Driver
    - Standard Maneuvers
      - Severe Double Lane-Change
      - Steady-State Circular Driving
  - Environment
    - Air Drag
  - Vehicles
    - Curve-based Vehicle with MBS-Connectors (without Wheels)
    - Curve-based Vehicle with 1D-Connectors
    - Simple Vehicle (only Vertical Displacement of the Wheels)
    - Bodies
      - Car Body
      - Trailer Body
    - Axles with Suspensions
      - Axle w. Independent Suspension
      - Axle w. Independent Suspension and Steering
      - Rigid Axle
    - Suspensions
      - One Wheel Suspension
    - Wheels
      - Single Wheel (incl. Rim, Tire, Pacejka's MF 6.1)

Figure 5. Structure of *Driving Maneuvers* library in SimulationX

3.1 Environment, Drivers and Maneuvers

The sub-library *Driver* contains the two-level driver model with anticipatory (open-loop) and compensatory (closed-loop) control for vehicle longitudinal and lateral dynamics, considering a curvature difference, steering angle difference and lateral displacement (Figure 6), which is generally based on Donges’ model (Donges, 1978).

Driver models can be configured individually, with the option of externally-defined speed and steering profiles. The output signals are steering, load and brake demands (steer, gas, brake). Maneuver element types allow setting of preconfigured standard driving tests defined by International Standard Organization, for instance:

- (Severe) Double Lane Change (ISO 3888-1, 1999; ISO 3888-2, 2002)
- Steady-State Circular Driving (ISO 4138, 2012)

The driving track and designation of maneuver sections are depicted in Figure 7. There is the possibility to set customer-specific test tracks.

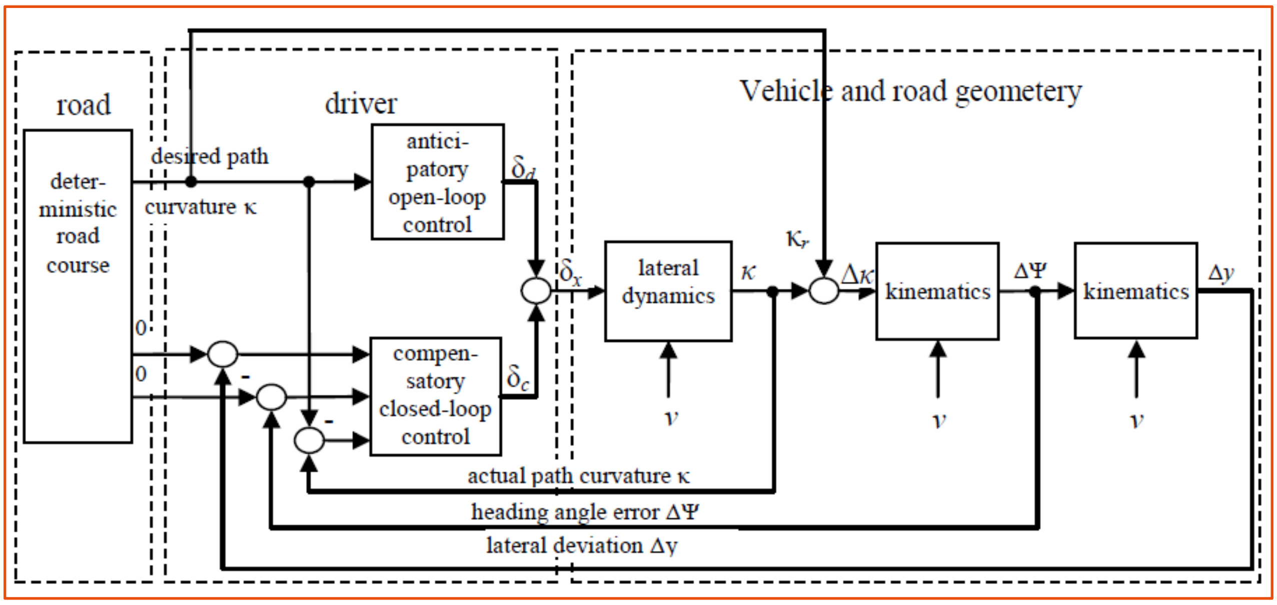
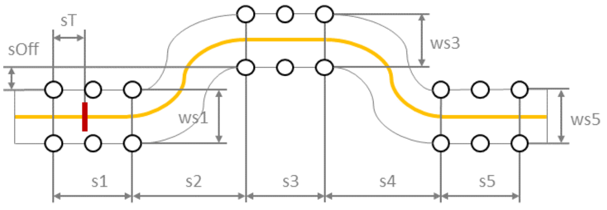


Figure 6. A two-level model of driver steering behavior

a



b

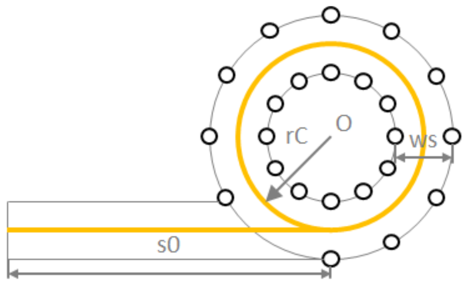


Figure 7. Maneuver tracks and designation of sections: a – *(Severe) Double Lane Change*; b – *Steady-State Circular Driving*

The model element *Air Drag* provides air drag forces and torques in all directions and gives the possibility to define corresponding air drag coefficients:

- Only Cw-coefficient
- All air drag coefficients (cx,cy, ...) scaled on the Cw-value
- All air drag coefficients
  - Air drag force coefficients
  - Air drag torque coefficients

3.2 Bodies and Wheels

Vehicle bodies can be visualized by default shapes or user-defined imported CAD geometry with corresponding scaling in all directions.

By default, three car types are available for visualization of a vehicle body:

- Compact car
- Station wagon
- Sport utility vehicle

A single CAD geometry is used for the representation, with corresponding scaling coefficients for each vehicle type. The car body model element can be used to represent, for instance, a truck cabin or a bus saloon, requiring only corresponding CAD data. The trailer body component has the same functionality as the car body. Figure 8 presents the 3D model visualization with the above-described library elements.

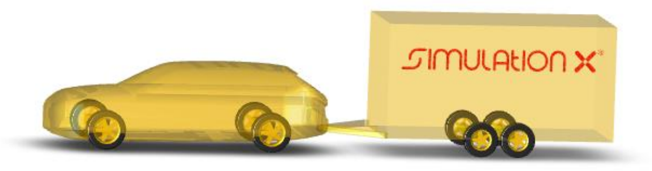


Figure 8. Visualization of *Car Body* and *Trailer Body* model elements

The tire is modelled according to Pacejka's MF 6.1 (Bakker *et al*, 1987) and is used in a single wheel element, which includes a corresponding visualization body.

3.3 Suspensions and Axles with Suspensions

These parts contain predefined curve-based models of wheel and axle suspensions with anti-roll bars.

Suspension kinematics are defined by zero-order kinematics parameters (e.g. value) or wheel lift trajectories (e.g. camber angle over vertical displacement of a wheel center).

Different steering types are available:

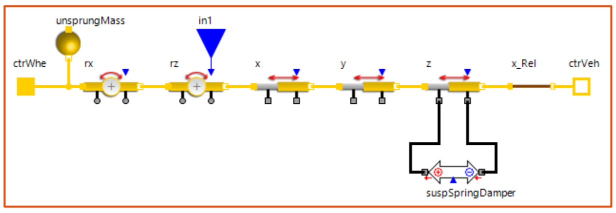
- With constant or variable steering gear ratio

- With direct input of wheel rotation angles over a steering wheel angle

Five degrees of freedom of one wheel are constrained in the case of an independent suspension.

Figure 9 shows the model structure of a one-wheel suspension and an axle with independent suspension and steering.

a



b

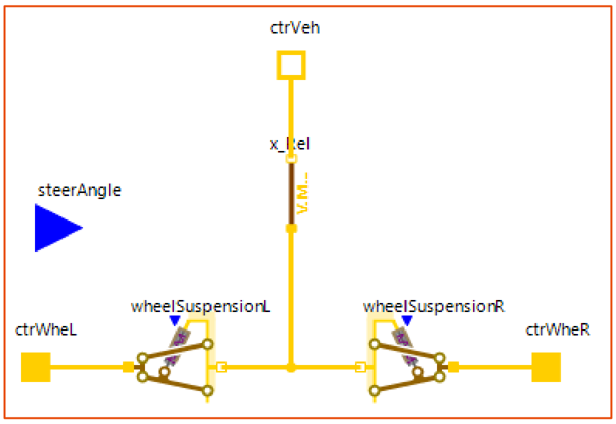


Figure 9. Structures of suspension models:
a – one-wheel suspension; b – axle with independent suspension and steering.

The rigid axles library allows the modelling of specific axles with suspensions also for heavy-duty trucks and buses:

- Rigid axle with conventional suspension
- Complete rigid axle
- Swing axle (without suspension)

A mirror function for parameterization of symmetric suspensions facilitates this routine process.

3.4 Vehicles

The real-time capable curve-based vehicle models with 1D and MBS connectors are based on the following library elements:

- Car body
- Axle with independent suspension
- Axle with independent suspension and steering
- Air drag and motion sensor

Up to six load masses for passengers and luggage can be defined.

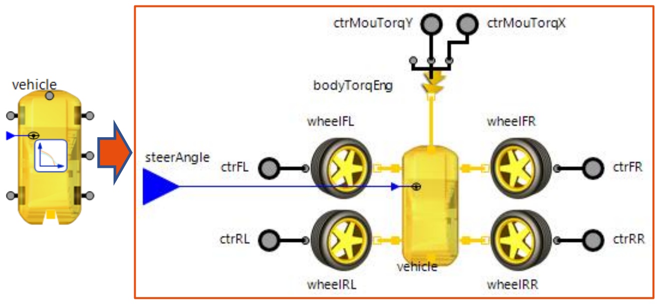
Engine and powertrain mount torques can be considered, depending on the engine position (longitudinal or transverse).

There is a signal connector for steering input and different variants are available to set wheel parameters:

- 4 equal wheels
- 2 equal front wheels, 2 equal rear wheels
- Free definition (individual settings)

The corresponding diagram views of characteristic curve-based vehicle models with 1D and MBS connectors are presented in Figure 10.

a



b

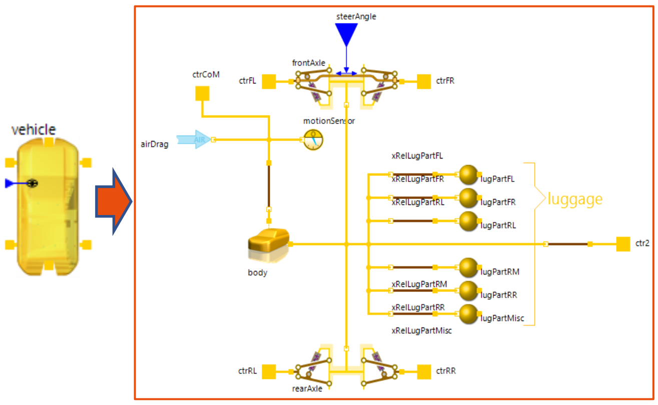


Figure 10. Structures of curve-based vehicle models:
a – curve-based vehicle with 1D-connectors; b – curve-based vehicle with MBS-connectors (without wheels).

4 Analyzing Faults

As well as structuring the modeling and insertion of faults, it is also helpful to structure and systematize all output quantities of the model that constitute related information about fault effects. Moreover, the checking of requirements fulfillment in the model-based applications must consider not only pre-defined criteria, but also scenarios executed for them. The additional libraries for special signal analysis in *SimulationX* presented in this section serve as helpers for this task.

4.1 Signal Feature Extraction

For model validation, the model output is compared to real systems' data. Such data is, in general, sampled, can sometimes be averaged, noisy, or even in frequency-domain representation. To prepare the model output in the same way, one needs ways to extract features from the variables.

To avoid dealing with (different) sampling rates and the quality of time series, or to use single-valued data from output time series, meaningful features must be extracted. Furthermore, requirements fulfillment is categorized by blocks returning a single value: successful, failed, or undecided/not clear. The determination of these categories (for more details see subsection 4.2) is based on calculations of output variables, which are again features of the latter.

The library for signal feature extraction contains helper blocks that support feature extraction. Helper blocks shown in Figure 11 are provided for extracting features such as time interval between two pulses or two events and band check. The list is extended based on the examples studied. One important requirement is that all features are insensitive to numerical side effects. For example, the extraction of the time between two pulses should not pick a "numerical" peak, the height of which is dependent on tolerance or other numerical artefacts. The extraction of the time between two mean values should be possible over restricted time spans to avoid a dependency on the overall simulation time (e.g. the average velocity of the vehicle decreases to zero, because the model driving scenario depicts an unnecessary amount of time span after the vehicle has come to rest).

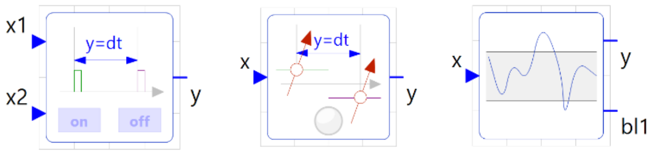


Figure 11. Time Interval bw. 2 pulses or 2 events and band check elements

The additional sub-packages ChecksInFixedWindow, ChecksInSlidingWindow, SignalAnalysis from (Otter et al., 2015) - although motivated by a completely different application - serve similar purposes.

4.2 Requirements Fulfillment Analysis

The sample vehicle model analyzed in the next section addresses the question of the requirements fulfillment when it is subject to faults. Stated differently: which fault or combination of faults leads to the violation of pre-defined criteria. An example of such a criterion, taken from the drive train example, is: The vehicle shall be able to brake from 100 km/h to stop in 7 Seconds. To assess this in the model, output variables (velocity, time) are read, features are extracted (velocity at 7 seconds after start of braking) and tested against the criterion “stopping”. The formalization of the last step is supported by the elements modeled in the library *Requirements Fulfillment.*

The basic definition of a criteria contains an array of assertions as an input. In the example of the velocity test this array has one entry: v($t_{StartBraking}$ +7)[km/h]. If this entry is lower than zero, the criterion is fulfilled, if not, it is violated. If the array of the assertions contains more than one entry, it has to be defined whether they are connected by an AND (i.e. all must be fulfilled to fulfill the whole criterion), an OR (i.e. only one must be fulfilled), or NOT (i.e. all must be violated to fulfill the whole criterion). Sometimes it does not make sense to test a criterion at all - for example, if there was no ignition, there is no startup time. To address this scenario, the requirements fulfillment elements contain a second input named *validity indicator* defined in a similar way as the *assertion* - i.e. if this variable is positive, the validity is given and the assertion can be tested, if not, the assertion does not need to be tested. The integer output *requirements fulfillment* is based on the validity indicator and the assertions, and is restricted to three values, which represent the categories as listed in Table 1.

Table 1. Categories of the output of the *Requirements Fulfillment* elements based on the incoming validity and insertion.

requirements fulfillment	*category*	*conditions*
1	fulfilled	validity>0, assertion>0
0	violated	validity>0, assertion<0
-1	undecided	valididy<0

For convenience, the assertions are fed to an output variable assertionsOut. The output *requirements fulfillment* can only tell if the simulation fulfilled or violated the criterion, but not how far it was from violating or fulfilling it. To have a measure for this, the relevant continuous variables should be analyzed. Sometimes it is important to have information about how close to violating/fulfilling the specification, e.g. since due to noises/variations which are not in the model the outcome might not be robust. Furthermore, it proves helpful to have some information about whether an increasing fault intensity influences an assertion. This information cannot be obtained from the integer output.

Figure 12 contains elements of the *Requirements Fulfillment* library. Currently, the connection of assertions via AND, OR and NOT is possible, since any logical expression can be brought into either of these forms (disjunctive/conjunctive normal form, see e.g. (Hazewinkel, 1994)). More flexible definitions of *Requirements Fulfillment* components will be developed as applications demand.

The *Modelica_Requirements* library presented in (Otter et al., 2015) contains a similar 3-valued logic to those presented here. Its motivation comes from a connection between system simulation and the formal definition of requirements. For the *Modelica_Requirements* library an extension of the Modelica language is proposed to handle the 3-valued temporal logic (satisfied, violated, undecided). The formalization of proving the logical expressions built up from validity and assertions (as described above) could be improved if the handling of the 3-valued logic becomes part of the language standard. However, it is not necessary in our case.

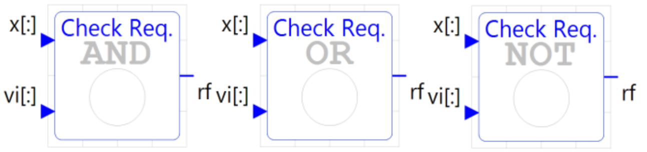


Figure 12. Checking requirement criteria – ALL, ONE or No Requirements must be met.

5 Simulation and Analysis of Vehicle Dynamics with Fault Effects

To demonstrate the application of the *SRA* library in *SimulationX* for considering fault effects by the analysis of vehicle dynamics, the model for a preconfigured standard driving test double lane change is used. It is a well-known test method which is generally used for subjective evaluation of vehicle dynamics (ISO 3888-1, 1999). The driver can use a vehicle brake system passing through maneuver track. Consequently, there is the possibility to simulate fault effects in the brake system and analyze its influence on a vehicle dynamics and road-holding ability.

5.1 Fault-Augmented Vehicle Model

The vehicle model is built using the model elements of *Driving Maneuvers* library, described in the section 3. Figure 13 shows the corresponding diagram view of the vehicle model augmented with fault elements from the SRA library and is described below.

The model structure includes the maneuver element described in the subsection 3.1, whose input signal is the driven route of the vehicle with a transverse engine. The simplified vehicle model includes the torque source for front drive wheels, whose value is defined by a corresponding curve, and the differential element. The corresponding mount torque of the car with transverse engine is also considered.

The output from the maneuver component is connected to the driver element and provides the closed-loop driver model with the information about the desired route curvature and desired vehicle speed. The instantaneous curvature and car speed for the driver model are received from the fault-augmented vehicle element. The output signals from the driver element are: steering wheel angle demand and load and brake requirements. Wheel brakes provide corresponding brake torques based on the brake signal from the driver model element.

For the fault augmentation by the SRA Add-In described in the section 2.4, the elements of the brake system and the vehicle were selected as candidate components, as shown in Figure 14. The brake system was augmented with ConnectorFaults (sticking 1-4) to analyze the following failures:

- Drop of a brake pad friction coefficient, reducing the braking torque acting on a wheel. It can be caused by uneven wear of brake pads or their surface contamination with oil film, dirt, etc.
- Drop of an actuation force of brake drive, reducing the braking torque acting on a wheel because of a drop of a hydraulic pressure caused by pipe rupture or leakage in a brake drive.
- Jamming the wheel brake, decreasing vehicle stability and control because of brake lining wear or wear of sealing ring of a wheel hydraulic cylinder.

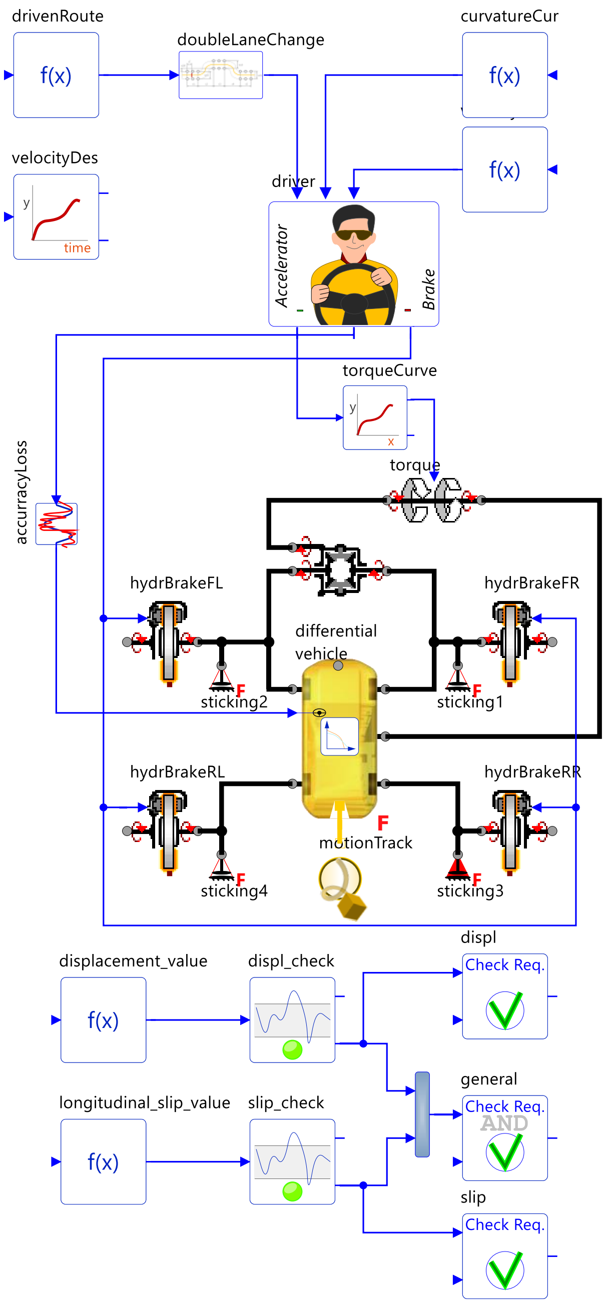
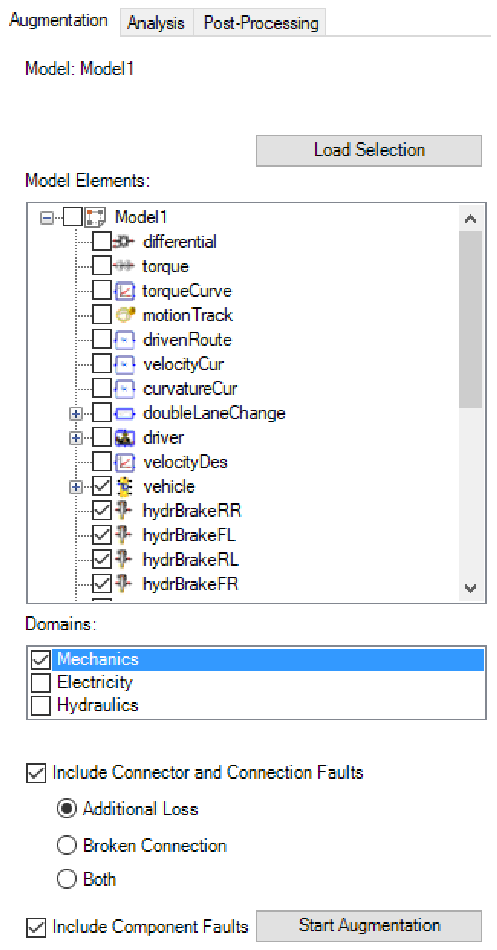


Figure 13. Vehicle model with augmented connector, signal and component faults

The vehicle component was replaced with its fault-augmented counterpart (*ComponentFault*) to investigate the friction change in the contact between wheel and road because of:

- Change of wheel pressure
- Tire tread wear


Augmentation | Analysis | Post-Processing

Model: Model1

Load Selection

Model Elements:

- [ ] Model1
  - [ ] differential
  - [ ] torque
  - [ ] torqueCurve
  - [ ] motionTrack
  - [ ] drivenRoute
  - [ ] velocityCur
  - [ ] curvatureCur
  - [ ] doubleLaneChange
  - [ ] driver
  - [ ] velocityDes
  - [✓] vehicle
  - [✓] hydrBrakeRR
  - [✓] hydrBrakeFL
  - [✓] hydrBrakeRL
  - [✓] hydrBrakeFR

Domains:

- [✓] Mechanics
- [ ] Electricity
- [ ] Hydraulics

[✓] Include Connector and Connection Faults

- [✓ Additional Loss]
- Broken Connection
- Both

[✓] Include Component Faults

Start Augmentation

Figure 14. Selection of model components supported by the SRA Add-In.

The component *accuracyLoss* from the *SRA*-library was placed in the model between the *driver* and *vehicle* components to consider signal fault (control signal distortion) and to investigate its influence on the vehicle's behavior during the standard driving test with steering and braking failures.

5.2 Sample Analysis of Vehicle Dynamics with Fault Effects

As was mentioned above, the double-lane change maneuver is used for subjective determination of a vehicle obstacle performance, which is a part of vehicle dynamics and road-holding ability. If a vehicle is in the track frames during the test, then it is assumed that it has a sufficient obstacle performance and road-holding ability. The model allows visual analysis based on a vehicle trajectory through the maneuver track (Figure 15). However, this kind of analysis inefficient for assessment of multitude fault combinations and cannot provide sufficient information for the post-processing.

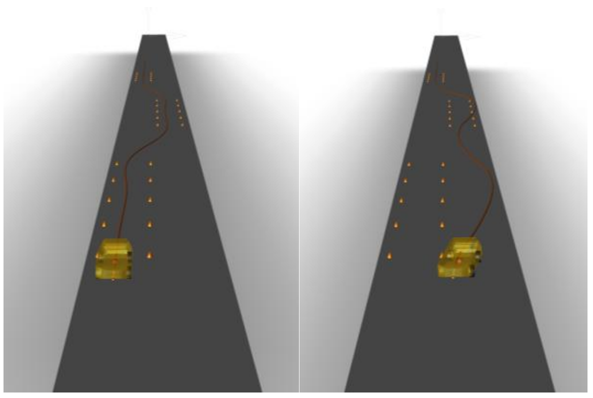


Figure 15. Vehicle trajectories by successful (left) and unsuccessful (right) passing through maneuver track

To systematically analyse fault effects in the brake system, components from the helper library *Signal Feature Extraction* were used as shown in the lower-part of Figure 13. These components (*displ_check* and *slip_check* in the model) check information about passing of a vehicle through maneuver based on the displacement of a vehicle in XY-plane and the longitudinal slip of vehicle front right wheel (Figure 16).

Figure 16 shows the deviation of the selected output values of the fault-augmented model from its nominal behaviour (without fault) because of the activation of the connector faults. The connector faults (components *sticking 1-3* in Figure 13) decreases the braking torque in the right rear wheel (fault in RR) or front left wheel (fault in FL) up to 50% or jam the front right wheel brake (fault in FR).

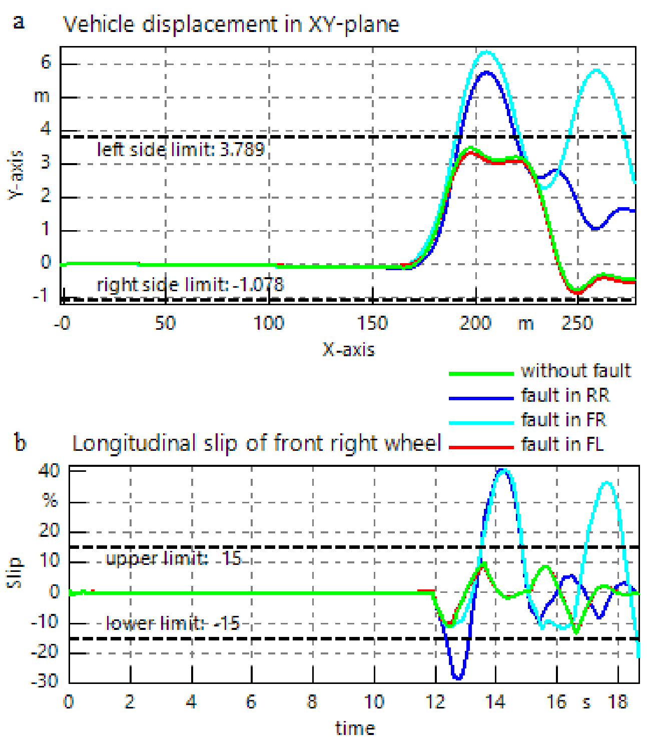
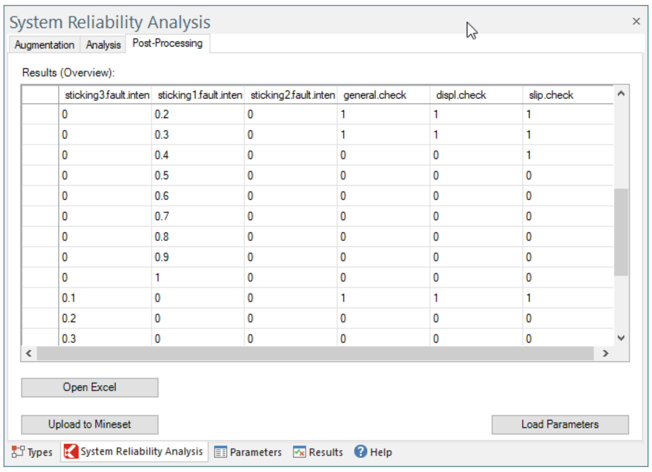


Figure 16. Selected output values of the fault-augmented model with different faults in the brake system

The limits depicted in Figure 16 define the parameters of the components *displ_check* and *slip_check* that serve to automatically extract the important features for the *Requirements Fulfillment* components (*displ*, *slip* and *general*) in the vehicle model. In the “Post-processing” tab of the SRA Add-In output quantities in these components provide a means to analyse requirements fulfillment under different faults. Figure 17 shows the results of the fault effect analysis with different fault intensities (between 0 and 1) and checked requirements (successful – 1, unsuccessful - 0). The analysis results can be saved in CSV format or uploaded to the further investigation in the data analytics tool ESI MINESET (Mineset).



	sticking3.fault.inten	sticking1.fault.inten	sticking2.fault.inten	general.check	displ.check	slip.check
	0	0.2	0	1	1	1
	0	0.3	0	1	1	1
	0	0.4	0	0	0	1
	0	0.5	0	0	0	0
	0	0.6	0	0	0	0
	0	0.7	0	0	0	0
	0	0.8	0	0	0	0
	0	0.9	0	0	0	0
	0	1	0	0	0	0
	0.1	0	0	1	1	1
	0.2	0	0	0	0	0
	0.3	0	0	0	0	0

Figure 17. Results of requirement fulfillment analysis for different fault combinations.

Conclusion

In this publication, we introduced the *System Reliability Analysis* library, which enables the user to model and simulate physical systems outside their nominal behavior in a systematic way. We motivated the utility of the *System Reliability Analysis* library with a vehicle model from the *Driving Maneuvers* library having different failures.

Based on the model, the broad range of applicability of the fault effect analysis in multi-physics domains was outlined. Illustrative simulation results - based on the given sample model - were presented and show the possibility of systematic simulation of fault behavior by analysis of vehicle dynamics using new *System Reliability Analysis* and *Driving Maneuvers* libraries.

In addition, the motivation for and implementation of helper libraries (*Signal Feature Extraction*, *Requirements Fulfillment*) and tools (*SRA* Add-In) was described.

References

Edmund Donges. A two-level model of driver steering behaviour. *Human factors*, 20(6), 1978, pp. 691-707.

E. Bakker, L. Nyborg, and H. B. Pacejka. Tyre Modelling for Use in Vehicle Dynamics Studies. *Society of Automotive Engineers*, January 1987.

P. Fritzson. Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach,” Nov 2014, Wiley-IEEE Press, ISBN: 978-1-118-85912-4.

J. Gundermann, A. Kolesnikov, M. Cameron, and T. Blochwitz. The Fault library - A new Modelica library allows for the systematic simulation of non-nominal system behavior. *Proceedings of the 2nd Japanese Modelica Conference*, Japan, Tokyo, May 17-18, 2018, Industrial Paper, pages 161-168, 2018, doi: 10.3384/ecp18148161.

Michiel Hazewinkel. Encyclopaedia of Mathematics (set). Encyclopaedia of Mathematics. Springer Netherlands, 1994. ISBN 9781556080104. URL https://books.google.de/books?id=uxUBQwAACAAJ.

International Organization for Standardization, 1999. ISO 3888-1 Passenger cars — Test track for a severe lane-change manoeuvre — Part 1: Double lane-change. Geneve: ISO.

International Organization for Standardization, 2002. ISO 3888-2 Passenger cars — Test track for a severe lane-change manoeuvre — Part 2: Obstacle avoidance. Geneve: ISO.

International Organization for Standardization, 2012. ISO 4138 Passenger cars - Steady-state circular driving behaviour - Open-llop test methods. Geneve: ISO.

J. de Kleer, B. Janssen, D. G. Bobrow, T. Kurtoglu, et al. Fault augmented modelica models. *24th International Workshop on Priciples of Diagnosis*, Jerusalem, Israel, pages 71-78, 2013.

A. Kolesnikov, M. Andreev, and A. Abel. The Fault-Augmented Approach for the Systematic Simulation of Fault Behavior in Multi-Domain Systems in Aerospace. *SAE Technical Paper* 2018-01-1917, 2018, doi: 10.4271/2018-01-1917.

F. L. J. van der Linden. General fault triggering architecture to trigger model faults in modelica using a standardized blockset. *10th International Modelica conference*, number 96 in Linköping Electronic Conference Proceedings, LiU Electronic Press, pages 427-436, 2014, URL http://elib.dlr.de/90576/

R. Minhas, J. de Kleer, I. Matei, B. Saha, at al. Using fault augmented modelica models for diagnostics. *Proceedings of the 10th International Modelica Conference*, March 10-12; 2014; Lund; Sweden, number 96, Linköping University Electronic Press; Linköpings universitet, pages 437-445, 2014.

Martin Otter, Nguyen Thuy, Daniel Bouskela, Lena Buffoni, Hilding Elmqvist, Peter Fritzson, Alfredo Garro, Audrey Jardin, Hans Olsson, Maxime Payelleville, et al. Formal requirements modeling for simulation-based verification. In *Proceedings of the 11th International Modelica Conference*, Versailles, France, September 21-23, 2015, number 118, pages 625–635. Linköping University Electronic Press, 2015.

Mineset: https://cloud.esi-group.com/analytics

B. Saha, T. Honda, I. Matei, E. Saund, at al. A model-based approach for an optimal maintenance strategy. *Second European conference of the prognostics and health management society*, pages 521-531, 2014.

D. Tretsiak, T. Wiedemann, C. Bellanger, and F. Kocksch. Modeling of vehicles with varying level of detail for system simulation - Development of a modular chassis model kit including a consistent parameterization process. *Proc. of ESI SimulationX Conference*, Dresden, Germany, 2018.

SESSION 4C: AEROSPACE

Modeling and Simulation of Dual Redundant Electro-Hydrostatic Actuation System with Special Focus on Model Architecting and Multidisciplinary Effects

Duansen Shangguan[1] Liping Chen[1] Jianwan Ding[1] Yuhui Liu[1]

[1] School of Mechanical Science and Engineering, Huazhong University of Science and Technology, China, {ahcq1990,chenlp,dingjw, yuhuiliu}@hust.edu.cn

Abstract

Electro-hydrostatic actuator (EHA) is a new trend in the more electric aircraft related research works and engineering applications. As a high-performance mechatronics product, however, the physical effects of actuator behavior are multidisciplinary, coupled and strongly nonlinear. Although many commercialized multi-domain and system-level simulation packages exist, they are rarely considered and analyzed as a whole, lacking of a unified model architecture, efficient modeling forms, and comprehensive simulation verification. In this paper, Modelica is used to build a multi-domain virtual prototype of the dual redundant electro-hydrostatic actuation system (DREHAS) that consists of two EHAs in parallel, which supports multi-view modeling and interdisciplinary application of the system. Finally, a simulation application case of the elevator actuation system is presented to demonstrate the effective role of Modelica models in system modeling and evaluation.

Keywords: more electric aircraft, dual redundant electro-hydrostatic actuator, working mode, system model, Modelica

1 Introduction

In the last decade, the power-by-wire (PBW) actuators became sufficiently mature to be applied in the more electric aircraft (Cao *et al*, 2012; Rosero *et al*, 2007). As the carrier of PBW, the EHA a hydraulic actuator driven by a dedicated pump, rather than a hydraulic network, which drives the pump to control the actuating components by adjusting the motor speed. Compared to conventional hydraulic actuators, the EHA is a typical mechatronic system with the advantages of high-power density, low load and easy modularization, which emphasizes the integration and synergy in specific domains such as mechanics, electronics, controls and hydraulics (Charles, 2017; Li, 2007). But the EHA also presents more challenges in some aspects. One of the non-negligible aspects is the physical effects of the EHA behavior are multidisciplinary, coupled and strongly nonlinear. For example, the causes of force-fighting phenomenon inherent in the DREHAS are more complicated, possibly due to voltage spikes, current transients, pressure pulses, electromagnetic interference, electromagnetic interference and mechanical losses.

Traditionally, the modeling and simulation methods for the DREHAS mainly include the theoretical modeling method based on transfer function and the co-simulation method with commercialized multi-domain simulation software. The former modeling method is to simplify and linearize some models of the system (Waheed, 2015). In this way, there is a certain gap between the processed model and the actual system, which cannot fully reflect some characteristics and working conditions of the actual system. The latter modeling method can make the model more detailed and accurate, using a variety of commercialized multi-domain and system-level simulation packages. For example, the performance analysis in normal and failure modes can be achieved by co-simulation based on AMESim and Matlab (Ji et al, 2009). However, a single discipline-oriented multi-program combination often requires more effort to achieve the optimal results that a unified model system can achieve, which can lead to complexity in modeling and simulation. Moreover, the establishment of the EHA multidisciplinary unified system model is conducive to evaluating the design scheme and improving the quality of further analysis and decision making.

As is well known, Modelica implements a multi-domain unified statement description of mechatronics systems, integrating energy flow, mass flow and information flow, based on the generalized Kirchhoff law (Broenink, 1999; Fritzson, 2014). The core topic of this paper is the presentation of the multi-domain unified Modelica model for the DREHA. On the one hand, it is possible to provide an appropriate method for the close cooperation of different disciplines, and on the other hand to achieve the rapid and accurate evaluation of the system characteristics, especially the phenomenon of competing forces, during the conceptual and preliminary design phases.

The rest of the paper is structure as follows. The following section covers the introduction to the structure and characteristics of the DREHAS. Section 3 shows the

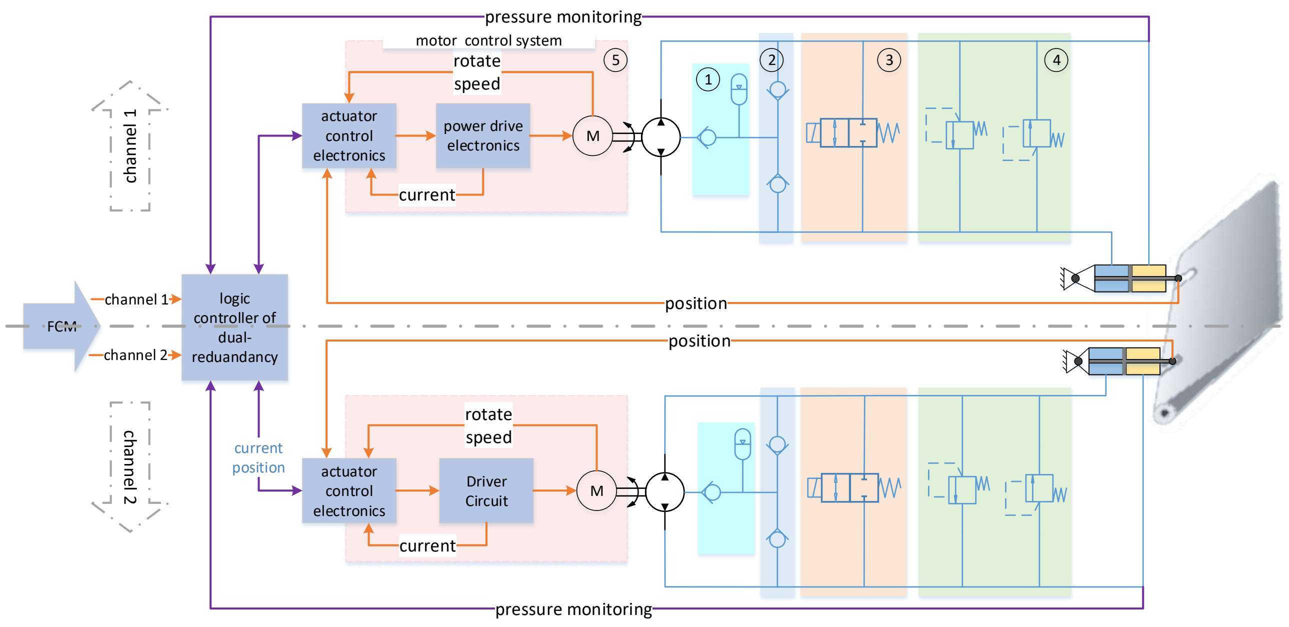


Figure 1 Schematic of a redundant electro-hydrostatic actuation system for elevator

model development process of the DREHAS. Section 4 showcases the simulation results and discussions. Finally, Section 5 gives the conclusions.

2 The principle of the DREHAS

The DREHAS involved in this study, as shown in Figure 1, is composed of the two identical EHA in parallel. The controller receives the FCM instruction signals and monitors all sensor signals in real time to realize the redundancy management and control of the system. Each EHA system is a position servo control, which controls the pump output flow of the pump by adjusting the motor speed to achieve a specific actuator output displacement and speed.

In addition, the safety-critical functions such as flight control used by the DREHAS must have a very low failure rate, which requires that each channel must have fail-safe devices to allow the remaining channels to operate correctly. In the DREHAS, the safety response to faults is easily achieved with hydraulic components at low quality and low cost. Figure 1 shows the functions implemented in the DREHAS, such as oil compensation function (part ①), protection against outgassing or cavitation (part ②), oil unloading function (part ③), bypass and safety pressure setting function (part ④).

2.1 Main components

In this paper, the DREHAS for elevator actuation system consists of the following components:

- controller, which performs the closed loop control of the EHA.
- permanent magnet synchronous Motor (PMSM), which drives the quantitative plunger pump to control the pump output flow by controlling the motor speed.
- plunger pump, which converts mechanical energy into hydraulic energy that drives the actuator.
- aircraft control surface, where a change in the angle of deflection causes a change in the hinge moment on the operating surface.

2.2 Working modes

In general, the DREHAS has three kinds of working modes: active/active (A/A) mode (both the EHAs are actively controlled), and active/passive (A/ P) mode (one EHA is actively controlled and one passive following). In addition, the last working mode refers to the fault damped (DP) mode for the two EHA failures. When the EHA is in the damping mode, the solenoid valve will break the connection between the actuator and the pump, and the oil passage is connected at both ends of the actuator. In this case, the actuation system is equivalent to a damper.

2.3 Control structure

The DREHAS follows the requirements of the pilot or autopilot to drive the elevator to deflect a specific angle and overcome the uncertain interference of the external aerodynamic load. The dual redundant logic controller obtains the working state parameters of the two-channel hydraulic cylinder according to the sensor detection. In the modules of channel 1 and channel 2, the working state of the system is judged: A/A mode, A/P mode, DP mode.

The surface position setpoint and the working mode from the dual redundancy logic controller are used as inputs to the EHA. As shown in part ⑤ of Figure 1, when the EHA is actively controlled, PMSM adopts a linear control method involving a PID serial corrector. A commonly used controller structure consists of a cascade of three nested loops: a current (internal) loop, a speed (middle) loop, and a position (external) loop. In addition, current feedback, speed feedback, position feedback, and output pressure of the actuator are used for real-time monitoring and control of the EHA.

2.4 Force-fighting phenomenon and multiple physical effects

In the active/active mode, the two EHAs that make up the DREHAS work together to push the rudder surface. However, the magnitude of the respective output forces in the physical actual state may not be completely uniform, and the rudder surface has a large rigidity, which causes a force-fighting phenomenon between the plurality of main actuators on the same rudder surface.

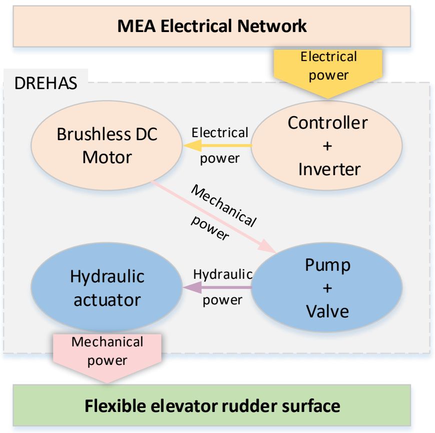


Figure 2 Energy transfer process among different physical domains in the DREHAS

In the traditional hydraulic double-residual actuation system, the force-fighting phenomenon is mainly caused by the accumulation of manufacturing and installation errors of sensors, actuators and rudder surfaces. However, the energy transfer during the operation of the DREHAS for the elevator is more complicated, as shown in Figure 2, and its control precision is more precise with 4 closed-loop controls. In this case, the factors causing force-fighting phenomenon are multidisciplinary, such as grid pollution, voltage spikes, current transients, electromagnetic interference, electromagnetic interference and mechanical losses.

2.5 Model structures for system modeling and evaluation

In terms of model modeling and simulation systems, choosing the right "good enough simulation model" means choosing a model with the corresponding granularity, which depends largely on the needs of the current engineering task. In many cases, a very precise system modeling is not a reasonable way to describe complex mechatronics, because even the uncertainty and cost of a relatively detailed model may be so high that its disadvantages become unaffordable compared to simpler modeling. So, the best model should be just enough to answer design questions, but not a more elaborate model.

This article focuses on issues related to system modeling and evaluation of the DREHAS, such as system function verification in different working modes, analysis of multiple physical phenomena (motor torque pulsation, current transients, pressure pulsation, rudder flutter), analysis and optimization of force-fighting phenomenon. This requires different granularity of the model of the DREHAS components. The models concerned mainly include: the motor and its controller, the plunger pump and the flexible rudder surface.

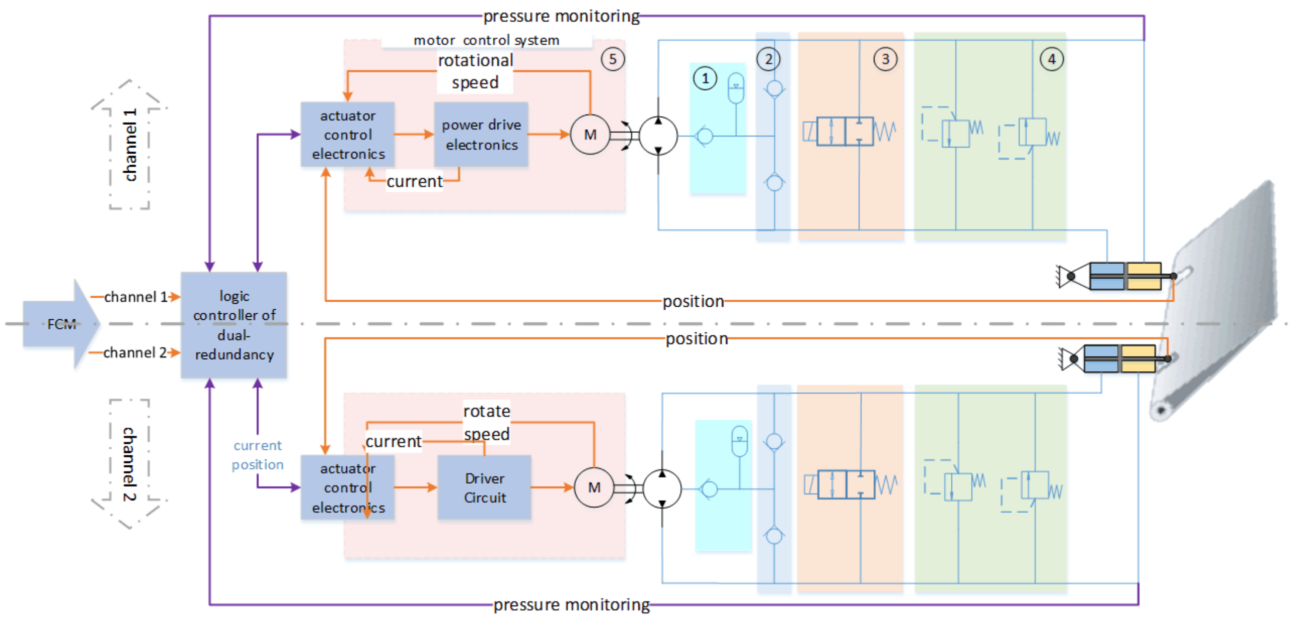
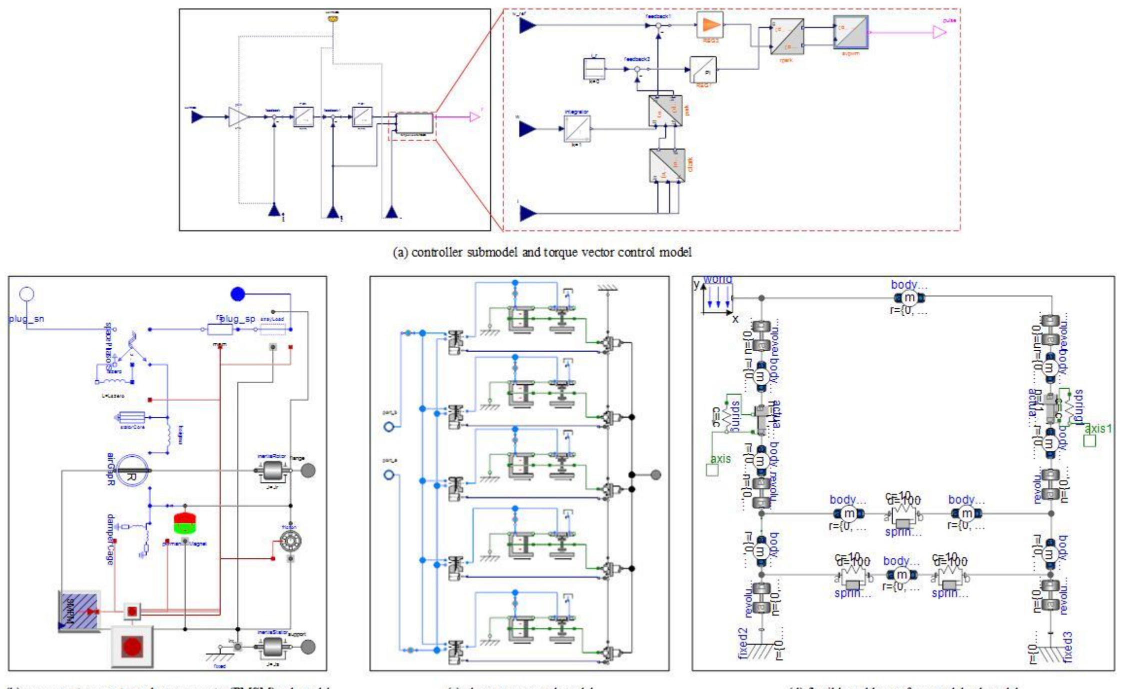


Figure 3 Realistic DREHA model in MWorks


(a) controller submodel and torque vector control model

(b) permanent magnet synchronous motor(PMSM) submodel

(c) plunger pump submodel

(d) flexible rudder surface model submodel

Figure 4 Five main submodels that make up the DREHAS in MWorks

3 Model implementation and Virtual prototype

The previous sections introduced the highly nonlinear DREHAS model architecture and multidisciplinary effects. In this section, a virtual prototype of the DREHAS for elevator control is established in the multi-domain simulation environment MWorks. MWorks provides management of models and integrative solvers, where Modelica standard library version 3.2.1 and older versions can be invoked (Chen et al, 2011).

Reusing existing models can greatly improve the efficiency of modeling and enable designers to focus more on design than on detailed model development or derivation of mathematical formulas. In this paper, based on the principle structure of Figure 1, the standard library model is called as much as possible to achieve a unified model that considers multidisciplinary effects. In the light of the design requirements in the preliminary design stage, the DREHAS model, Figure 3, is divided into five main sub-models (Figure 4): the controller model, the motor model, the plunger pump model and flexible surface.

3.1 Controller model

The DREHAS is composed of two independently controlled the EHA. The single-channel controller model is shown in Figure 1, which uses a classic three-closed loop control structure: the current (inner) loop, the speed (middle) loop and the position (outer) loop.

As shown in Figure 4(a), all the control loops adopt PID controller. The current loop is the innermost loop of the control law, allowing the motor current to quickly track a given current, thereby increasing system stiffness. Fast dynamic response and good tracking performance are required, but no static difference is required. The rotating speed loop is the intermediate loop of the control law, which makes the rotate speed fast track the given rotating speed, thus improving the dynamic performance of the system. The position loop is the outermost loop of the control law structure, which determines the dynamic and steady performance of the EHA, to ensure the system has fast dynamic performance and the steady state error is zero.

3.2 Motor model

The pump drive motor in the elevator actuation servo control system is a 270V high voltage PMSM with a rated power of 10KW and a maximum output speed of 10000r/min. The model SM_PermanentMagnet in the Modelica standard library can be directly reused. The model considers common loss effects: heat loss of armature winding resistance， brush losses in the armature circuit, friction losses, core losses, eddy current losses, and stray load losses.

The motor driver adopts three-phase full bridge circuit to convert dc voltage into a specific PWM waveform and drive the motor. In this paper, switching dynamics of the inverter are not accounted, and an average inverter is used.

Obviously, unlike traditional hydraulic servo actuators, there are multi-domain coupling effects and high-frequency loops in the EHA, such as motor torque ripple, current transients and energy losses, all of which are considered in the Modelica model shown in Figure 4(b).

3.3 Plunger pump model

Hydraulic component model can be implemented by using the Hydraulic component Design (HCD) library developed based on MWorks/Modelica. The HCD library contains 1-dimensional hydraulic components, such as pistons, spools, poppets, etc.

In this paper, the EHA uses a 5-cylinder quantitative axial plunger pump as shown in Figure 4(c). Referring to the basic structure, the plunger pump model is developed by using the HCD library and the model in the 1D Modelica.Mechanics.Translational library.

3.4 Flexible rudder surface model

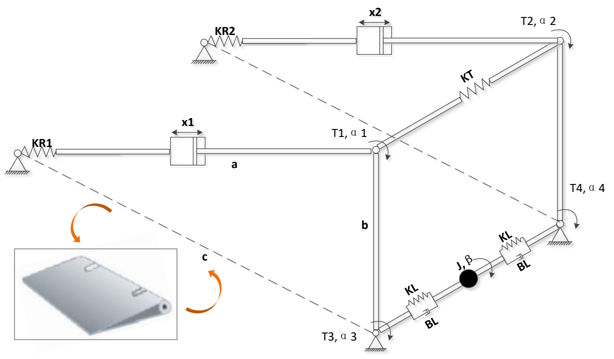


Figure 5 An elastic "structural beam" for the rudder surface

The rudder surface driven by the DREHAS cannot be considered as a simple rigid body due to the presence of two incompletely consistent driving forces. In the preliminary design stage, finite element analysis is not yet available. However, the rudder surface can be equivalent to an elastic "structural beam", as shown in Figure 5.

The relationship between the actuation displacements x_1, x_2 of the two parallel EHAs and the corresponding steering surface deflection angle β is as follows:

$$\begin{cases} J\ddot{\beta} = T_3 + T_4 \\ T_3 = F_1 * \dfrac{bc*\cos(\alpha_3)}{a + x_1 + F_1 / KR_1} + T \\ T_4 = F_2 * \dfrac{bc*\cos(\alpha_4)}{a + x_2 + F_2 / KR_2} - T \\ T = KT(\alpha_1 - \alpha_2) \end{cases} \qquad (1)$$

Where T_1 and T_2 is the equivalent upper beam torque, KL and BL are the equivalent lower beam stiffness and damping, KL_1 and KR_2 are the stiffness of the two EHA fixed joints, T_3 and T_4 are the hinge moments at the rudder surface fulcrum, α_3 and α_4 are the deflection angles at the fulcrum of the rudder surface.

It is obvious that the numerical solution of equation (1) is highly nonlinear. Solution environment MWorks provides the equation modeling language Modelica to describe the above equations and implement automatic analytical solution of the model. Moreover, mechanical components in the standard library can be reused to build an equivalent beam model of the rudder plane, as shown in Figure 4 (d).

4 Simulation results and discussions

Table 1 Simulation parameters of the DREHAS model

Parameters	*Values*
Motor supply voltage (V)	270
Maximum motor speed (r/min)	10000
Swashplate inclination (deg)	12.5
Armature winding resistance of motor (Ω)	0.245
Armature winding inductance (mH)	0.008
Motor-pump inertia (kg·m^2)	0.001
Displacement of pump (cc/rev)	1.5
Gas pre-charge pressure (bar)	3
Accumulator volume (L)	0.5
Equivalent beam stiffness (N/m)	2.7e^5
Equivalent beam damping (Nm/(rad/s))	80
Stiffness at the outboard fixed joints (N/m)	10e^7
Stiffness at the inboard fixed joints (N/m)	8e^7

In this section, based on the system model shown in Figure 3, the system characteristics of the DREHAS two main and fault operating modes (A/A mode, A/P mode，and DP mode) are simulated and analyzed. In addition, the force-fighting phenomenon caused by multidisciplinary effects in the A/A mode is simulated. The main parameters used in the following simulations are listed in Table 1.

Moreover, at 0.5s, a 0.02m step signal is given to the system as the input of displacement instruction. And at 2s, a loading of external pneumatic disturbance of 2000 N is applied to the flexible rudder surface.

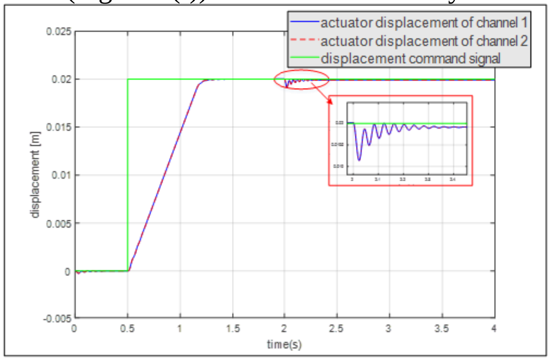
4.1 Active/active mode

In the A/A mode, the EHAs are in active control, and the two channels of the DREHAS work in parallel, driving their respective actuators to drive the rudder surface motion. The position control performance of the two EHAs are shown in Figure 6(a) indicates that the EHA almost reaches stability without overshoot at 1.5s, and the steady state error is less than 1%. When aerodynamic disturbance occurs, the steady state of the system is broken and then stabilized again under the action of the controller (consuming 0.4s). It can be seen that the controller can quickly restore the original position and has strong anti-interference ability. In addition, the EHA is a complex mechatronic product with multidisciplinary coupling. Different from the traditional hydraulic actuator, the actuator pistons move more rapidly, as shown in Figure 6(b). Part of the reason is due to the mechanical inertia of the mechanical system (traditional factors); the other part is due to the high frequency response of the motor and the plunger pump (multidisciplinary factors), as shown in Figure 6(c)&(d).

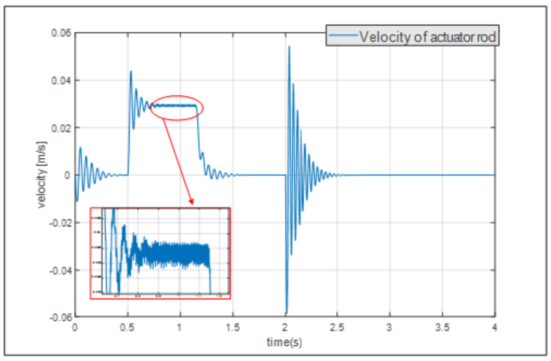
In the above simulation process, both EHA are in the ideal condition, without considering the actual multiple physical factors such as sensor error, electromagnetic interference, voltage pulsation, mechanical loss, and so on. In this case, there is no force fighting phenomenon between the two channels of DREHAS. In order to simulate the phenomenon of force fighting phenomenon, the model needs to be modified accordingly, where it is chosen to add a deviation at the motor drive current input. There is a large deviation between the two EHAs output forces, that is, there is the force fighting phenomenon (Figure 6(f)). Due to the flexibility of the rudder surface and the respective different driving forces, there is a certain deviation (Figure 6(e)) in the output displacement of the two EHAs.

4.2 Active/passive mode & fault mode

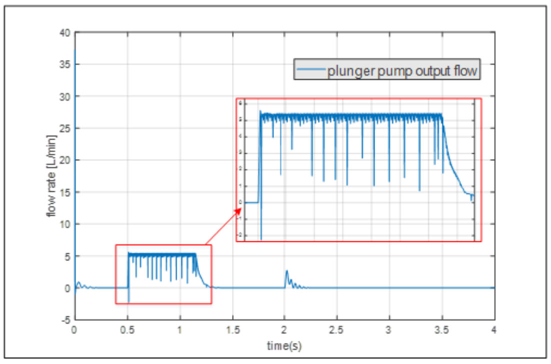
In the A/P mode, one channel of the EHA is actively controlled and the other passively follows. For EHA in passive mode, by-pass solenoid valve connects both ends of the actuator and the chamber will be filled with fluid under the pressure of the compensator. At this time, when there is an external load on the rudder surface, the hydraulic oil of the oil return system will flow between the two cavities through the throttle valve to act as a damping. Generally, the rudder surface can respond quickly under a given load gradient. Figure 7 shows that the system is stable within 0.8s, so the system performance can meet the requirements.



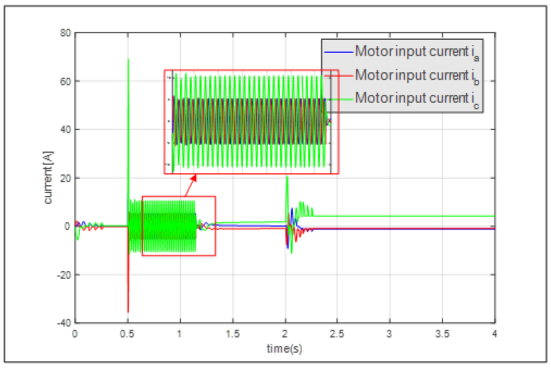
(a) Actuator displacement



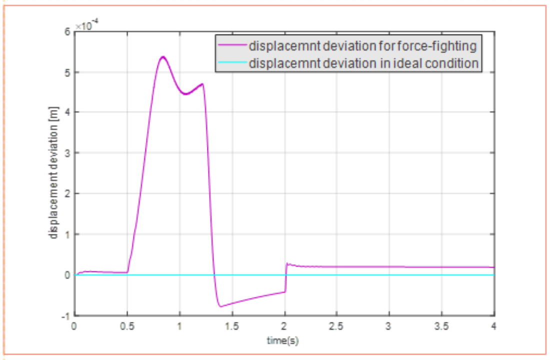
(b) Actuator rod velocity



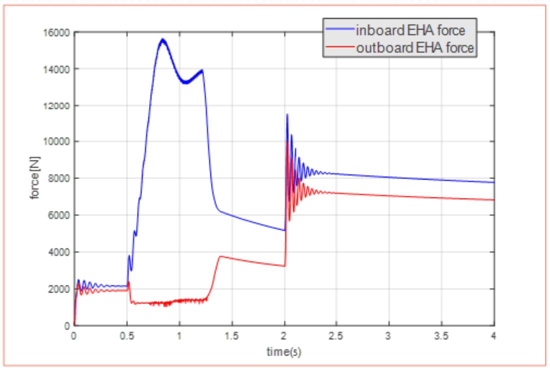
(c) plunger pump output flow



(d) PMSM drive current



(e) displacement deviation



(f) fighting force

Figure 6 DREHAS response in A/A mode

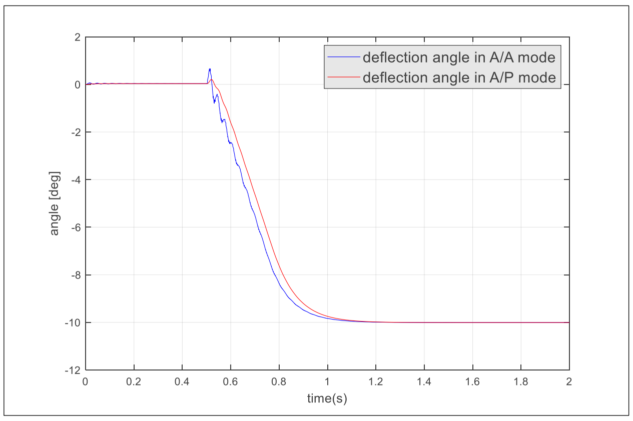


Figure 7 Deflection angle of the surface in A/P mode

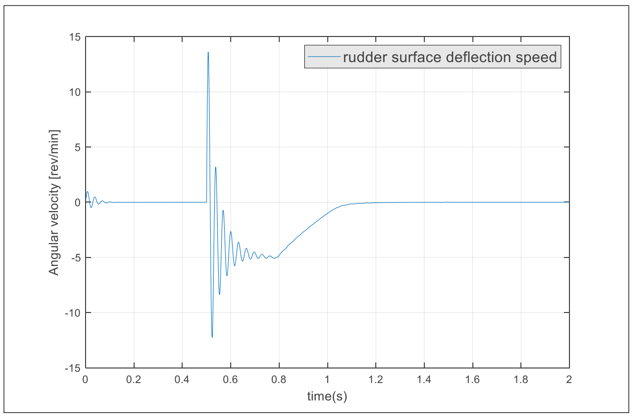


Figure 8 Angular velocity of surface deflection in the fault state

An important function of the DREHAS is to suppress the surface flutter. In normal operating mode, any EHA can use its own stiffness to suppress chatter. When the two EHAs are in a fault state, the DREHAS is in the damping mode and should be able to meet the requirements for suppressing chatter vibration through the damping circuit. Figure 8 shows the rudder surface can be stabilized rapidly under the action of external aerodynamic disturbance.

5 Conclusions

The DREHAS, a multidisciplinary coupled mechatronic product for more electric aircraft, is confronted with the multidisciplinary coupling problems in design. This paper presents a multi-domain unified Modelica model for multi-view modeling and interdisciplinary application in the preliminary design phase. For the multidisciplinary effects existing in DREHAS, a unified modeling language Modelica is used to establish the system model on the basis of reasonably planning the model hierarchy, defining the model interface and abstracting the model. The strong coupling and nonlinear problems are weakened to the greatest extent, which facilitates rapid simulation and verification between systems. The example of a simulation application in elevator actuation system has shown that the Modelica model not only supports the characteristic analysis of key components (motors, plunger pumps, etc.) and the overall performance evaluation of the system (system characteristics in three working modes), but also provides a suitable method for the close collaboration of experts or designers in different fields.

Acknowledgements

Thanks for the support of the members of the complex system design team in Huazhong University of Science & Technology CAD Center.

References

Jan F. Broenink. Object-oriented modeling with bond graphs and Modelica. *SIMULATION SERIES*, 31: 163-168, 1999.

Liping Chen, Yan Zhao, Fanli Zhou, et al. Modeling and Simulation of Gear Pumps based on Modelica/MWorks®. *The International Modelica Conference, Technical Univeristy, Dresden, Germany. pp*. 421-429, 2011.

Wenping Cao, Barrie Mecrow, Glynn Atkinson, et al. Overview of Electric Motor Technologies Used for More Electric Aircraft (MEA). I*EEE transactions on industrial electronics*, 59(9): 3523-3531, 2012. DOI: 10.1109/TIE.2011.2165453

Peter Fritzson. Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach[M]. *John Wiley & Sons*, pp. 565-589, 2014.

Youzhe Ji, Song Peng, Li Geng, et al. Pressure loop control of pump and valve combined EHA based on FFIM. In *Electronic Measurement & Instruments, 2009. ICEMI'09. 9th International Conference on*. IEEE, pp. 3-578, 2009. DOI: 10.1016/j.cja.2017.03.013.

Kai Li and Shaoping Wang. Multidisciplinary modeling method and simulation for Electro-Hydrostatic Actuator. *Industrial Electronics and Applications (ICIEA), 2010 the 5th IEEE Conference on*. IEEE, pp. 544-548, 2010. DOI: 10.1109/ICIEA.2010.5517088.

Jean-Charles Mare, Jian Fu. Review on signal-by-wire and power-by-wire actuation for more electric aircraft. *Chinese Journal of Aeronautics*, 30(3): 857-870, 2017.

J.A. Rosero, J.A. Ortega, E. Aldabas, et al. Moving towards a more electric aircraft. *IEEE Aerospace and Electronic Systems Magazine*, 22(3): 3-9, 2007. DOI: 10.1109/MAES.2007.340500

Ur Rehman Waheed, Shaoping Wang, Xingjian Wang, et al. A position synchronization control for HA/EHA system. In *Fluid Power and Mechatronics (FPM), 2015 International Conference on*. IEEE, pp. 473-482, 2015.

A Modelica-based environment for the simulation of hybrid-electric propulsion systems

Max J. Arzberger   Dirk Zimmer

Insitute of System Dynamics and Control,
DLR German Aerospace Center, Germany,
{max.arzberger, dirk.zimmer}@dlr.com

Abstract

In this paper, a framework for the modeling of hybrid electric propulsion system architectures for aviation is presented in form of a novel Modelica library. The scope and requirements for an aviation power train modelling framework are specified. The presentation then follows the hierarchical modelling structure of the library. Alongside, key modeling concepts are presented. Finally, the integrated analysis capabilities are highlighted and briefly demonstrated based on classic hybrid power train architecture.

Keywords: hybrid-electric power trains, electric flight

1 Introduction

In order to meet the challenging CO2 emission goals defined for aviation in Flightpath 2050 (compare European Commission, 2011), new disruptive aircraft developments are required to accelerate the progression of aviation technology. In addition to novel airframe and engine technologies, hybrid electric aircraft propulsions systems constitute a promising avenue for future developments by extending the design space for potential aircraft configurations (see Figure 1). This extension includes for instance electrical energy storages, or airframe integration concepts of the propulsion systems like boundary layer ingestion and distributed propulsion. Therefore similar to the automotive sector, a research trend for hybrid electric propulsion systems rose up in aviation over the last years (Hepperle, 2012). Besides the numerous studies published by NASA (SCEPTOR focusing on distributed propulsion, STARC-ABL focusing on boundary layer ingestion, SUGAR Volt), also leading companies in Europe collaborate in studies on hybrid electric aviation (compare Siemens Extra 330, Airbus E-Fan X).

In the context of hybrid electric propulsion systems in aviation, novel power train architecture topics emerge along with the classic aircraft design challenges. In turn, this enquires for a propulsion system analysis tool providing a proper framework for the analysis of the various power train architectures of interest and trade studies amongst them.

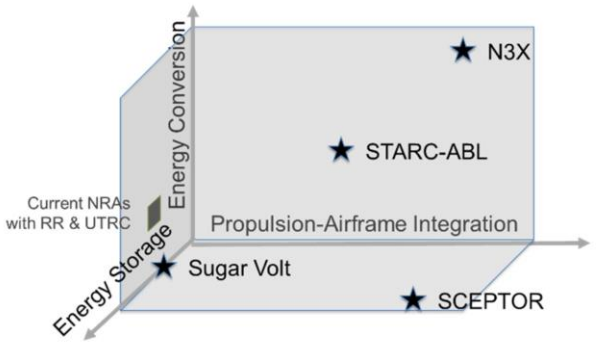


Figure 1. Additional dimensions of the design space availed by hybrid electric propulsion (Jansen, 2016).

1.1 State of the art Modelica libraries

The modeling of hybrid-electric powertrains is one of the first and most prominent applications of Modelica as the DLR Powertrain Library (Tobolář, 2007) for the automotive sector documents since over 10 years.

For aviation, similar approaches have been undertaken in the frame of individual projects. This includes the modeling of more-electric architectures for power and thermal management (Schlabe, 2012, 2015) as well the modeling of unmanned high-altitude platforms operating on solar energy (Klöckner, 2013).

Regarding electric propulsion of passenger aircraft, first studies with Modelica were performed by (Kastner, 2016) and dedicated frameworks are currently built up as in (Batteh, 2018). Meanwhile a number of research projects are dedicated to the topic of electric propulsion and the library of this paper shall be one of the needed building blocks for early design and optimization.

1.2 Specification for a hybrid electric power train modeling environment

The environment is meant to accompany the design process thought the different design stages ranging from basic concept studies to detailed architecture analysis. Thus, one of the main challenges is to pinpoint the required level of detail representing the significant effects for the phenomenon under investigation while neglecting insignificant second order effects with minor contributions to alleviate the computational effort. If the models are interchangeable, the modelling complexity becomes selectable for various investigation interests and can be set to aimed for fidelity level accordingly. To meet these requirements, the framework of library must be object oriented to allow for interchangeable models of various complexities and detailing.

In addition to the different levels of detail in the modeling of components and subsystems, the framework is required to accommodate a variety of potential power train architectures (compare Figure 2). Thus, the interface definition between the components and subsystems becomes a matter of interconnectability, and flexibility in the number of power train subsystems. To enable such flexibility, the modeling framework is obligated to confine the modeling complexity in the subsystems relaxing the computational effort for the simulation of the power train and the equation system for the initialization. This requirement calls again for an object oriented approach which also supplies the user with a building set for the diverse power train architectures.

strong coupling of the subsystems in an aircraft. This apparent coupling inhibits limiting the benefit analysis to the isolated power train since efficiency gains on the power train level are not directly reflected onto the aircraft level. Other effects as additional drag (e.g. for changed/additional nacelles or cooling demand) and additional weight of the power train might cancel the drive train benefits partially or even surpass the benefit inducing an overall disadvantage on system level. Hence, the repercussion of the power train on the other subsystems of the overall aircraft has to be taken into account necessitating a holistic analysis of the hybrid electrically propelled aircraft. To accommodate the variety of domains and disciplines, Modelcia evidently is a proper modelling environment.

Besides the modeling of the components and subsystems, the coordination of the power flow between components embodies a main challenge for the environment framework. Furthermore for consistency purposes, physical system constraints must be taken into account and be avoided during system simulation.

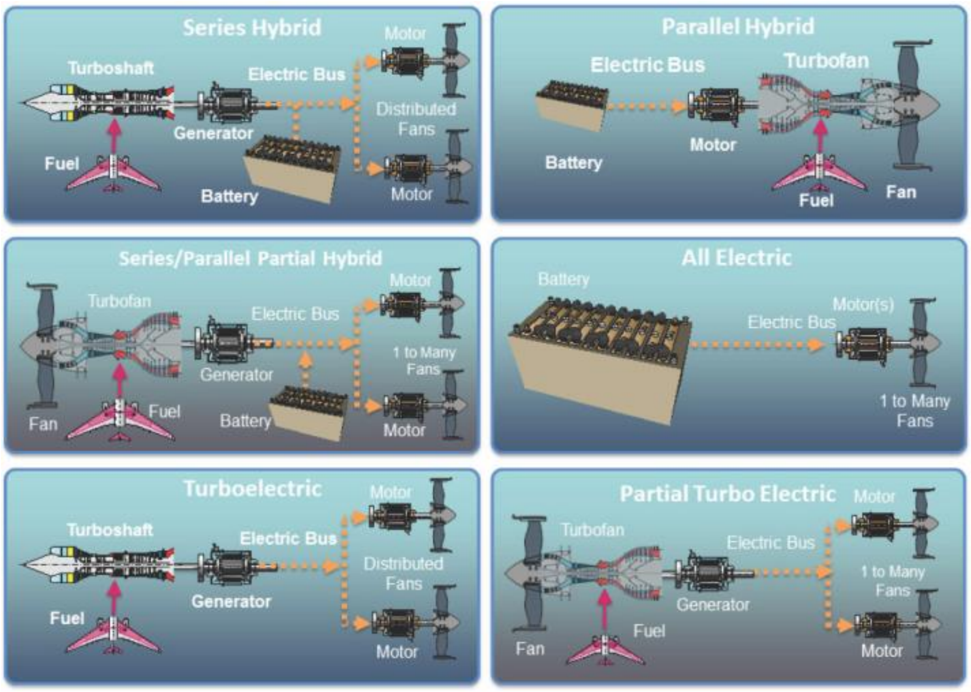


Figure 2. Categorization of the potential power train architectures for hybrid electric propulsion systems in aeronautics (National Academies of Sciences, Engineering, and Medicine, 2016).

In addition to the topology requirements, different operation strategies can lead to components changing their power flow direction during a flight simulation. E.g. an electric motor assisting the main engines during top of climb might be used as a generator to charge the battery pack during cruise. Hence, the non-causal approach from Modelica has an advantage over causal modeling tools like Simulink.

The modelling as well as the conception of hybrid electric aircraft is highly interdisciplinary due to the

The library introduced in this paper solely focuses on the modeling of the power train for hybrid electric propulsion systems. To represent the strong interdependencies between the subsystems, the library is to be combined with another one modelling the lumped aircraft without the drive train. Generally, it covers all the power distribution and conversion components present in a power train. At this point, cooling of the losses is not included in the library even though it constitutes a major challenge in hybrid electric aircraft propulsion. However, the interfaces have already been adapted for future use. Besides the overall power train analysis, the library is intended to also provide means of testing and analysis tools for subsystems of the drive train which could be extended to unit testing for the process of sizing or modeling. During the implementation of the environment in Modelica, issues like the combined initialization of the power train and the aircraft, confining the model complexity in subsystems and defining a common framework for various power train topologies have to be addressed.

2 Organization of the modeling of the hybrid electric power train

In the following, the hierarchical organization of the power train models is presented and the concepts as well as the interfaces for linking the subsystems and portraying the framework for the model interaction are introduced.

2.1 Hierarchical structure of the power train models

The power train models are subdivided into three hierarchy layers: One layer for the overall power train level, one layer for the subsystems and one layer for the components of the respective subsystems of the propulsion units subsequently referred to as power train participants. For the combination of the overall power train model with a lumped aircraft model, an additional top level hierarchy is called for. The layer incorporating this level contains in addition to the hybrid electric power train a flight mission, a reference aircraft as well as the respective autopilot flying the desired mission (see Figure 3).

2.2 The power train level model

To handle the complexity and to ease the orientation of the user, the power train is subdivided into main power train participants (hybrid electric turbofans, electric fans, battery packs, turbofans, turbine propelled generators), a component representing the central DC backbone modeling the grid condition, an interface component for the aircraft power train interface, a central management unit for thrust and power distribution control, and collectors for the thrust and fuel flow (see Figure 4). Besides the physical interconnection of the power train participants via the DC backbone bus transferring the electric power between the components, a second bus is established. This central power train bus handles all power train internal communication and also forwards the aircraft information to the subsystems.

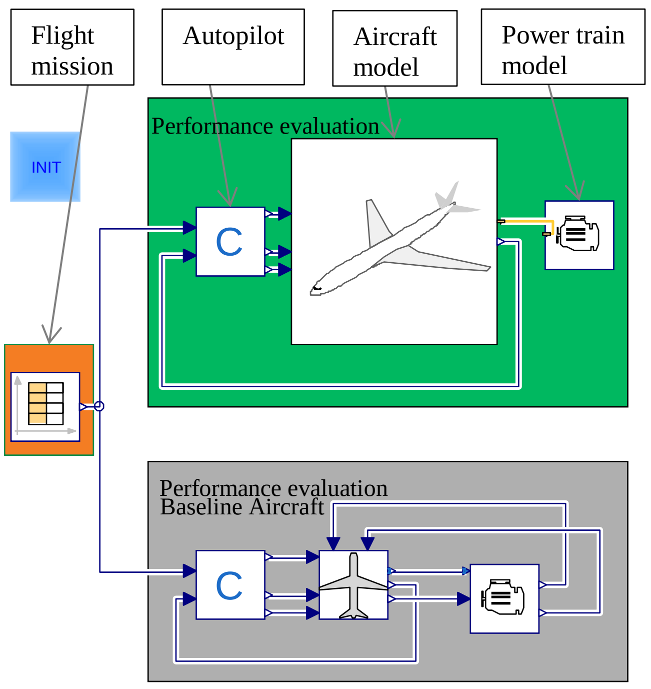


Figure 3. Top level layer for the combination of the power train library with an external lumped aircraft modelling library.

To guide the user through power train models, a graphical template depicting an aircraft sketch is used. The power train participants can be put to their respective position leading to a direct understanding of the basic power train architecture and the interaction between the subsystems. It also serves the system engineer as a standardized representation of the power train.

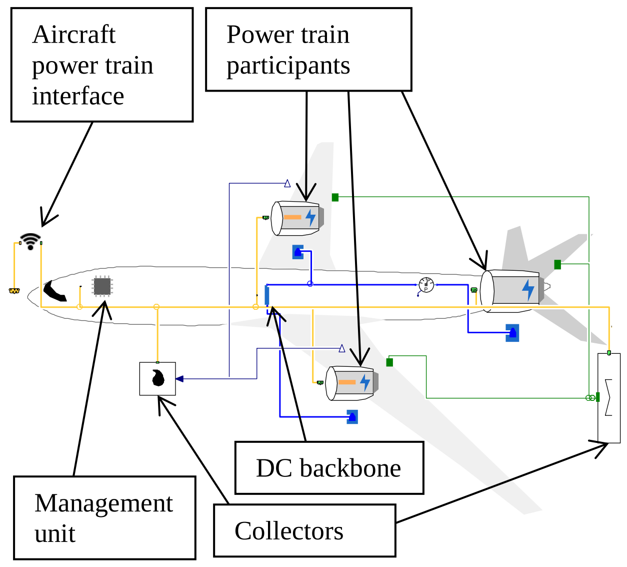


Figure 4. Power train architecture level for an exemplary partial turbo electric power train configuration.

2.3 Aircraft power train interface

The purpose of this component is to define the interface between the aircraft model and the power train model. While the aircraft receives information on the thrust provided by the power train and its fuel consumption, the power train is supplied with atmosphere and flight state information like the velocity and the Mach number the aircraft operates at. The exchanged information is displayed in Table 1. Further to ease the compatibility between power train and aircraft model, the naming conventions of the respective libraries can be mapped in this component.

Table 1. Information flow in the aircraft bus

Information flow from the aircraft to the power train	*Information flow from the power train to the aircraft*
Thrust scheduled by the autopilot	Thrust provided by the power train
Aircraft state: Mach number, true air speed	Fuel flow leading to a variation of the aircraft weight
Atmosphere data: air density, ambient tem-perature (or condensed in altitude information based on the ISA standard atmosphere)	Excess thrust demand surplus to the maximum available thrust of the power train (as feedback information for the autopilot)

Secondly, this interface component can be used to decouple the initialization of the power train and the aircraft model. A simultaneous steady-state initialization of both aircraft and power train model can be numerically challenging for various flight conditions due to the occurring non-linearities and constitutes one of the major challenges in modeling of hybrid electric powertrains.

In more concrete terms: for initializing the aircraft in a specific flight state within its envelope, only the altitude and path angles and the flight velocity/ the attitude are predefined. Hence, the thrust demand is an iteration variable of the aircraft initialization and not specified by the user. The power train is in turn to be initialized for a specific thrust demand and needs to handle numerous interpolations as well as nonlinear equations. Thus, the combined initialization of an aircraft in a quasi-steady state at an arbitrary start point results in a complex equation system which can be of poor condition. Hence, this added complexity resulting from the power train model could impede the overall initialization or possibly lead to degenerated solutions. To ease the initialization, the powertrain and aircraft model can be decoupled at initialization and a subsequent blending process makes the power train and the aircraft model converge to a meaningful common set-point.

2.4 Central power train bus

The central control bus is depicted by the yellow line in Figure 4. This bus broadcasts the atmosphere data as well as flight state data relevant for fans, engine cores and turbofans obtained at the aircraft power train interface, the control signals provided by the central management unit, measured signals required for the management of the power train and the information to be send back to be aircraft model (e.g. fuel mass flow, thrust provided by the power train).

To cope with a variable number of powertrain components, the control and measurement signals are implemented as arrays of variable size. To prevent overlapping of signals in these arrays, a numbering of the components is introduced serving as ID. Its consistency is currently to be granted by the modeler. In future, the UID library may be used instead (Hellerer, 2017). The power train components set and get the information in the respective array position defined by their numbering. The bus is expandable to allow the user to dynamically adapt it to the signals required in the topology or at the aimed for fidelity level, respectively.

2.5 DC-Backbone

The majority of the promising architectures for hybrid electric aircraft propulsion rely on a high voltage DC-backbone (kV region) for the distribution of electric energy in the system. In the example of Figure 4, the DC-Backbone is depicted as blue electric connection between the main components. As found in literature (Shuai *et al*, 2018; Chen, 2012), the basic dynamics of a DC backbone can be modeled by a capacitance relating the difference between feed currents and currents drawn from the grid to a change in the grid voltage. In a physical system, the grid capacitance represents the electric inertia of the cabling and also accumulates contributions from the input/output filter capacitances of the power converts connecting the power train participants to the DC backbone (if not modeled separately). Hence, the change in the grid voltage contains direct information on power distribution imbalances which consequently can be exploited for the management of the power train. To make this information available for the power management unit, it is broadcasted in the central power train bus.

The different actors on the backbone are similar to participants in a micro grid system. In a micro grid the behavior of the participants is organized in multiple levels (Shuai *et al*, 2018). This basic concept is adapted here. In our library two levels are defined: A primitive voltage-level based behavior serving as hard constraint and a top-level coordination logic trying to enforce the desired power flow by a certain management strategy on the components with subject to the lower level voltage based constrains.

In the library, the primitive level does not solely monitor the grid voltage but also the power train participants. Considering that electric machines are relating the rotational speed of the shaft to a certain back EMF voltage, also limitation for the shaft speeds can be implemented on voltage level basis. Moreover, the battery pack voltage can serve as measure for the remaining capacitance and utilized for the consideration of the capacitance limit. Hence, the primitive level assures system protection and robustness in off-design regions potentially caused by improper sizing of the power train and failure cases.

The main power management is realized by a central control logic and sets the current flow between the different power train participants and the DC-backbone as well as the thrust split amount the propulsion units. The sources feeding current to the backbone try to stabilize the voltage while the loads try to achieve defined subsystem set points (e.g. fixing a fan shaft speed according to a desired thrust setting by feeding current into an electric machine) by drawing current from the backbone. Hence, the power exchange with the grid is realized on current level. The low level constraint enforcement is implemented locally in the respective power train components as dynamic saturation for these current signals and linearly down scales the maximum allowable current in vicinity of the voltage bounds.

At this stage, all sub-systems are connected to the DC-backbone via power converters acting according to the two coordination levels and serving as main drivers for the power distribution.

2.6 Central power management model

This component contains the system level logic (top level management) for operating the power train. The control signals calculated within this component are normalized such that all controller outputs range in-between the interval of -1 to 1 for bi-directional components (that may act as both source and load) and 0 to -1/1 for unidirectional components (that may act exclusively as source or load). In the power train participants, this signal is scaled by the maximum power capability of the respective power converter or component. Hence, a value of -1 represents a maximum power to be drawn from the grid while a value of 1 represents a maximum power to be provided by the component. The main power flow direction for the electric systems is hence defined to be towards the grid. For instance, in case of the engine cores, the power flow is unidirectional and ranges from 0 to 1. The normalization of the control signals simplifies the reusability of the respective central management units since proper scaling is not performed centrally in the management logic and becomes sizing independent. All control and measurement signals are broadcasted in the central power train bus (see section 2.4).

Because of the non-linearity in many power train participants, a direct stipulation of the set point by means of algebraic equations may lead to difficult to solve non-linear equation systems. Hence, the implementation of the central power management follows a rapid-prototyping control approach which converges the subsystems to defined set points. This approach allows for confining the nonlinearities in the components and enables the robust modeling and simulation of a variety of different architectures via control states. The goal is hence not to implement a realistic controller but rather the power management is seen as an enabler to simulate the complete system at the desired set-point. Because of the rapid-prototyping approach, simple PI controllers with anti-windup are used. The anti-windup is modified to also consider external saturations and limitations in form of the primitive level of the grid management (introduced in section 2.5).

Also on the level of the power train components, a local control logic may be additionally implemented. The nominal value for the local control variables is again managed centrally in the main management unit. Examples for such a local logic are all forms of turbofans. Since these can provide a set thrust autonomously (subjected to the engine core constraints), a local control is preferable.

However, the engine cores of hybrid electric turbofans can be assisted by an electric machine during critical operation conditions otherwise dominating the sizing. Moreover, they can provide power to the grid if excess power of the engine core is accessible. Hence a central control for the electrical support provides more flexibility.

2.7 Power train component models and their interfaces

The different power train components considered at this stage are (geared) turbofans, (geared) hybrid electric turbofans, electric fans and battery packs. These power train subsystems are the main building blocks for hybrid electric propulsion systems in aviation. Similar to the power train model, a template for organizing these subsystems is provided (as illustrated in Figure 5). The subsystems themselves are built with the aid of component modules allowing for a quick adaption of the fidelity level.

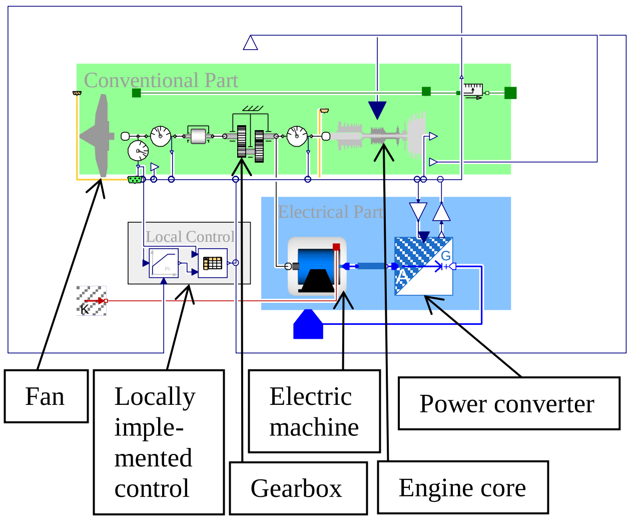


Figure 5. Power train architecture level for an exemplary partial turbo electric power train configuration.

For the junction between fans, engine cores and electric machines, the connector from the rotatory mechanics standard library is chosen since it provides a shaft speed and torque interface. This information is in turn required to calculate the efficiency and the back EMF of the electric machine. Furthermore, the interface is capable of portraying the shaft speed and torque characteristics of fans for different ambient conditions, flight speeds and thrust outputs.

For an electric fan and a hybrid electric turbofan, the sizing torque and the nominal speed of the electric machine are dependent of the operation strategy and the characteristics of the turbomachinery. Thus, this approach also allows for checking the sizing of the electric machine and for analysis of the operation points it is subjected to.

The electric interface is based on the DC interface provided by the standard library. AC subsystems are modeled via DC surrogate models in the dq0 frame.

3 Analysis framework

In correspondence to the hierarchy levels, three levels of analysis are designated in the current state of the library: A component level investigation frame to provide means of performing subsystem and component test in static steady state condition (static environment), a quasi-steady state frame in which the subsystems and components can be subjected to the demands and environment conditions of a classic flight mission (indirect aircraft environment), and a combined simulation of the lumped aircraft and the overall power train (aircraft environment). For the latter a second library contributing a lumped aircraft model, a flight mission, and an autopilot model is required. By default a library based on the Base of Aircraft Data 3.7 standard (BADA) (for details see (Eurocontrol Experimental Centre, 2009)) is availed as default. This library encompasses three degrees of freedom aircraft models, a standard mission and an autopilot. The BADA standard also contains information on the classic propulsion system for the included aircraft. Therefore, it can also serve as reference configuration for classical propulsion systems. As aforementioned, the combined simulation of the lumped aircraft and the power train is the level of analysis providing meaningful flight performance results for hybrid electric aircraft. The other two levels of analysis can be utilized as intermediate analysis for building up architectures from the scratch and to preliminary tune their parameters if no external sizing tool is accessible. Furthermore, these steps can be used as check for external sizing results. However, the quasi-steady state simulation can also be instrumented for simply assessing the influences on power train level. A summary of the capabilities for the 3 levels is given in Table 2.

Table 2. Capabilities of the different test levels.

Testing Capabilities	*static environ-ment*	*indirect aircraft environ-ment*	*aircraft environ-ment*
For the thrust in critical design points on sub-system level	X	-	-
For the thrust in critical design points on power train level	X	-	-
for quasi-steady state simulations	-	X	X
For hand tuning the sizing of components in a subsystem	X	X	-
For analysis of the power train level benefit	-	X	-
For analysis of the aircraft level benefit	-	-	X

4 Exemplary a power train analysis

4.1 Description of the power train architecture

For demonstration purposes, a parallel hybrid with two main hybrid electric turbofans (see Figure 6) is analyzed with the presented Modelica library. In the selected analysis scenario, the engine cores of the hybrid electric turbofans are undersized and are backed up by electric machines. The power for the electric machines is supplied to the DC-backbone by a battery pack. The power train is operated such that the electric motors buffer the power demand of the fans if the engine cores approach the vicinity of their power limit. To show the capability of a combined simulation of the aircraft and the power train, an aircraft environment type simulation is selected utilizing the default BADA library. As aircraft data set, A320 is picked. The mission profile is based on a typical medium distance mission for an A320.

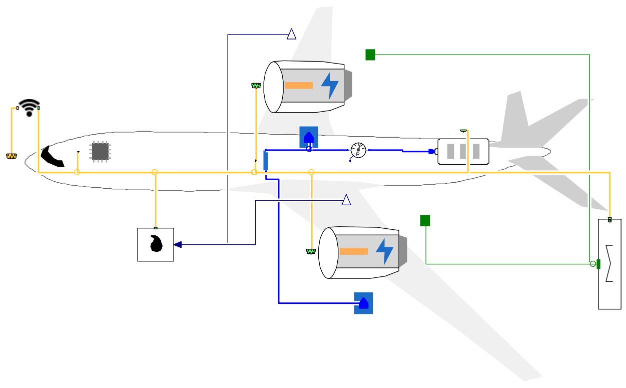


Figure 6. parallel hybrid with two main hybrid electric engines.

4.2 Analysis revealing main features of the library

As depicted in Figure 7, during takeoff and in the approach phase, the engine cores can provide all the power required to propel the fans. Close to the final cruise altitude (at ca. 5300 s), the electric machines back up the engine cores to provide sufficient power for fans.

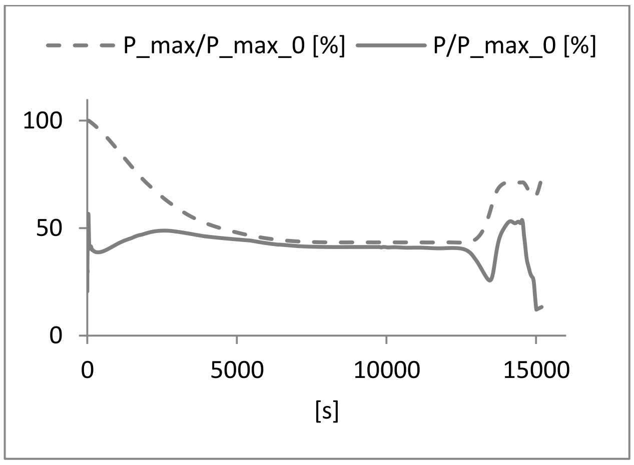


Figure 7. Power provided to one fan by the respective engine core (solid line) and maximum power of the respective engine core (dashed line) normalized by the maximum takeoff power [%].

The power consumed by the electric motors is drawn from the grid by the respective power converters. Consequential, the power converter controlling the power flow of the battery pack tries to fix the voltage set point for the backbone by feeding power into the backbone. At ca. 6300s the power rating of the battery pack inhibits the controller dominating the battery pack behavior from stabilizing the backbone (as apparent in Figure 8). Hence, the backbone voltage drops (see Figure 9) until the vicinity of the lower operation bound of the backbone is reached. At this point the current rating for the motor control is down scaled by the primitive level to equilibrate the power rating of the battery pack and the motors. As soon as the power rating becomes adequate to balance the full load demand, normal grid operation is restored (as illustrated in Figure 9 at ca. 7200 s). In the meantime, the thrust demand of the autopilot cannot be achieved due to a lack in power rating. For normal mission operation, this issue does not arise for a properly sized power train because the sizing is also considering component faults.

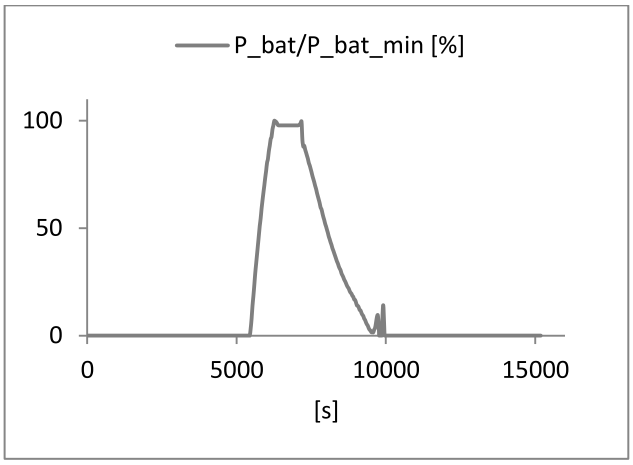


Figure 8. Power supplied to the DC-backbone by the battery pack normalized by the maximum output power [%].

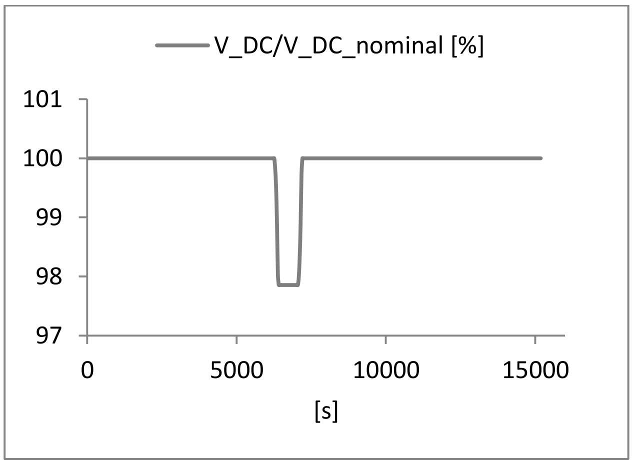


Figure 9. DC-backbone voltage normalized by the nominal voltage [%].

The drop in its energy content of the battery caused by the power feed to the grid is depicted in Figure 10. It can be seen that a surplus of energy remains for the selected mission.

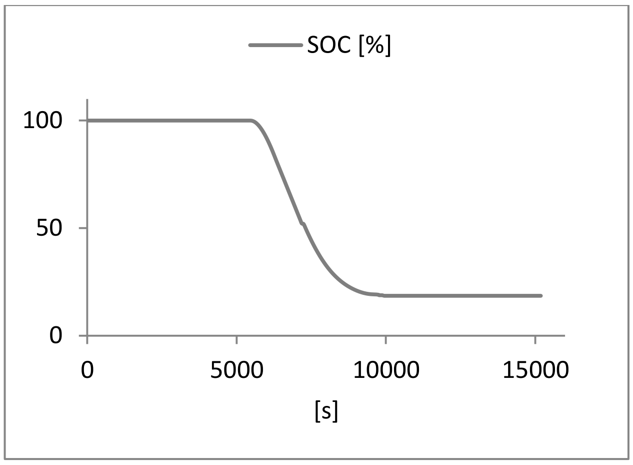


Figure 10. state of charge of the battery pack[%].

5 Conclusion and future work

The current library can handle varying numbers of power train participants and gives insight into the first order interactions between the power train participants. It can serve as analysis tool for pre-sized power trains. Furthermore, the capability of a combined simulation of an aircraft model and the power train over a whole flight mission has been proven.

Currently, the thrust collector does not consider the position of the different propulsion units. For extending the library for simulation of system failures also the torques created by off centered thrust must be expressed by the power train model. Furthermore, a coupling the library with a six degree of freedom capable aircraft modeling environment like DLR Flight Dynamics library (Klöckner 2014) will be undertaken.

The main missing part of the analysis is the modeling and sizing of the cooling system for the electric components since its additional weight and power demand in form of electricity and drag has a significant effect on aircraft level.

References

Eurocontrol Experimental Centre. User Manual for the Base of Aircraft Data (BADA) Revision 3.7, 2009.

European Commission. Flightpath 2050: Europe’s Vision for Aviation, 2011.

Ralph H. Jansen, Cheryl Bowman and Amy Jankovsky. Sizing Power Components of an Electrically Driven TailCone Thruster and a Range Extender. *16th AIAA Aviation Technology, Integration, and Operations Conference*, 2016, doi: 10.2514/6.2016-3766.

Loan T. F. W. Silva, Lucas P. Resende and Marcelo A. Tomim. Mathematical modeling and numerical simulation of locomotives electrical drive systems in Modelica, *Brazilian Power Electronic Conference (COBEP)*, pp. 1-8, 2017.

Zhikang Shuai, Junbin Fang, Fenggen Ning, Z. John Shen. Hierarchical structure and bus voltage control of DC microgrid. Renewable and Sustainable Energy Reviews, Volume 82, Part 3, pp. 3670-3682, 2018.

Dong Chen and Lie Xu. Autonomous DC Voltage Control of a DC Microgrid With Multiple Slack Terminals. *IEEE Transactions on Power Systems*, vol. 27, no. 4, pp. 1897-1905, 2012.

National Academies of Sciences, Engineering, and Medicine. Commercial Aircraft Propulsion and Energy Systems Research: Reducing Global Carbon Emissions. *The National Academies Press*, 2016, doi: 10.17226/23490.

Martin Hepperle. Electric Flight - Potential and Limitations. *NATO Science and Technology Organization*, 2012, ULR: https://elib.dlr.de/78726/, [retrieved 2018].

Matthias Hellerer and Fabian Buse (2017) Compile-time dynamic and recursive data structures in Modelica. In: Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT), Munich, Germany.

Klöckner, Andreas, Looye, Gertjan et. al. (2014) Object-Oriented Aircraft Modeling with the DLR FlightDynamics Library. 9th AIRTEC 2014 International Congress, 28 – 30 Oct. 2014, Frankfurt, Germany.

Jakub Tobolář, Martin Otter and Tillman Bünte: Modelling of Vehicle Powertrains with the Modelica In: Systemanalyse in der Fahrzeugtechnik IV.

Klöckner, Andreas und Leitner, Martin und Schlabe, Daniel und Looye, Gertjan (2013) Integrated Modelling of an Unmanned High-Altitude Solar-Powered Aircraft for Control Law Design Analysis. In: Advances in Aerospace Guidance, Navigation and Control. EuroGNC 2013, 2nd CEAS Specialist Conference on Guidance, Navigation & Control, April 10-12, 2013, Delft, The Netherlands.

Schlabe, Daniel (2015) Modellbasierte Entwicklung von Energiemanagement-Methoden für Flugzeug-Energiesysteme. Dissertation, Technische Universität Dresden.

Schlabe, Daniel und Zimmer, Dirk (2012) Model-Based Energy Management Functions for Aircraft Electrical Systems. SAE Power Systems Conference 2012, 30. Okt. - 01. Nov. 2012, Phoenix, USA.

Kastner, Nir (2016). Modelica environment of hybrid-electric propulsion. Presentaiton on electric & hybrid technology symposium. Cologne, Germany.

John Batteh et. al. (2018) Development and Implementation of a Flexible Model Architecture for Hybrid-Electric Aircraft In: the American Modelica Conference 2018.

Advances in Flight Dynamics Modeling and Flight Control Design by Using the DLR Flight Visualization and Flight Instruments Libraries

Daniel Milz[1] Christian Weiser[1] Franciscus L.J. van der Linden[1] Matthias Hellerer[1] Andreas Seefried[1] Tobias Bellmann[1]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR), 82234 Weßling, Germany,
{daniel.milz, christian.weiser, matthias.hellerer, andreas.seefried, tobias.bellmann}@dlr.de

Abstract

This paper presents the Flight Instruments and Flight Visualization Libraries developed within the DLR Institute of System Dynamics and Control. For the design of dynamic models and control systems, a visual evaluation of the dynamic simulation is indispensable for a successful design and test process. Especially when it comes to aircraft models, an overview of the overall dynamics is needed. Therefore, the presented libraries provide fast assembly of fully configurable and generic flight visualization tools in which the view positions as well as the setup of Primary Flight Displays can be chosen freely. This provides the visual components of a rapid prototyping environment which can be used in the development of flight dynamic models and flight control laws. Moreover, the camera views and displays can easily be reconfigured for each purpose and research focus area. Furthermore, the libraries can be used for desktop simulation, motion simulator experiments as well as flight testing on a real aircraft.
Keywords: Visualization, Flight Simulation, Flight Control

1 Introduction

Modelica has become an important language in the field of flight dynamics modeling and flight control design (Looye 2008). Moreover, the possibility of rapid prototyping within these fields has become a key technology within the development of modern aircraft (Looye 2007a). In addition to the enhancement of this rapid prototyping process for flight control laws (Looye 2007b), a visualization framework has been developed and proven helpful in supporting the design process in terms of configuration analysis, simulation and experiments.

In the field of engineering, particularly in the area of complex model and control design, visualization of physical data is indispensable for an efficient and successful proceeding. R.B. Haber already stated the importance of visualization for engineering (Haber 1990).

Until now, there is no integrated solution available for flight dynamics visualization in Modelica. Therefore, based on the Modelica Visualization Library (Bellmann 2009; Hellerer et al. 2014), the Flight Visualization and Flight Instruments Libraries have been developed. These libraries open up the possibility of a completely accessible, in-house developed aircraft visualization and simulation environment with the following main applications to be used within a rapid prototyping process:

- Desktop Simulation. The visualization models and displays can be fitted for analysis and demonstration video purposes to display relevant information directly on the screen.
- Motion Simulator Experiments.
- Flight Testing. The created flight instrument and data display views can be reused on mobile devices inside real aircraft.

Section 2 introduces the Flight Visualization Library and its components, such as the modeling of flexible structures. Following, Section 3 gives insight into the implementation of different data displays and shows exemplary setup of a standard Primary Flight Display (PFD). In a further step, Section 4 shows synergies generated through combination of both libraries. Moreover, additional features such as on-board three-dimensional virtual reality views to be used with commercial virtual reality glasses are introduced. As a conclusion of the presented work, Section 5 contains examples for easy usability of the presented framework within the process of flight dynamic modeling and flight control law development.

2 Aircraft Visualization

The DLR Flight Visualization Library is based on the DLR Visualization Library (Bellmann 2009) adding high level structures for a rapid design of flight visualization models.

Figure 1 depicts a visualization of a dynamic aircraft simulation generated with the Flight Visualization Library. In this example, a flexible aircraft structure is displayed and the forces which cause the deflection of the wing are depicted in different colors.

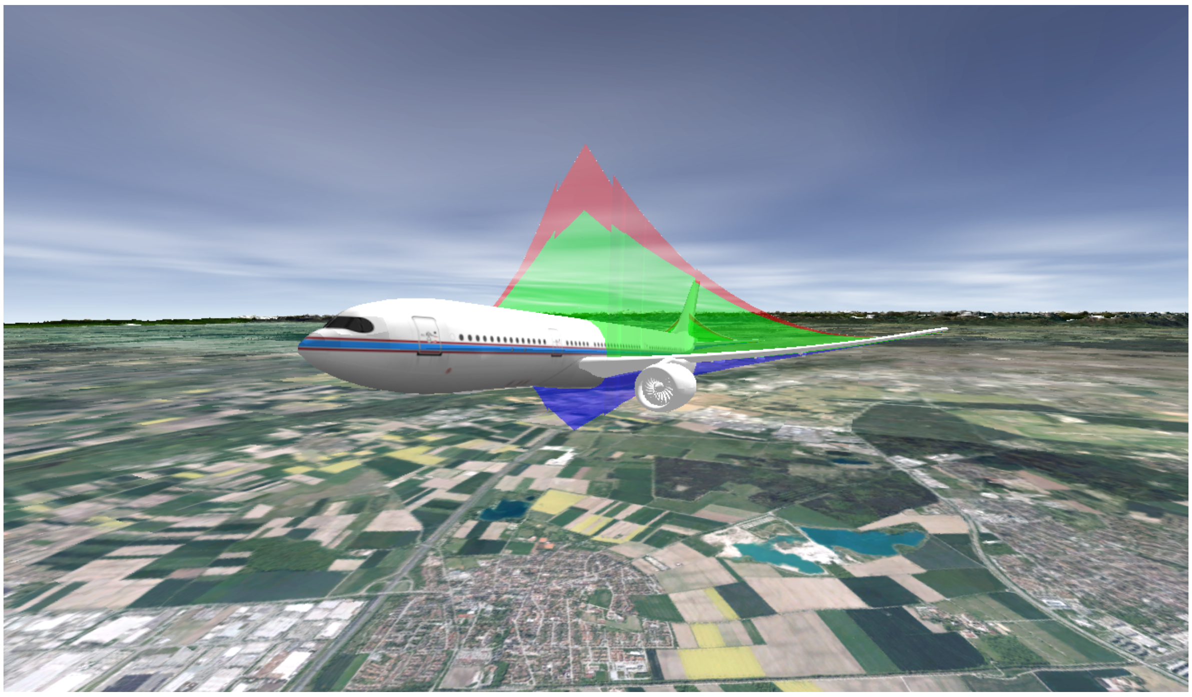


Figure 1. Example of a visualization created with the DLR Flight Visualization Library.

On the one hand, the DLR Flight Visualization Library can be used for creating stand-alone models that communicate by various internet protocols, e.g. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Via these protocols, input data containing the current flight state and additional information on the aircraft status are received. Alternatively, a predefined trajectory can be stored internally.

On the other hand, this library can directly be integrated in various Modelica dynamic models that are e.g. models created by the DLR Flight Dynamics Library (Looye 2007a). The created models can be individualized in the following points:

- Data sources. Data in- / output using TCP / IP and UDP protocols receive data from the main flight dynamics and flight control law models.
- Terrain. Terrain files which are referenced to geodetic coordinates.
- Aircraft models. Aircraft models can be built up by one single or an assembly of multiple components.
- Camera position. Different cameras can be placed with options of switching / moving the view during the simulation.

This brings up a generic framework to visualize all known aircraft configurations as well as other vehicles that have a structure similar to aircraft such as e.g. underwater vehicles. Figure 2 shows the top level of an aircraft visualization.

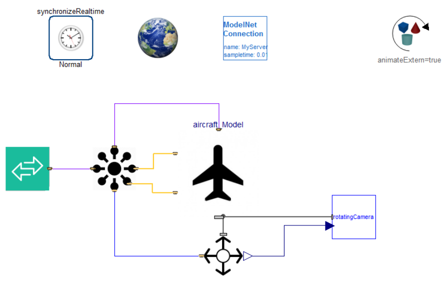


Figure 2. Top Level view of an Aircraft Visualization Model.

2.1 Aircraft Model

The aircraft model contains all visible components of the airframe structure, such as fuselage, engines, actuators. For simple simulations, one CAD file with geometry and texture is sufficient. However, the modeling of the airframe can be extended to more sophisticated setups, in which each movable component is modeled as a separate object. With this approach, control surface positions may be displayed during the simulation and can be useful as visual feedback of control action.

Flexible Models Furthermore, the results of load calculations for a flexible aircraft (Kier and Hofstee 2004) can be integrated into the aircraft visualization. For the integration of this feature, the displacement of the structural grid points obtained from the dynamic, flexible simulation is mapped to the 3D object file (Heckmann et al. 2006). One powerful method to calculate real-time deformations of free-form surfaces is the use of poly-harmonic splines defined via radial basis functions (Botsch and Kobbelt

2005).

Polyharmonic Splines A polyharmonic spline is a linear combination of radial basis functions (RBFs) denoted by:

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(|\mathbf{x} - \mathbf{c}_i|) + \mathbf{v}^T \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \qquad (1)$$

where $\mathbf{x} \in \mathbb{R}^3$ is the current vertex to be transformed, $\mathbf{c}_i \in \mathbb{R}^{N\times 3}$ denotes all center points, $\mathbf{w} \in \mathbb{R}^N$ represents the weights of the RBFs and $\mathbf{v} \in \mathbb{R}^{4\times 3}$ denotes the weights of the polynomial. In this case, the RBF $\phi(x)$ is defined as $\phi(x) = x^3$.

The function $f(\mathbf{x})$ calculates the displacement of a point on a flexible element. The absolute position is thus calculated as $\mathbf{x}_{\text{displaced}} = \mathbf{x} + f(\mathbf{x})$. The center points are reference points of which the deflection is known. This deflection is transformed into the weights w_i and inserted into the function.

Implementation During simulation time the calculation of the object displacements are GPU accelerated and therefore sufficiently high frame rates and smooth movement display are achieved. The GPU acceleration is implemented by adding the polyharmonic spline calculation for the vertices into the corresponding shader that iterates over all vertices. This is an essential tool for visualizing e.g. reactions to gust and turbulence and qualitative assessment of load alleviation.

2.2 Environment/ Terrain

The terrain modeling supports texture files with varying resolutions. The terrain files are mapped to geodetic longitudes and latitudes. The terrain is created using the OpenSceneGraph (OSG) Virtual Planet Builder (Pordes et al. 2007) by combining georeferenced digital elevation models (DEM) data with texture data. The Virtual Planet Builder then generates an OSG based terrain database with several levels of detail (Hellerer et al. 2014).

For simulation of landing maneuvers and collision detection, feedback from the visualization is fed back into the dynamic model in order to achieve accurate visualization results for touchdown and ground contact.

3 Cockpit Visualization

Besides a Modelica library for the external visualization of aircraft, a library for internal visualization aiming at cockpit displays is available through the DLR Flight Instruments Library. This library includes all established cockpit displays as head-up and head-down instruments and a framework for the 3-dimensional realization of virtual reality cockpits. This virtual reality framework is introduced in Section 4.1.

The two-dimensional cockpit displays are suitable for various applications including overall flight system testing, design of flight control laws, additional displays for real flight tests and many more. The main displays inside an aircraft are the Primary Flight Display (PFD), the Navigation Display (NAV) and the Electronic Centralized Aircraft Monitoring (ECAM) or Engine Indication and Crew Alerting System (EICAS). Those components consist of a set of different generic elements such as linear bars and dial gauges. In addition, aerospace related elements, e.g. an artificial horizon display, are built up.

Moreover, there is an interface to external databases and APIs included. Those may provide additional information such as navigational aids and the current or simulated weather.

3.1 General Components

The library does not only consist of high level and aircraft specific display instruments but also of low level and general use displays: A dial gauge and a linear bar instrument.

Dial Gauge The dial gauge is a standard instrument not only in terms of aerospace. Within aerospace applications it is mainly used in order to display engine data or information related to the flight control system. Figure 3 visualizes different engine parameter gauges. In addition to the marking of desired and critical domain regions for the displayed value, a change of background color gives a visual alert in case of a value exceeding its limits. The limit parameters can be adjusted quickly using a GUI based Modelica editor in order to support rapid prototyping use in experimental setups.

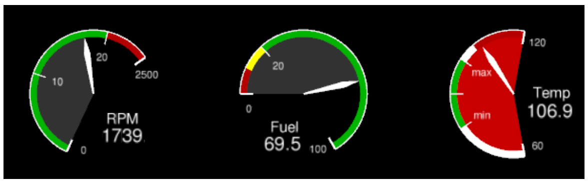


Figure 3. Screenshot of different dial gauges.

Linear Bar The linear bar represents an important instrument for modern and digital flight displays. Especially digital PFDs use bar instruments for displaying airspeed and altitude. The bar includes arrows for trend values and color-based markings for validity of a signal region.

3.2 Primary Flight Display (PFD)

The primary flight display (PFD) is the most important display in an aircraft. The main component of the PFD itself is the artificial horizon that shows the flight attitude. Modern digital PFDs show the current flight attitude, velocity and altitude of the aircraft.

Artificial Horizon The artificial horizon shows the current roll attitude Φ (Phi) and pitch attitude Θ (Theta). For civil aircraft, the center of the artificial horizon is a dot representing the aircraft's nose. In general, an artificial horizon consists of a blue area, the sky, and a brown area, the earth. This is depicted in Figure 4. The Modelica implementation of the sky surface is shown in the following code snippet.

In the beginning the model and the used variables are declared.

```
model ArtificialHorizon
    // Roll and Pitch angle
    input Real phi, theta;
    // Points defining the horizon
    Real horizon[2,2];
    // Extra points to span the two areas
    Real expt[3,2];
    // Points spanning the sky surface
    Real spt[6,2];
    // Surface, DLR Visualization Library
    Face sky(points=displaysize/2 * spt);
equation
    // ...
end ArtificialHorizon;
```

This model only takes the roll and pitch angle as an input and displays a simple artificial horizon that consists of only one surface. The design of the artificial horizon corresponds to current implementations where the visualization completely fills a quadratic display. The two horizon points are calculated as a transformation of the points `[-1,0; 1,0]` around the center `[0;0]`. Subsequently, the transformation of `horizon` and `expt` is stated.

```
if abs(horizon[1, 2]) <= 1 then
    horizon[1,1] = -1;
else
    horizon[1, 1] = minmax(interpolate({
        horizon[1, 2],-1},{horizon[2, 2],
        horizon[2, 1]},sign(horizon[1, 2]))
        );
end if;

horizon[1, 2] = -tan(phi) -theta;

if abs(horizon[2, 2]) <= 1 then
    horizon[2, 1] = 1;
else
    horizon[2, 1] = minmax(interpolate({
        horizon[1, 2],horizon[1, 1]},{
        horizon[2, 2],1}, sign(horizon[2,
        2])));
end if;

horizon[2, 2] = tan(phi) -theta;

expt[1, :] = {horizon[1,1],interpolate(
    horizon[1, :],horizon[2, :], expt[1,
    1])};
expt[2, :] = {horizon[2,1],interpolate(
    horizon[1, :],horizon[2, :], expt[2,
    1])};
expt[3, :] = {1,extrapolate(horizon[1, :],
    horizon[2, :], expt[3, 1])};
```

The `minmax` command restricts the input value to an interval $[-1;1] \subset \mathbb{R}$. Furthermore, `interpolate` and `extrapolate` take two points and a single x-value as an input. The functions calculate the y-value corresponding to the x-value on a line spanned by the two points. The `sky` surface is spanned by six two-dimensional points. Those are finally computed by the following equations:

```
spt[1,1] = if horizon[1, 2] >= 1 then
    horizon[1, 1] else -1;
spt[1,2] = 1;
spt[2,1] = if horizon[2, 2] >= 1 then
    horizon[2, 1] else 1;
spt[2,2] = 1;
spt[3,1] = if spt[3, 2] >= 1 then
    extrapolate({horizon[1, 2],horizon[1,
    1]},{horizon[2, 2],horizon[2, 1]},1)
    elseif horizon[2, 2] >= 1 then horizon
    [2, 1] else 1;
spt[3,2] = minmax(extrapolate(horizon[1,
    :],horizon[2, :],1));
spt[4,1] = if spt[3, 2] <= -1 then
    extrapolate({horizon[1,2],horizon
    [1,1]},{horizon[2,2],horizon[2,1]},-1)
    else expt[2, 1];
spt[4,2] = if spt[3, 2] <= -1 then minmax(
    extrapolate(horizon[1,:],horizon
    [2,:],1)) else minmax(expt[2, 2]);
spt[5,1] = expt[1, 1];
spt[5,2] = minmax(expt[1, 2]);
spt[6,1] = if horizon[1, 2] >= 1 then
    horizon[1, 1] else -1;
spt[6,2] = minmax(horizon[1, 2]);
```

The earth surface can be implemented analogously to the sky surface.

In Figure 4 it is shown that the current PFD setup can easily be adapted to show additional information compared to standard PFDs used in commercial aircraft. In the scope of the different works on adaptive control (Lombaerts et al. 2016; Lombaerts et al. 2018) the PFD was extended to display various flight envelopes. This is illustrated by the colored bars and areas within the display.

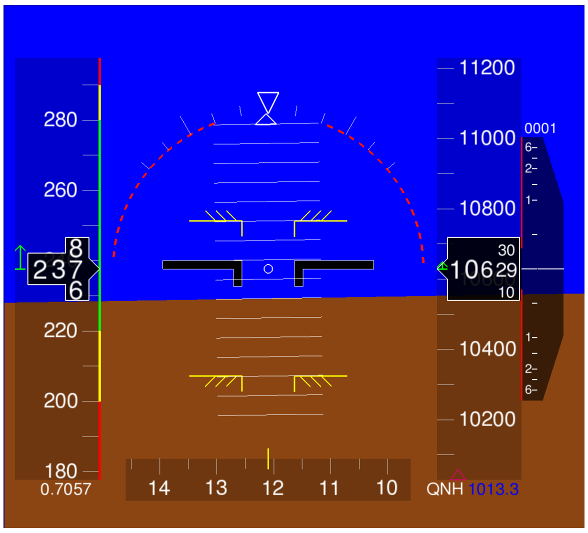


Figure 4. Primary Flight Display.

3.3 Navigation Display (NAV)

The navigation display shows a map of the surrounding navigational aids. Those include airports, VORs and many more. Furthermore, surrounding aircraft are displayed. The data can either be predefined in the model or retrieved during run-time from a database by various APIs

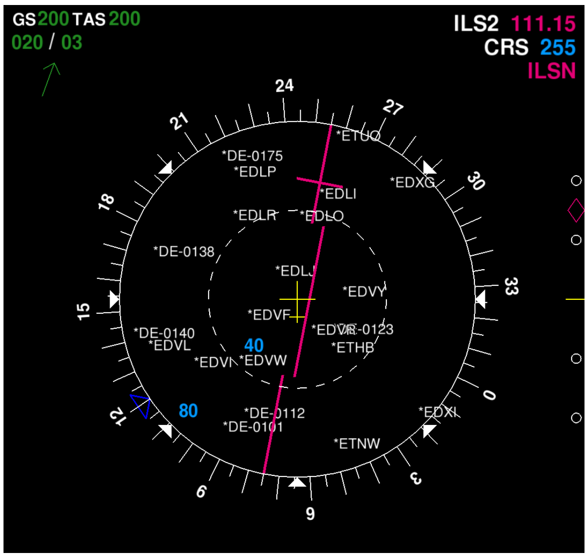


Figure 5. Navigation Display. Current simulated position near Bielefeld, Germany, while intercepting radial 255 of a radio navigation aid.

provided. Inside the DLR, a SQL database containing most of the worldwide navigational aids exists. From this database, the navigational aids within a selected radius are retrieved every second. These are stored in an array and visualized by elements provided by the DLR Visualization Library. More details on the implementation of a SQL interface is given in Section 4.3. Figure 5 illustrates an exemplary navigation display.

3.4 Additional Displays

In Figure 6, the ECAM or EICAS is an additional display that visualizes engine data such as shaft speed and fuel flow as well as the current health state of the aircraft and messages from the flight control system (FCS).

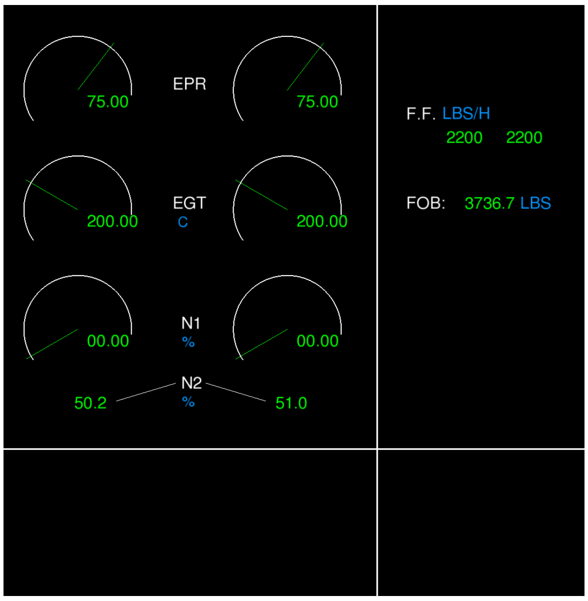


Figure 6. ECAM or EICAS display.

4 Linking of the Libraries and Additional Features

After the introduction of the two developed libraries, their linking through a common data interface is enforced in order to achieve additional benefits within the aircraft modeling and control design process. Besides, additional functions and features relevant for both libraries are introduced.

4.1 Virtual Reality Environment

A three-dimensional virtual reality cockpit perfectly fits the use for realistic pilot training and flight tests. This harmonizes with the DLR Robotic Motion Simulator described in (Bellmann et al. 2011).

For motion simulation, the visual cues are of exceptional importance. Therefore, a head mounted display (HMD) is used with a high detailed cockpit. The displays can be used within this virtual environment and be placed at the positions of the 'real' displays.

The virtual reality environment can be used in a ground based simulator as well as in combination with a motion platform, as described in Section 5.

4.2 Interaction of external and internal visualization

To add full compatibility of the two introduced libraries, a standardized interface to communicate between the models using a bus system is created. This allows the use of different predefined cockpits with individual instruments. It is not necessary to connect every display input as Modelica assumes a zero input on all unconnected bus inputs. This is in particular helpful for fast testing of new flight controllers as the user only needs to connect the needed sensor signals, leaving additional displays input blank. If required for further testing or for development of a demonstration model, the remaining sensor inputs can be connected for a fully operational cockpit.

This standardized interface allows the development of several cockpits and displays based on a single flight model. This feature has already proven to be helpful in flight tests, when the pilots shall receive a different flight display than the flight test engineers (Linden et al. 2018; Grondman et al. 2018).

To be able to use the flight instruments library with a wide range of flight models, interface converters are part of the library that convert signals from different inputs such as UDP / TCP connections directly from a real time flight computer or a Simulink model.

4.3 SQL Database Integration

In the scope of the introduced work, a generic SQL database interface is developed. M. Tiller already introduced a generic data retrieval Modelica library that is particularly designed for XML and MATLAB files (Tiller 2005). Although the use of SQL databases is promised, this solution was not available when creating this library.

Furthermore, no satisfying Modelica SQL interface is available to the point where the development of this libraries started. This motivated the creation of an own SQL interface. The relational database API is implemented in an object-oriented manner and within extenal C funcitons. Based on the `ExternalObject` parent class to manage the Modelica objects, a system of various different functions to connect, disconnect and query databases is implemented.

5 Application Examples

One major application for the previously introduced framework is the DLR Robotic Motion Simulator (RMS) (Bellmann et al. 2011). The simulator is depicted in Figure 7 and uses a robotic arm instead of a hexapod to simulate aircraft movement and accelerations. Inside the gondola, the user is wearing virtual reality glasses which provide a first person view from the pilot's seat including a visualization of the cockpit. Here, the use of the VR technology has the big advantage that no view projection is needed and the whole field of vision is covered.

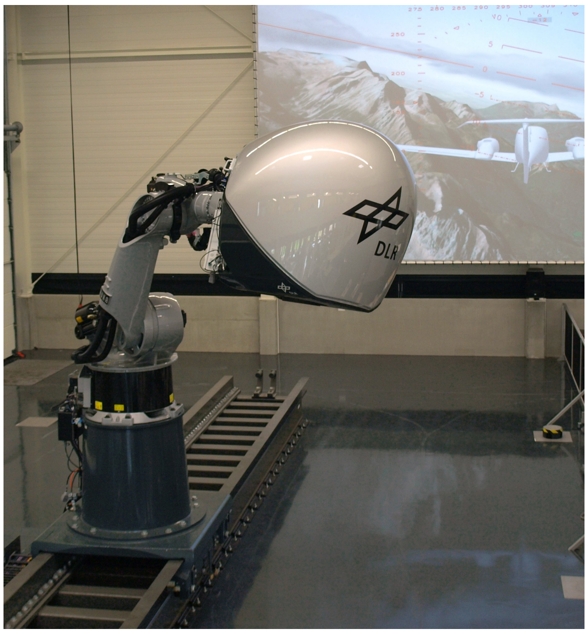


Figure 7. The DLR Robotic Motion Simulator.

A further example is the use of primary flight displays during real flight test. For this purpose, the same display as used during the design process in the desktop and simulator evaluation can be used on tablet computer and with an additional interface receive all relevant data which shall be displayed from the flight test instrumentation. This has been a major advantage during several flight test campaigns on novel flight control laws (Grondman et al. 2018) as well as current and force control for the actuation system (Linden et al. 2018). For this purpose, additional gauges displaying the actuator position and current are added to the Modelica flight displays.

Figure 8 shows the use of the Modelica Flight Instruments Library on-board the test aircraft during flight preparation. In the center of the picture a standard consumer tablet running the model can be seen. This includes a standard PFD in the top half of the picture and additional instruments and graphs for experiment relevant information in the bottom half.

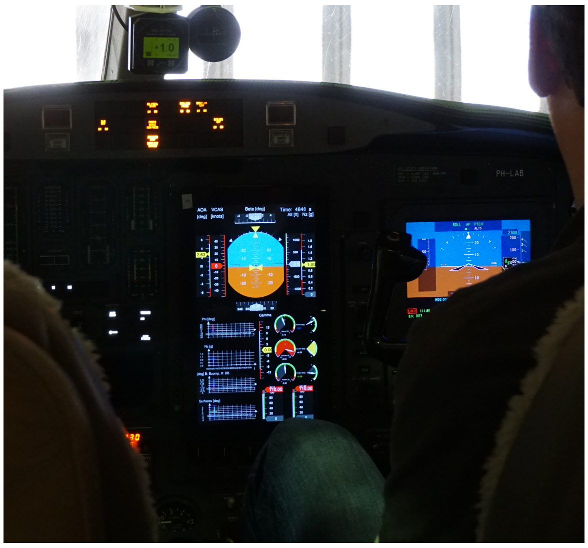


Figure 8. Cockpit instruments in use for flight testing on a Cessna Citation aircraft.

6 Conclusions

This work introduced two libraries for visualization of dynamic aircraft simulations. The libraries are not only restricted to use in aircraft simulation, but can also be used for any dynamic vehicle simulation, e.g. underwater vehicles. The first part covered is the simulation of a moving vehicle in an environment, which provides the user with easy accessible visual information of the vehicle's attitude and moving direction. Secondly, instruments and displays can efficiently be used to embed important information into the visualization model or to generate a separate display. Thus, the overall process chain from model development, control design, simulation and testing is further enhanced and simplified. Additionally, all steps of the process chain are kept completely accessible for the engineer and no "black box" visualization tools are required. Both libraries are currently not distributed since development and extension are still ongoing.

References

Bellmann, Tobias (2009). “Interactive Simulations and advanced Visualization with Modelica”. In: *Proceedings of the 7 International Modelica Conference Como, Italy*. Linköping University Electronic Press. DOI: `10.3384/ecp09430056`.

Bellmann, Tobias, Johann Heindl, Matthias Hellerer, Richard Kuchar, Karan Sharma, and Gerd Hirzinger (2011). “The DLR Robot Motion Simulator Part I: Design and setup". In: *IEEE International Conference on Robotics and Automation*. DOI: 10.1109/icra.2011.5979913.

Botsch, Mario and Leif Kobbelt (2005). "Real-Time Shape Editing using Radial Basis Functions". In: *Computer Graphics Forum* 24.3, pp. 611–621. DOI: 10.1111/j.1467-8659.2005.00886.x.

Grondman, Fabian, Gertjan Looye, Richard O. Kuchar, Q. Ping Chu, and Erik-Jan Van Kampen (2018). "Design and Flight Testing of Incremental Nonlinear Dynamic Inversion-based Control Laws for a Passenger Aircraft". In: *2018 AIAA Guidance, Navigation, and Control Conference*. American Institute of Aeronautics and Astronautics. DOI: 10.2514/6.2018-0385.

Haber, R.B. (1990). "Visualization techniques for engineering mechanics". In: *Computing Systems in Engineering* 1.1, pp. 37–50. DOI: https://doi.org/10.1016/0956-0521(90)90046-N.

Heckmann, Andreas, Martin Otter, Stefan Dietz, and José Díaz López (2006). "The DLR FlexibleBody library to model large motions of beams and of flexible bodies exported from finite element programs". In: *5th International Modelica Conference*, pp. 85–95. URL: https://elib.dlr.de/47219/.

Hellerer, Matthias, Tobias Bellmann, and Florian Schlegel (2014). "The DLR Visualization Library - Recent development and applications". In: *The 10th International Modelica Conference 2014*. Linköping Electronic Conference Proceedings. LiU Electronic Press, pp. 899–911. URL: https://elib.dlr.de/92153/.

Kier, T. and J. Hofstee (2004). "VARLOADS - Eine Simulationsumgebung zur Lastenberechnung eines voll flexiblen, freifliegenden Flugzeugs". In: *Deutscher Luft- und Raumfahrtkongress, Dresden, 20.-23. September 2004*. Vol. I & II. LIDO-Berichtsjahr=2004, monograph_id=DGLR-JT 2004-240, URL: https://elib.dlr.de/12207/.

Linden, Franciscus L. J. van der, Gertjan Looye, and Tijmen Pollack (2018). "Aircraft control using actuator current". In: *International Conference on Recent Advances in Aerospace Actuation Systems and Components 2018*, pp. 196–202. URL: https://elib.dlr.de/120314/.

Lombaerts, Thomas, Gertjan Looye, Andreas Seefried, Miguel Neves, and Tobias Bellmann (2016). "Development and Concept Demonstration of a Physics Based Adaptive Flight Envelope Protection Algorithm". In: *IFAC-PapersOnLine* 49.5. 4th IFAC Conference on Intelligent Control and Automation SciencesICONS 2016, pp. 248–253. DOI: 10.1016/j.ifacol.2016.07.121.

Lombaerts, Thomas, Gertjan Looye, Andreas Seefried, Miguel Neves, and Tobias Bellmann (2018). "Proof of concept simulator demonstration of a physics based self-preserving flight envelope protection algorithm". In: *Engineering Applications of Artificial Intelligence* 67, pp. 368–380. DOI: 10.1016/j.engappai.2017.08.014.

Looye, Gertjan H. N. (2007a). "An Integrated Approach to Aircraft Modelling and Flight Control Law Design". PhD thesis. Delft University of Technology.

Looye, Gertjan H. N. (2007b). "Rapid Prototyping Using Inversion-Based Control and Object-Oriented Modelling". In: *Lecture Notes in Control and Information Sciences*. Springer Berlin Heidelberg, pp. 147–173. DOI: 10.1007/978-3-540-73719-3_8.

Looye, Gertjan H. N. (2008). "The New DLR Flight Dynamics Library". In: *6th Modelica Conference*. URL: https://elib.dlr.de/55670/.

Pordes, Ruth et al. (2007). "The open science grid". In: *Journal of Physics: Conference Series* 78.1, p. 012057. URL: http://stacks.iop.org/1742-6596/78/i=1/a=012057.

Tiller, M (2005). "Implementation of a generic data retrieval API for Modelica". In: *4th Modelica conference*. URL: https://www.modelica.org/events/Conference2005/online_proceedings/Session7/Session7b2.pdf.

SESSION 4D: NUMERICAL METHODS

DAE Solvers for Large-Scale Hybrid Models

Erik Henningsson[1] Hans Olsson[1] Luigi Vanfretti[2]

[1]Dassault Systèmes AB, Lund, Sweden, {Erik.Henningsson, Hans.Olsson}@3ds.com
[2]Rensselaer Polytechnic Institute, Troy, NY, USA, vanfrl@rpi.edu

Abstract

We present a strategy for DAE mode simulations of large-scale Modelica models with state events. DAE solvers can be orders of magnitudes faster than traditional ODE solvers when simulating models with large algebraic loops. Such loops are common in, for example, power grid models. Central for our DAE mode approach is the accurate and efficient treatment of state events. Adapting, extending, and optimizing results known in the literature to the Modelica context resulted in a DAE mode implementation first released in Dymola 2019 and 3DEXPERIENCE 2019x. The implementation is verified by efficiency experiments featuring OpenIPSL power grid models. The run times for these models are competitive with domain-specific, state-of-the-art simulation tools.

Keywords: DAE mode, hybrid model, state event, large-scale, Modelica, power grid model

1 Introduction

As a high-level, equation based, and object-oriented language Modelica promotes easy construction, modification and reuse of models. It is therefore well suited for modeling large-scale, integrated physical systems, see e.g. (Baudette et al., 2018; Casella et al., 2016; Jorissen et al., 2015).

With the increased presence of such large-scale models, higher demands are put on Modelica tools to facilitate fast simulations. To meet those demands, special model structures are typically analyzed and exploited (Casella, 2015). Examples of strategies that have been successfully realized in Modelica tools, such as Dymola and the 3DEXPERIENCE platform, involve: multirate simulation (Thiele et al., 2014), mixed-mode simulation (Schiela and Olsson, 2000; Thiele et al., 2014), model decoupling and parallel execution (Elmqvist et al., 2014), and sparse solvers (Braun et al., 2017).

In this paper we will consider the strategy referred to as *DAE solver* or *DAE mode*. The name comes from the mathematical representations of Modelica models: hybrid *differential-algebraic equations (DAEs)*. When generating simulation code a Modelica tool performs a series of symbolic transformations involving common subexpression elimination, equation sorting, index reduction, and tearing (Cellier and Kofman, 2006). During this process the high-index DAE is transformed into an index-1 DAE, and then, by solving systems of equations, it is normally transformed into an ordinary differential equation (ODE). The latter can be integrated by an ODE solver like CVode (Hindmarsh et al., 2005). For most models the transformations make the numerics simpler and result in more robust and efficient simulations.

However, some numerical integrators, such as Dassl (Brenan et al., 1996), also allows integration of the index-1 DAE directly. For certain models, such DAE mode simulations can be orders of magnitude faster, among other things, due to more efficient treatment of algebraic loops. Significant speed-ups have, for example, been observed when simulating national- and continental-sized models of electrical power systems (Braun et al., 2017). Rosenbrock DAE integrators were used by (Olsson et al., 2017) to achieve fast and predictable run times for model-based embedded control.

The goal of this paper is to present a strategy for robust, accurate, and efficient simulation of *hybrid* DAEs using DAE integrators like Dassl. Because integration of the index-1 DAE is well-understood, the main focus will be on how to accurately and efficiently localize and treat state events. To achieve this, we will argue for an approach where the same generated code is used in DAE mode as in ODE mode. By using all the symbolic transformations and optimizations, the DAE fed to the integrators is kept to a minimal size. The trade-off here being some loss of sparsity (Braun et al., 2017; Magnusson, 2016). As an outlook we will also discuss further benefits and possibilities enabled by this approach.

Based on this strategy, DAE mode for hybrid DAEs was introduced in Dymola 2019 (released in June 2018) and 3DEXPERIENCE 2019x for a diverse selection of numerical integrators. The efficiency and accuracy of the implementation is verified by simulations of the Nordic power grid model *Nordic 44* from the OpenIPSL library (Vanfretti et al., 2017).

2 DAE mode for hybrid systems

2.1 Mathematical formulation

Mathematically, Modelica models are represented by hybrid DAEs. That is, differential-algebraic equations that may have discontinuities and/or may be controlled by discrete variables and conditions that change at events.[1] The

[1]Note that varying-structure models and multi-mode simulations are out of scope of this paper.

general form of the DAE is

$$F(t,\dot{x},x,y,d) = 0, \qquad (1)$$

where t denotes the independent time and y the algebraic variables. Further, x are the differential variables and $\dot{x}$ are their time derivatives. The discrete variables, which may change value only at events, are denoted by d. For the initial value problem to be well-defined, appropriate initial conditions must also be supplied.

Throughout the simulation time- and state-dependent crossing functions

$$c = q(t,\dot{x},x,y,d) \qquad (2)$$

are monitored for sign changes. The variable c represents the conditions of all if- and when-clauses. At zero-crossings an event is triggered. If the corresponding crossing function q_i depends on any of $\dot{x}$, x, or y it is called a state event, otherwise a time event. The former are more difficult to locate and their combination with DAE mode simulations is the main topic of this paper. When an event is triggered a reinitialization is performed using the DAE (1) and the crossing equations (2) together with additional discrete equations

$$d = \eta(t,\dot{x},x,y,\mathrm{pre}(d),c). \qquad (3)$$

Here $\mathrm{pre}(d)$ are the previous values of the discrete variables. For details see (Olsson, 2017, Appendix C). This combination of equations defines a continuous-discrete mixed system of equations to be solved for the derivatives $\dot{x}$, the algebraic variables y, the discrete variables d, and the conditions c. Together, the three systems (1) – (3) define a hybrid differential-algebraic equation.

To construct simulation code a Modelica tool, such as Dymola, applies several symbolic transformations to the original hybrid DAE. These steps involve e.g. common subexpression elimination, sorting, index reduction, and tearing. For details see (Cellier and Kofman, 2006). During the process of reducing the index to one, the number of differential variables may decrease and the number of algebraic equations may increase. As the goal of these transformations is to transform the DAE (1) into an ODE the tool will select states $x(t) \in \mathbb{R}^{n_x}$ and solve for the state derivatives. Each derivative $\dot{x}_i$ that cannot be solved for symbolically is replaced by an algebraic variable $\hat{x}_i$ and the equation $\dot{x}_i = \hat{x}_i$.

For a typical Modelica model a significant part of the algebraic variables y in the original DAE (1) do not affect the dynamics of the model. These are the auxiliary variables and are not required during continuous simulation. We will therefore exclude them from the DAE provided to the numerical integrator. However, they must be computed when evaluating the crossing equations (2) and the discrete equations (3) and we must therefore consider them when locating and resolving events.

The symbolic transformations turns the original system into a sequence of assignment statements intertwined with smaller (nonlinear) systems of equations, the algebraic loops. These loops are then torn to minimize their size (Elmqvist and Otter, 1994). Denote by n_G the number of loops that affect the dynamics. Further, denote by z_i the iteration (tearing) variables of loop i, where $z_i(t) \in \mathbb{R}^{n_i}$, $n_i \geq 1$. These normally constitute a small subset of the algebraic variables. The algebraic loops are represented by the systems of equations

$$0 = G_i(z_i; t,x,z_1,\dots,z_{i-1},d), \qquad (4)$$

for $i = 1,\dots,n_G$. Due to the equation sorting each algebraic loop is independent of later ones. The functions G_i do not depend on the state derivatives $\dot{x}$, since these were either solved for or substituted for an algebraic variable as described above.

The goal of the symbolic transformation is to causalize the DAE into an ODE,

$$\dot{x} = \hat{f}(t,x,d). \qquad (5)$$

Here, the algebraic variables z and the algebraic loops are internal to the ODE. Thus, the evaluation of $\hat{f}$ requires the solution of the systems of equations defined in (4), which may be solved in sequence, separate from each other.

For the DAE mode approach presented in this paper we consider the DAE in the form it takes after all of the symbolic transformations have been performed, with one exception: the iterative solution of algebraic loops. Note that the loops that can be solved symbolically are still solved in that way, involving linear loops and the inversion of elementary functions like sin. Thus, we elevate, from the function $\hat{f}$, the loops that cannot be solved symbolically and arrive at the semi-explicit, index-1 DAE of interest for this paper

$$\dot{x} = f(t,x,z,d), \qquad (6a)$$
$$0 = G(z;t,x,d), \qquad (6b)$$

where $z = (z_1^{\mathrm{T}},\dots,z_{n_G}^{\mathrm{T}})^{\mathrm{T}}$ and $G = (G_1^{\mathrm{T}},\dots,G_{n_G}^{\mathrm{T}})^{\mathrm{T}}$. The important difference between $\hat{f}$ and f is that the evaluations of the latter do not require the solution of the loops (4), rather the algebraic variables z are inputs.

The remaining algebraic variables y can be computed from the variables in Equation (6). The computation of the subset of y that affects the dynamics is internalized in f and G. By construction, these computations are merely assignment statements.

Similarly, the computations of y can be internalized in the crossing functions, giving

$$c = Q(t,\dot{x},x,z,d).$$

Note that computing the auxiliary part of y may involve solving further (torn) algebraic loops, not considered in the dynamics.[2] Thus, computing all of y from

[2]Alternatively these algebraic loops for auxiliary variables may also be handled by the integrator, resulting in better predictors for the involved variables.

the variables in Equation (6) may be expensive. However, the crossing functions themselves q_i are typically cheap. Therefore, computing several crossing functions Q_i with the same input is not significantly slower than computing just one.

2.2 Continuous simulation in ODE mode

Since Modelica models are often stiff, implicit numerical time-stepping schemes are commonly used to integrate the ODE (5). This procedure requires, at each time step, the solution of one or more nonlinear systems of equations inside the integrator. For example, for a multistep method, such as the BDF methods implemented in Dassl, the system

$$\frac{1}{Ch_n}x_n - \hat{f}(t_n, x_n, d_n) = \text{old}(\hat{f}, x), \tag{7}$$

has to be solved for the next approximation x_n of the state. Here C denotes a method-dependent constant, h_n is the current step size, and $\text{old}(\hat{f}, x)$ is a linear combination of old $\hat{f}$-evaluation and x-approximations. Similar equations have to be solved to find the stages if using an implicit Runge–Kutta method, such as the Radau schemes (Hairer and Wanner, 1996).

The integrator equation (7) is typically solved by a quasi-Newton iterative method. In each iteration a linear system of equations is solved using the Jacobian

$$\hat{J} = \frac{1}{Ch_n}\text{I} - \frac{\partial \hat{f}}{\partial x},$$

where I is the identity matrix. Even though the Jacobian is not updated each iteration, in fact not even with each time step, the evaluation of $\frac{\partial \hat{f}}{\partial x}$ is one of the major bottlenecks when simulating a large Modelica model.

The Jacobian $\hat{J}$ is normally approximated numerically using finite differences, which requires a large number of $\hat{f}$-evaluations. As previously mentioned, each $\hat{f}$-evaluation requires the solution of the algebraic loops (4). Thus, solving Equation (7) may involve treating nested nonlinear systems of equations.

To illustrate the problem, consider that the cost for constructing and factoring a Jacobian often grows superlinearly in the number of variables. This complexity depends on the sparsity structure of the Jacobian and what other optimizations are applied. In the worst case scenario the construction cost can grow quadratically and the factorization cost cubically. With this in mind, the cost for constructing (but not factoring) the integrator-Jacobian $\hat{J}$ can be approximated as

$$c_{\hat{J}} \approx \text{const.} \cdot n_x^{p_x - 1} \cdot \left(c_{\text{rem}}(n_x) + \sum_{j=1}^{n_G} n_i^{p_i} \right),$$

where $p_x \in [1,2]$ and $p_i \in [1,3]$ depend on how well sparsity and other optimizations can be utilized. The last factor is the cost of one $\hat{f}$-evaluation, where $c_{\text{rem}}(n_x)$ denotes the total cost of everything in the $\hat{f}$-evaluation, except the algebraic loops. This term typically grows with the number of states n_x. The factor $n_x^{p_x-1}$ is the number of $\hat{f}$-evaluations required. For certain models, the size n_i of some of the algebraic loops may be as large as the number of states n_x or even larger. Note that several optimizations are applied in Dymola to keep the exponents p_x and p_i small, with the aim to minimize the above cost. Especially when constructing the integrator Jacobian.

Further $\hat{f}$-evaluations are required to compute the residuals in the Newton iteration for the next step (7). Also at output points and events the model must be evaluated. However, for models like Nordic 44, considered in Section 4, the construction of integrator Jacobians dominate the simulation cost in ODE mode.

2.3 Continuous simulation in DAE mode

When integrating using the DAE mode proposed in this article, the algebraic loops are elevated and solved by the integrator. Rather than hiding the loops in Equation (5) they are handed to the integrator via the semi-explicit DAE formulation (6). When evaluating f the algebraic variables must be known prior to the evaluation. This means that, in DAE mode, the integrator has to handle also the iteration variables z.

One important benefit of the DAE mode approach presented in this paper is that the problem size is kept to a minimum by using the sorting and tearing information. The state vector consists of the vectors x and z. If the DAE solver was instead applied directly to the DAE (1) it would have had to solve for x and all of y.

Applying e.g. a multistep method to Equation (6) yields, in analog to Equation (7), the integrator equations

$$\begin{gathered} \frac{1}{Ch_n}x_n - f(t_n, x_n, z_n, d_n) = \text{old}(f; x, z), \\ G(z_n; t_n, x_n, d_n) = 0, \end{gathered} \tag{8}$$

to be solved for the next approximations x_n and z_n. The corresponding Jacobian needed for the quasi-Newton solution of these equations is

$$J = \begin{pmatrix} \frac{1}{Ch_n}\text{I} - \frac{\partial f}{\partial x} & -\frac{\partial f}{\partial z_1} & -\frac{\partial f}{\partial z_2} & \cdots & -\frac{\partial f}{\partial z_{n_G}} \\ \frac{\partial G_1}{\partial x} & \frac{\partial G_1}{\partial z_1} & 0 & \cdots & 0 \\ \frac{\partial G_2}{\partial x} & \frac{\partial G_2}{\partial z_1} & \frac{\partial G_2}{\partial z_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial G_{n_G}}{\partial x} & \frac{\partial G_{n_G}}{\partial z_1} & \frac{\partial G_{n_G}}{\partial z_2} & \cdots & \frac{\partial G_{n_G}}{\partial z_{n_G}} \end{pmatrix},$$

where we have considered the algebraic loops separate from each other to reveal the sparsity pattern that is given by construction.

The cost for a numerical approximation of the DAE mode Jacobian J differs in its structure from the cost of the ODE Jacobian $\hat{J}$. An approximation of the complexity can be written as

$$c_J \approx \text{const.} \cdot \left(n_x + \sum_{j=1}^{n_G} n_i \right)^{p-1} \cdot \left(c_{\text{rem}}(n_x) + \sum_{j=1}^{n_G} n_i \right),$$

where $p \in [1,2]$ depends on how effectively sparsity can be used to minimize the number of f-evaluation. Since the state vector is longer in DAE mode the Jacobian is larger and more right-hand-side evaluations are typically needed. However, instead of solving the algebraic loops the DAE solver just inquires for their residuals G_i giving the smaller second factor in the cost. Comparing with the cost for approximating $\hat{J}$ one can conclude that computing one Jacobian in DAE mode is much faster when there are large algebraic loops with non-trivial sparsity structure ($p_i > 1$). Throughout this paper these are the type of models we consider.

2.4 Event handling in ODE mode

During integration the crossing functions (2) are monitored, applying special care to correctly handle multiple zero crossings in the same crossing function.

When zero crossings are detected in one or more crossing functions, the state x is interpolated and a root finding algorithm is applied to accurately find the time of the first zero crossing. Each of the crossing functions $\hat{Q}_i$ is scalar-valued and defines, together with Equation (5), a system of nonlinear equations

$$\begin{aligned} \dot{x} &= \hat{f}(t, P^x(t), d), \\ 0 &= \hat{Q}_i(t, \dot{x}, P^x(t), d), \end{aligned} \tag{9}$$

in the variables t and $\dot{x}$. The interpolation polynomial for the state x is denoted by $P^x(t)$. Note that the algebraic variables z have here been internalized in Q_i giving $\hat{Q}_i$, compare with f and $\hat{f}$ in Equations (5) and (6).

Noting that the algebraic variables are solved for with a high accuracy in ODE mode, the above root finding approach guarantees that the solution will be in a consistent state when a crossing is detected and an event iteration is started. See for example (Eich-Soellner and Führer, 2008, Chapter 6) for details on how to locate the first zero crossing using iterative methods.

2.5 Event handling in DAE mode

When simulating in DAE mode the integrator handles the algebraic variables z and approximates them to fit the supplied integrator tolerance. Similarly to the ODE case we may use the, now extended, state vector $(x;z)$ to monitor the crossing functions (2). Further, we may interpolate the whole extended state when solving for the first crossing

$$\dot{x} = f(t, P^x(t), P^z(t), d), \tag{10a}$$
$$0 = Q_i(t, \dot{x}, P^x(t), P^z(t), d). \tag{10b}$$

However, the integrator tolerance is several orders of magnitude larger than the tolerance for the solver of the algebraic loops in ODE mode. This means that the algebraic equations (4) will generally not be fulfilled at the time of crossing (10). The upcoming event iteration will then start in an inconsistent state; the algebraic equations not being accurately fulfilled. Severe problems may be experienced when Equations (1), (2), and (3) are to be simultaneously solved for a consistent restart state.[3]

Example 1. Consider the Modelica Standard Library model `EngineV6`, which is a multibody model of a V6 engine, see Figure 1. To compute the force generated by the combustion in an engine cylinder the piston velocity is monitored. An event is triggered and the integration is restarted when a piston velocity changes direction.

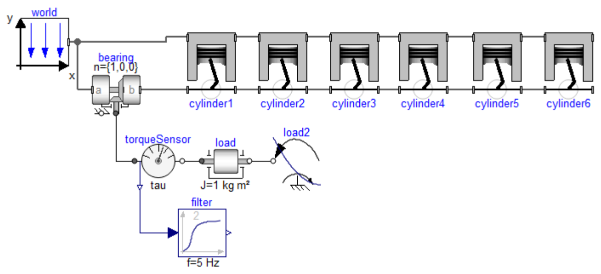


Figure 1. `EngineV6`, a multibody model of a V6 engine from the Modelica Standard Library.

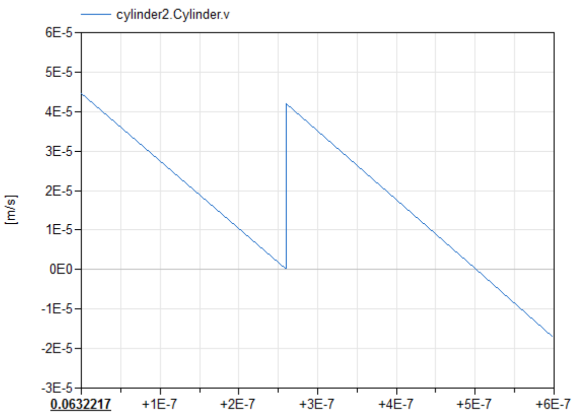


Figure 2. Inconsistent solution state causing empty events and problems to locate the correct time of the zero crossing.

Monitoring the crossing functions and locating the zero crossings according to Equation (10) gives the result of Figure 2, where the piston velocity of the second cylinder is used as an example. At time $t \approx 0.06322196$ Equation (10) signals for a zero crossing and an event is localized. Starting the event iteration, the crossing time t and interpolated state $P^x(t)$ are used as input and the algebraic variables y are solved for, cf. Section 2.1. Being an algebraic variable, the piston velocity `cylinder2.Cylinder.v`, is thus solved for with high

[3]Similarly, the state derivatives $\dot{x}$ may also be interpolated rather than computed from Equation (10a). However, the same problems as when interpolating z are to be expected.

accuracy. With the integrator error removed from the velocity it jumps up to its consistent value of approximately $4 \cdot 10^{-5}$ m/s. This renders the event empty since the corresponding crossing function $Q_i(t, \dot{x}, P^x(t), z, d)$ is again positive and no discrete variables were changed. The integration restarts and the process repeats a few times (cannot be seen in the figure since the subsequent velocity jumps are of very small scale). When sufficiently close to the correct crossing time $t \approx 0.06322220$ the event is at last correctly handled. △

To make sure that the solution is in a consistent state at zero crossings in DAE mode it is suggested by (Eich-Soellner and Führer, 2008, Section 6.3.8) to accurately solve the algebraic loops when crossing functions are evaluated. We will here adopt a slightly generalized and modified version of this idea. In our setting, and for each Q_i, we consider the following systems of equations

$$\dot{x} = f(t, P^x(t), z, d), \tag{11a}$$

$$0 = G(z; t, P^x(t), d), \tag{11b}$$

$$0 = Q_i(t, \dot{x}, P^x(t), z, d), \tag{11c}$$

to be solved when locating a zero-crossing. In contrast to the simple generalization (10) of the ODE case, where also the algebraic variables z were interpolated, we here only interpolate the original states x. The algebraic variables must be solved for using Equation (11b).

For each Q_i the system (11) has the unknowns t, $\dot{x}$, and z. These systems must be solved with high accuracy to guarantee that we get the correct crossing time and a consistent state for the event iteration. Additionally, solving the algebraic loops (11b) may be expensive, as we have previously discussed. However, the benefit of having the sorting and tearing information available here is that we do not have to solve for all the algebraic variables y simultaneously, rather the remaining can be computed from z. In Section 3.3 we will discuss further optimizations that can be applied to efficiently and accurately monitor and locate events in DAE mode.

3 Dymola DAE mode implementation

In Dymola and the 3DEXPERIENCE platform, DAE mode has been implemented for the solvers Dassl, Radau IIa, Sdirk34hw, and Esdirk of different orders. Thus offering a selection of both multistep and Runge–Kutta methods. With one of these solvers selected DAE mode is enabled by the command

```
Advanced.Define.DAEsolver = true.
```

In the current section we will present a few key features of this implementation. Most importantly the accurate and efficient handling of state events.

3.1 Reusing the simulation code

When initializing and when locating and resolving events the algebraic loops need to be accurately solved. For efficiency and robustness it is typically beneficial to use all the optimizations applied when generating ODE mode code, including e.g. sorting and tearing. This means that the code generated for ODE mode is needed also when integrating a hybrid system in DAE mode. To keep the simulation code simple and the code duplication to a minimum our implementation strategy is therefore to reuse the ODE code to the greatest extent possible.

As seen in Section 2.3 this strategy also allows us to minimize the size of the integrator equation (8) during continuous simulation in DAE mode. However, the drawback here is that tearing may cause fill-ins, i.e. the reduced nonlinear system $G_i = 0$ may be less sparse than the original system. In some cases this may even make simulations slower (Braun et al., 2017). On the other hand, it is concluded by (Magnusson, 2016) that tearing typically is beneficial for DAEs resulting from hierarchical Modelica models. One may additionally gain in efficiency by taking care to reduce fill-ins when tearing, especially for large models. To which extent tearing should be used and how it should be applied is considered out of scope of this paper.

As discussed in Section 2 the only difference between ODE and DAE mode continuous simulation is how the algebraic loops are handled. In ODE mode the algebraic loops (4) are solved for the algebraic variables z during the $\hat{f}$-evaluations. By small changes in the simulator code that handles the algebraic loops, these loops can easily be modified to instead take z as an input from the integrator. The input can then be used to compute the residuals $G(z)$, which are returned to the integrator for correction. All this without any need to change the code generation or the generated simulation code itself.

3.2 Utilizing the Jacobian structure

So far no changes are required to the generated simulation code. However, there is one piece of auxiliary information needed for efficient simulations.

The sparsity pattern of the integrator Jacobian is analyzed by Dymola when constructing the simulation code. Knowing all explicit dependencies of the functions $\hat{f}$ on the variables x (respectively f on $(x; z)$), Dymola can reduce the number of function evaluations required to construct a numeric Jacobian. Several columns can be grouped together and computed at the same time by utilizing column independencies.

Since the dependencies change in DAE mode the ODE mode analysis can not be reused. The DAE Jacobian J is larger and typically more sparse, cf. (Braun et al., 2017, Section 2.2). For example, partial explicit dependencies in each algebraic loop can be taken into account when constructing the sparsity pattern for the DAE mode Jacobian. In contrast, in ODE mode, where the algebraic loops are solved, all the iteration variables z_i for each loop depend on all of the loop inputs. To summarize, this enables Dymola to be more aggressive when constructing column groups in DAE mode.

Moreover, due to the increased size and sparsity of the integrator Jacobian, the benefits of using sparse linear algebra for storing and factorization are even greater. In Dymola multithreaded SuperLU (Li, 2005) is used for this task and is fully compatible with DAE mode simulations.

3.3 Efficient and accurate event localization

In Section 2.5 we demonstrated that correctly monitoring and locating crossings can be expensive in DAE mode. Interpolating the algebraic variables z led to problems and instead the full equations (11) had to be considered. Additionally, to avoid having to discard time steps or output points it is important to evaluate the crossing functions often during continuous integration, normally after each time step. Considering that each solution of the equations may be expensive, the aggregated cost may become a major bottleneck for DAE mode simulations of hybrid models. However, there are several optimizations that can be performed to considerably shorten the time needed for event localization.

First assume, for the sake of simplicity, that there is only one crossing function Q and consider how zero crossings are typically localized in ODE mode. It is straightforward to rewrite Equation (9) as an equation

$$0 = \hat{Q}\big[t,\ \hat{f}(t,P^x(t),d),\ P^x(t),\ d\big],$$

in the single, scalar variable t. This equation can be efficiently solved with an iterative method, e.g. using regula falsi variants like the safeguard techniques (Eich-Soellner and Führer, 2008, Section 6.3.2).

The same techniques can be adopted in the DAE case to efficiently solve the system (11). This results in the nested systems of equations

$$0 = Q\Big[t,\ f\Big(t,P^x(t),G^{-1}\big(0;t,P^x(t),d\big),d\Big),\\ P^x(t),\ G^{-1}\big(0;t,P^x(t),d\big),\ d\Big].$$

In the outer equation the time of the crossing t is the only unknown. Given an approximation of t the polynomial P^x can be evaluated. Using this, the algebraic variables can be solved for from $0 = G(z)$. Then $\dot{x}$ can be computed and finally the crossing function residual. This nonlinear equation in t can be solved by applying the same iterative root finding techniques as in ODE mode.

Indeed, as the algebraic loops are solved also when $\hat{f}$ is evaluated the above described procedures for (9) and (11) are equivalent. This means that the same code can be used for event localization in ODE and DAE mode. The only difference is that, in the latter case the simulator code handling the algebraic loops must be told to solve them, and to do this with high accuracy. After the event is fully handled the code must again be told to only compute the residuals of the loops for continuous DAE mode simulation.

Handling several crossing functions at the same time is neither significantly more expensive, nor more difficult, cf. Section 2.1. Thus, the above discussed solution technique for crossing equations is easily generalized to several crossing functions.

Finally, and perhaps most importantly, we note that accurately solving the algebraic loops is only important when finding the correct crossing time and during the event iteration. During continuous simulations, and when no crossing function is close to zero, it is enough to consider the more direct formulation first discussed in Section 2.5. That is, we use the integrator approximations of both x and z and after each time step we evaluate

$$\begin{aligned} \dot{x} &= f(t,x,z,d), \\ c &= Q(t,\dot{x},x,z,d). \end{aligned} \tag{12}$$

If any of the variables c_i is close to zero we must switch to the accurate crossing function handling (11) so the correct crossing time can be located.

With these optimizations Dymola can accurately and efficiently locate and resolve state events. The algebraic loops must only be solved to a high accuracy when closing in on a crossing, when localizing the crossing, and when resolving the event. Typically, only on the order of ten solutions per state event is required, cf. Section 4.2. A remaining problem is how to efficiently handle the situation where a crossing function is close to zero throughout most of the simulation, but never crosses.

4 Application Example – Nordic 44

To verify the efficiency of the Dymola DAE mode implementation we here perform experiments with the Nordic 44 power grid model (Vanfretti et al., 2017).

4.1 Model description and test cases

Nordic 44 consists of 44 buses, 61 controlled generators, 67 lines, and 43 loads, which model the Nordic grid, see Figure 3. The model is part of the Open-Instance Power System Library (OpenIPSL), a Modelica library for power system dynamic analysis (Baudette et al., 2018).

The DAE representation (6) of Nordic 44, given by the symbolic transformations in Dymola 2019 FD01, consists of $n_x = 1013$ states and $n_G = 47$ torn algebraic loops. The first loop has $n_1 = 448$ iteration variables and the remaining have one each.

We will consider three different fault scenarios. Models for all of them have been added to the OpenIPSL library. They can also be found in the supplementary material to this article and at a dedicated GitHub repository[4]. The first two scenarios are reproductions of the experiments performed by (Vanfretti et al., 2016, Section 3). There, the second order Runge–Kutta scheme Rkfix2 was used with the fixed time step $h = 0.01$ s.

For the first scenario we introduce a line opening between Bus 5103 and Bus 5304 to occur at $t = 2$ s. The voltage for Bus 5304 is plotted in Figure 4, given as a result of the Dassl DAE mode simulation. To be able to

[4]GitHub: 2019_Modelica_Conf_DAESolvers4LargeHybridModels, https://github.com/ALSETLab/2019_Modelica_Conf_DAESolvers4LargeHybridModels

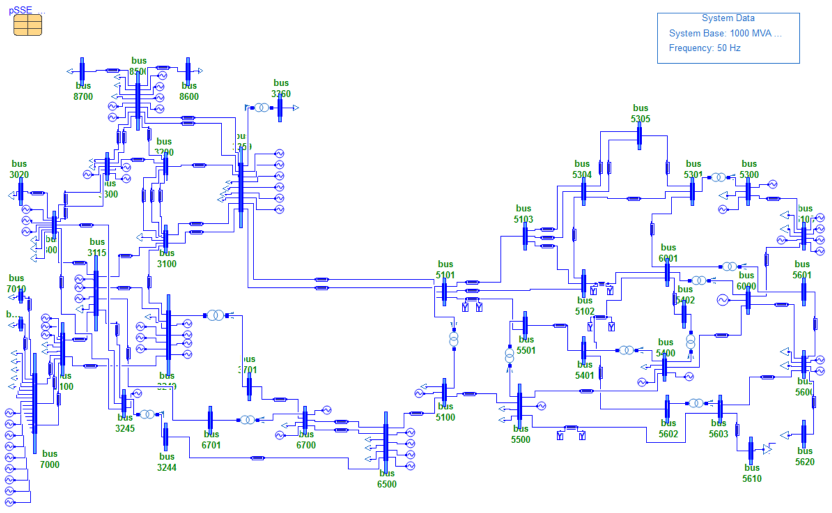


Figure 3. The Nordic 44 grid model.

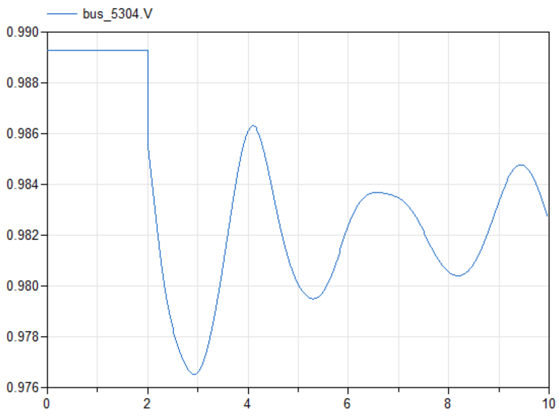


Figure 4. Voltage for Bus 5304 during a line fault between Bus 5103 and Bus 5304 occurring at $t = 2$ s.

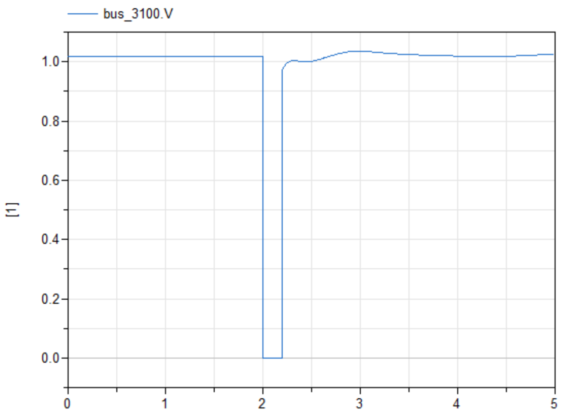


Figure 5. Voltage for Bus 3100 during a fault in this bus between $t = 2$ s and $t = 2.2$ s.

compare CPU-times with the Rkfix2 ODE mode simulations the tolerance 10^{-6} was chosen for Dassl. With this tolerance the error estimates of the two solvers are approximately the same.

In the second scenario a bus fault is instead simulated. At time $t = 2$ s Bus 3100 short-circuits and connects to the ground, with very small impedance, for 0.2 s. The simulated bus voltage is plotted in Figure 5. For this model, the Dassl tolerance 10^{-4} gives errors comparable to those of Rkfix2.

The final scenario also considers a bus fault, this time in Bus 5603. However, the model is also extended to include an additional generator connected to Bus 5610, see Figure 6. Compared to the other generators in the Nordic 44 model, a different excitation control system (IEEE Type AC2A Excitation System) is used. Depending on the generator field voltage, different control modes are used in the excitation system to change its outputs. To switch between the modes state events are required. This extended model has $n_x = 1313$ states and $n_G = 65$ algebraic loops of sizes $n_1 = 498$ and $n_2 = \cdots = n_{65} = 1$. Due to the extension the default Nordic 44 initial conditions do not define a steady state. To get close to steady state, a simulation is first performed until $t = 60$ s. Then, the bus fault occurs between $t = 61.05$ s and $t = 61.15$ s. The simulated generator field voltage (EFD) is plotted in Figure 7 together with the unmodified control output (EFD1). For comparable numerical errors the tolerance $5 \cdot 10^{-5}$ is used for Dassl.

4.2 Efficiency experiments

The CPU-times required to simulate the three different scenarios are listed in Table 1. All simulations in this section were run in Dymola 2019 FD01, with default settings if nothing else is specified, and using Visual Studio 2015 for model compilation. An ordinary Windows 7 (64-bit) laptop computer (Intel Core i7-6820HQ, 16 GB RAM) has been used for all experiments, including the reproduction of the Rkfix2 experiments ($h = 0.01$ s), reported by (Vanfretti et al., 2016). As mentioned above, the Dassl tolerances have been tuned to give comparable numerical errors between the two solvers.

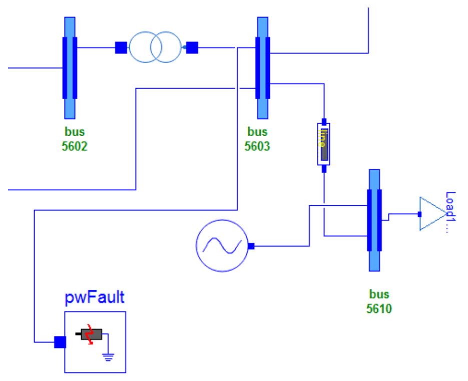


Figure 6. Nordic 44 extended with an extra generator featuring several control modes. The generator is connected to Bus 5610 and the fault occurs in Bus 5603.

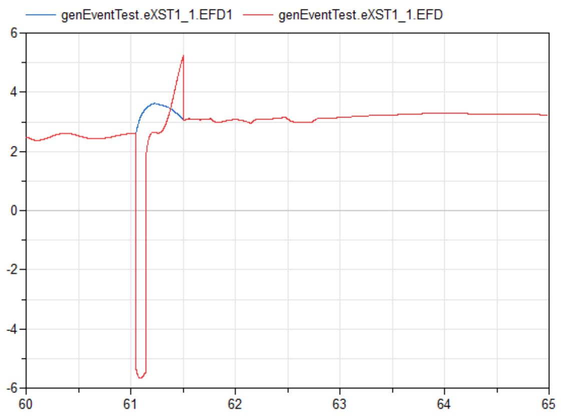


Figure 7. Generator field voltage (EFD) and the unmodified excitation control output (EFD1) for the extra generator at Bus 5610.

We observe that several orders of magnitude in simulation speed-up was gained by running Dassl in DAE mode; the construction of many expensive integrator Jacobians $\hat{J}$ makes the ODE mode times uncompetitive. We remind the reader that the Nordic 44 model has been specifically chosen as an example in this paper since it is difficult to handle efficiently in ODE mode. For most Modelica models ODE mode is more robust and as efficient. Finally, comparing with Rkfix2 we conclude that using an explicit method to altogether avoid integrator Jacobians does not pay off for these simulations.

All of the power grid faults are triggered at specific times, that is by time events. For the first scenario that is also the only event. This explains the very fast simulation time as the large algebraic loops only need to be solved during initialization and during the event iteration of the time event. In total each loop is solved only four times.

Table 1. CPU-times for the three Nordic 44 fault scenarios.

Fault	*Rkfix2*	*Dassl*	
	ODE mode	*ODE mode*	*DAE mode*
Line	587 s	2 015 s	4.21 s
Bus 3100	270 s	7 810 s	33.7 s
Bus 5603	344 s	49 800 s	121 s

However, the remaining two scenarios have several state events that must be located and resolved, namely 14 and 26, respectively. When the fault is triggered in Bus 3100, the excitation control system inside several of the generators reach their maximum limit. To prevent wind-up in the integral controllers their internal states are reset using state events. Since the desired set points cannot be reached the controllers continue to wind-up and several resets are required, cf. Figure 8. Even though reminiscent of Figure 2, the saw blade shape here represents a correct solution of the model. This can be easily verified by ODE mode simulations. Indeed, the fact that the accumulation of events is accurately handled when closing in on $t = 2.1$ s confirms the soundness of the event handling approach proposed by (Eich-Soellner and Führer, 2008) and used in this paper.

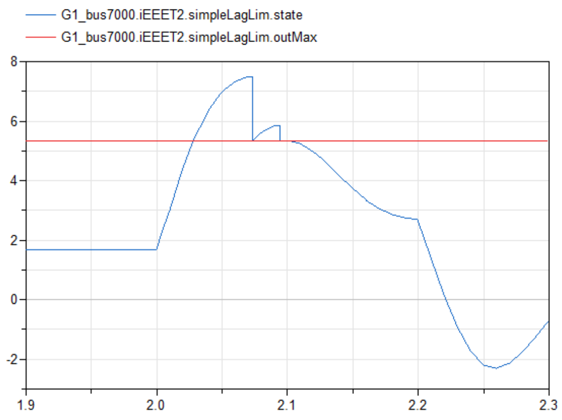


Figure 8. The fault in Bus 3100 triggers saturation in several generator excitation controllers. State events are issued to reset the internal controller state, as exemplified here with one of the generators connected to Bus 7000.

A similar analysis can be made of the third scenario with the extended Nordic 44 model. But there the state events instead represent switches between excitation control modes in the extra generator. Note that the higher number of state events and the larger algebraic loop are reflected by the longer simulation time.

As a final experiment we demonstrate the effect of one of the event handling optimizations presented in Section 3.3. Consider again the second scenario with the fault in Bus 3100. We rerun the Dassl DAE mode simulation, but during the simulation we monitor Equation (11) rather than Equation (12). This means that the algebraic loops (4) are solved throughout all of the simulation, not only when closing in on a zero crossing. This results in a run time of 197 s to be compared with 33.7 s for the efficient DAE mode implementation. In the former case the algebraic loops were solved 918 times, whereas in the latter case only 143 times.

Finally, we compare with the simulations performed by (Vanfretti et al., 2016) of equivalent models using the domain-specific simulation tool PSS/E. On a computer slightly faster than the one used for this paper the first two scenarios run in 5 s, respectively 4 s. *We conclude that the Dymola DAE mode performance is competitive against the industry state of the art.* In fact, even faster for the first scenario. At the same time, looking at the second scenario, we note that there is room for further efficiency improvements in the event handling.

5 Additional DAE mode challenges – an outlook

Other than the accurate handling of events, DAE mode simulations pose a few additional challenges not experienced to the same extent during ODE mode simulations.

5.1 Robustness and discontinuities

The nonlinear system of equations (8) to be solved inside the integrator is larger in DAE mode. Solving for both the states x and the algebraic variables z simultaneously may cause robustness problems. Compare with ODE mode where the algebraic equations (4) are solved one at a time, separate from each other, as part of each $\hat{f}$-evaluation. The nonlinear equation solvers that treat the algebraic loops may be optimized for this purpose. For example, when solving algebraic loops with only one iteration variable even major problems, such as a singular Jacobian, often do not pose an insurmountable threat. When this singularity becomes part of a large system of equations it becomes a problem that is much more difficult to handle.

When interpolated values are of interest, as when monitoring events, and the DAE is of index 1, it is recommended by (Brenan et al., 1996, Section 5.4.2) to apply error control also to the algebraic variables z. However, this may cause failed simulations when algebraic variables are discontinuous in time, consider e.g. van der Pol's equation

$$\begin{aligned} \dot{x} &= -z, \\ 0 &= x - \left(\frac{z^3}{3} - z\right), \end{aligned} \tag{13}$$

(Hairer and Wanner, 1996, Section VI.1). Discontinuities may also arise when using the `noEvent` operator.

5.2 ODE-powered DAE mode simulation

With the strategy of using the same generated simulation code both in ODE and DAE mode an opportunity opens up to handle these DAE mode specific problems. The idea is simple and based on the fact that we can readily switch between the modes: we integrate in DAE mode until a problem is encountered. Then we switch to ODE mode and integrate past the problem. When it is deemed fine to continue in DAE mode, the switch back is made.

Note that switching to ODE mode comes with a considerable cost. In contrast to the event handling strategy previously discussed, we here want to perform continuous integration in ODE mode. This requires the construction of ODE Jacobians $\hat{J}$, which is expensive. An important goal for any implementation of this idea is probably to keep the number of ODE Jacobians to a minimum. If additionally the rest of the simulation runs smoothly in DAE mode it may still be considerably more efficient than plain ODE mode simulation.

We have made a prototype implementation of this idea using Dassl. The algorithm is simple: when the integrator gives up in DAE mode we do not stop the simulation, but rather, switch to ODE mode. While in ODE mode we allow for one Jacobian computation and simulate with this until Dassl asks for a second. Instead of computing it, we switch back to DAE mode and continue simulation until the integrator gives up again or the simulation terminates.

Example 2. As discussed above, when simulating van der Pol's equation (13) in DAE mode it normally fails when closing in on a discontinuity in z. The error estimate in this variable becomes large and cannot be made smaller by shorter time steps. However, using the prototype implementation introduced here we can successfully simulate past the discontinuities by temporarily switching to ODE mode, see Figure 9. For this simulation Dassl required 132 Jacobian-evaluations in DAE mode and only two Jacobian-evaluations in ODE mode. △

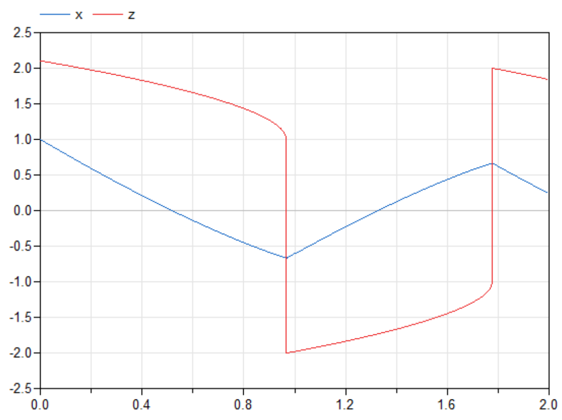


Figure 9. DAE mode solution of van der Pol's equation (13) using temporary switches to ODE mode to handle the discontinuities in z.

Of course, when simulating the van der Pol's equation there is no efficiency benefit in using DAE mode. However, for a production-level implementation that can handle large-scale models, the prototype implementation needs several improvements and tuning. The most important question is to decide when to switch, especially when to switch back to DAE mode. The simple prototype implementation presented above will often switch back too early. A more careful analysis of the state of the problem should probably be performed before switching back. Further questions involve how to best reinitialize the simulation and with which step size.

6 Conclusion

We have discussed and implemented a DAE mode strategy for hybrid DAEs based on the idea of one common code generation for ODE and DAE mode. By applying all of the symbolic transformations and optimizations we get an index-1 DAE of minimal size. The integrator only needs to handle the original states and the iteration (tearing) variables of the algebraic loops. Even though a smaller system is typically faster to simulate the tearing algorithm may cause fill-in making the reduced system more dense. How to best apply tearing in this context deserves deeper analysis but is out of scope of this paper.

Our DAE mode approach made it possible to accurately and efficiently handle state events, with minimal footprint on the generated code. To localize these events we have extended from results known in the literature to fit our Modelica context. Suggestions for optimizing the root finding were also discussed and implemented.

With these optimizations the algebraic loops only need to be solved with high accuracy during initialization, when closing in on a state event, when localizing it, and when resolving it. Most importantly, solving the loops is not required when constructing the integrator Jacobian or other evaluations of the dynamics. Typically, only on the order of ten solutions per state event are required. We argue that this is an acceptable cost for models with a moderate number of state events. For example, compare with the large number of loop solutions that are required to just construct the integrator Jacobian in ODE mode, cf. Section 2.2.

Therefore, DAE mode simulations can be vastly more efficient for models with large algebraic loops. As exemplified by simulations of the Nordic 44 model of the Nordic power grid, where orders of magnitude in simulation speed were gained. The measured simulation times are competitive with domain-specific, state-of-the-art simulation tools that have been optimized for more than three decades.

The presented DAE mode was made available in Dymola 2019 and 3DEXPERIENCE 2019x for a broad selection of numerical integrators. The implementation also features detailed DAE mode sparsity pattern analysis and is fully compatible with Dymola sparse solvers.

Acknowledgments

The authors thank Ricardo Rincon Ballesteros (Universidad Nacional de Colombia) for constructing the third simulation scenario of the application section specifically for this publication.

The authors also thank their colleagues for valuable feedback on the paper drafts.

The work of L. Vanfretti was supported in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877, and in part by the CURENT Industry Partnership Program.

References

Maxime Baudette, Marcelo Castro, Tin Rabuzin, Jan Lavenius, Tetiana Bogodorova, and Luigi Vanfretti. OpenIPSL: Open-instance power system library — update 1.5 to “iTesla power systems library (iPSL): A Modelica library for phasor time-domain simulations”. *SoftwareX*, 7:34–36, 2018.

Willi Braun, Francesco Casella, and Bernhard Bachmann. 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, 2017.

Kathryn E. Brenan, Stephen L. Campbell, and Linda R. Petzold. *Numerical Solution of Initial-Value Problems in Differential–Algebraic Equations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1996.

Francesco Casella. Simulation of large-scale models in Modelica: State of the art and future perspectives. In *Proceedings of the 11th International Modelica Conference*, pages 459–468. Linköping University Electronic Press, 2015.

Francesco Casella, Andrea Bartolini, Simone Pasquini, and Luca Bonuglia. Object-oriented modelling and simulation of large-scale electrical power systems using Modelica: A first feasibility study. In *IECON 2016 – 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 6298–6304, 2016.

Francois E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer-Verlag, Berlin, Heidelberg, 2006.

Edda Eich-Soellner and Claus Führer. *Numerical methods in multibody dynamics*. Teubner, 2008.

Hilding Elmqvist and Martin Otter. Methods for tearing systems of equations in object-oriented modeling. *ESM’94 European Simulation Multiconference*, pages 326–332, 1994.

Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Parallel model execution on many cores. In *Proceedings of the 10th International Modelica Conference*, pages 363–370. Linköping University Electronic Press, 2014.

Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential–Algebraic Problems*, volume 14. Springer-Verlag Berlin Heidelberg, second edition, 1996.

Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, pages 363–396, 2005.

Filip Jorissen, Michael Wetter, and Lieve Helsen. Simulation speed analysis and improvements of Modelica models for building energy simulation. In *Proceedings of the 11th International Modelica Conference*, pages 59–69. Linköping University Electronic Press, 2015.

Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software*, 31(3):302–325, 2005.

Fredrik Magnusson. *Numerical and Symbolic Methods for Dynamic Optimization*. PhD thesis, Department of Automatic Control, Lund University, 2016.

Hans Olsson, editor. *Modelica – A Unified Object-Oriented Language For System Modeling: Language Specification*. Modelica Association, 2017. Version 3.4.

Hans Olsson, Sven Erik Mattsson, Martin Otter, Andreas Pfeiffer, Christoff Bürger, and Dan Henriksson. Model-based embedded control using Rosenbrock integration methods. In *Proceedings of the 12th International Modelica Conference*, pages 517–526. Linköping University Electronic Press, 2017.

Anton Schiela and Hans Olsson. Mixed-mode integration for real-time simulation. *Proceedings of Modelica 2000 Workshop*, pages 69–75, 2000.

Bernhard Thiele, Martin Otter, and Sven Erik Mattsson. Modular multi-rate and multi-method real-time simulation. In *Proceedings of the 10th International Modelica Conference*, pages 381–393. Linköping University Electronic Press, 2014.

Luigi Vanfretti, Tin Rabuzin, Maxime Baudette, and Mohammed Murad. iTesla power systems library (iPSL): A Modelica library for phasor time-domain simulations. *SoftwareX*, 5:84–88, 2016.

Luigi Vanfretti, Svein H. Olsen, V.S. Narasimham Arava, Giuseppe Laera, Ali Bidadfar, Tin Rabuzin, Sigurd H. Jakobsen, Jan Lavenius, Maxime Baudette, and Francisco J. Gómez-López. An open data repository and a data processing software toolset of an equivalent Nordic grid model matched to historical electricity market data. *Data in Brief*, 11:349–357, 2017.

Adaptive Step Size Control for Hybrid CT Simulation without Rollback

Rebeka Farkas[1,2,3] Gábor Bergmann[1,2,3] Ákos Horváth[1,3]

[1]Department of Measurement and Information Systems, Budapest University of Technology and Economics, Hungary, {farkasr,ahorvath,bergmann}@mit.bme.hu
[2]MTA-BME Lendület Cyber-Physical Systems Research Group
[3]IncQuery Labs Ltd, Hungary

Abstract

The *Hybrid CT* approach for simulating cyber-physical systems uses *continuous time* simulation and provides *wrappers* for discrete event components that implement the required interfaces. Besides the general obstacles of continuous time simulation, Hybrid CT introduces new challenges, such as creating wrappers, detecting discrete events (with minimal latency), and finding the correct balance between the simulation step sizes required by different components.

We propose an adaptive step size controller that uses high level information of the model and the simulation (e.g. types of components, critical values of variables) to adjust the step size based on the possibility of the detection of a discrete event in the following step. Besides overcoming the challenges of Hybrid CT simulation the component also improves threshold-crossing detection. The proposed approach does not require step rejection (rollback), that discrete event components often fail to support.

In this paper we present the step size controller, demonstrate its usability on industrial case studies and evaluate the component both theoretically and based on measurements performed on our implementation that was integrated to the OMSimulator. We show that adaptive step size control can be used to bridge the gap between continuous time and discrete event simulation.
Keywords: hybrid CT simulation, step size control

1 Introduction

Hybrid systems demonstrate both discrete and continuous behaviour which makes their simulation challenging. A possible approach is *Hybrid CT* that uses *continuous time* simulation and provides wrappers for discrete event components (as opposed to *Hybrid DE* simulation where continuous time components are adjusted so discrete event simulation can be used).

The OMSimulator developed by the Open Source Modelica Consortium (OSMC) uses Hybrid CT simulation based on the Functional Mock-up Interface (FMI) standard (Blochwitz et al., 2012) that defines a centralized architecture for simulation, where each component of the system (the so-called Functional Mock-up Units, FMUs) is simulated on its own with a *master simulator* controlling the process.

Despite the fact that there are proper approaches to create FMUs from discrete event components, the co-simulation of continuous-time and discrete-event blocks is still in its early phases. From a simulation point of view, one of the main differences between the two types of components is the simulation step size: continuous systems are simulated by periodically calculating the value of the variables with relatively large step sizes (measured in seconds) but discrete event-based systems operate irregularly and their simulation requires smaller steps (measured in nanoseconds) since discrete events can trigger other discrete events (almost) instantly. It is possible to simulate continuous-time models with smaller step sizes (in fact, it yields more accurate results), but it is inefficient (often preventing industrial application) and mostly unnecessary as events occur rarely. The sporadic occurrence of discrete events raises the need for *adaptive step size control*.

We propose an adaptive step size control approach to overcome the difficulties of hybrid CT simulation. The proposed solution requires the user to select the variables that are used to model event-based behaviour and adjusts the step size when their values change. Moreover, our approach can also be used to increase the accuracy of *threshold-crossing detection* and location (which is an important aspect of hybrid simulation) without the need for rejecting steps (rollbacks).

This paper is organized as follows: section 2 presents some background knowledge on the challenges of Hybrid CT simulation, section 3 lists the related work, section 4 presents the proposed step size controller, section 5 demonstrates its applicability, section 6 evaluates its usefulness and section 7 concludes the paper.

2 Preliminaries

2.1 Running example: Thermostat

In this paper we use an advanced version of the commonly used *thermostat* example to illustrate the presented concepts. The thermostat example describes a room with a thermostat keeping the temperature near some target temperature (with a given tolerance) that is given by a user and can change through time. In addition we introduce a monitoring system to ensure that the thermostat operates correctly.

The monitoring system consists of three local monitors and each of them is responsible for a component of the heating system: the heating component, the heat sensor, and the thermostat. The monitoring system is controlled by a central monitor that communicates with the local monitors via messages to check wether the system is working correctly. Such a check is performed periodically every three minutes and each time before turning the heating on.

In the simulated scenario, the temperature initiates from 20°C with the target temperature set to 22°C. Originally the hysteresis is 3°C, but set to 2.5°C after half an hour. The temperature of the environment is 0°C – that is, when the heating system is not activated the temperature of the room is decreasing towards 0°C. Initially, the thermostat is turned off but it is turned on after 10 seconds.

2.2 FMI-based hybrid co-simulation

The FMI standard for co-simulation makes it possible to simulate a system containing various components described by different types of models that require different ways of simulation. Co-simulation requires that each model is encapsulated with an appropriate simulator making an FMU which implements an interface through which the *master simulator* can control the simulation. In addition, the FMU also contains an XML-based model description file, with high level information about the model and additional information, such as the *DefaultExperiment* element that contains the default values of basic simulation parameters (e.g. stop time, relative tolerance, step size).

The modular architecture of FMI-based simulation makes it appropriate for hybrid co-simulation where the two types of components are the discrete event and the countinuous time components. Additionally, neither the model nor the simulator internal data need to be accessed, which makes FMI-based co-simulation industrially applicable.

OMSimulator is an FMI-based simulator for cyber-physical systems, developed by the OMSC. In order to simulate, the architecture must be defined (input and output ports must be coupled) and configuration data must be provided, e.g. simulation step size, tolerance, duration (the values given by the *DefaultExperiment* may differ in each FMUs). After initialization the simulation is performed by alternating two types of steps: in a *simulation step* the master simulator instructs all FMUs to perform a step of the given step size, and in a *communication step* the output values of the source components of the connections are passed as the input values of the corresponding target components. Simulation terminates after a given duration.

Example In case of the thermostat example, we created the following FMUs.

- The *Thermostat* FMU contains a discrete event-based model describing the operation of the thermostat. The inputs of the thermostat include the settings and the current temperature.

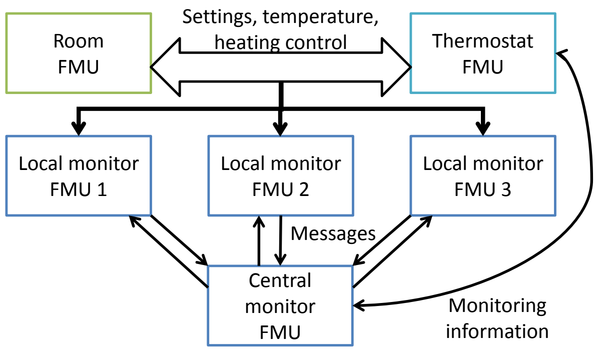


Figure 1. Thermostat FMU architecture

- The *Room* FMU contains a continuous model describing the characteristics of the physical world, including the temperature and the user that provides the settings of the thermostat. The user operations are pre-defined and the temperature is calculated based on the heating.

- The (discrete event-based) models of the each monitors are provided in separate FMUs. The monitors get the same inputs as the thermostat and check if the operations of the thermostat are correct.

Accordingly the *Thermostat* and the *Room* FMUs are connected, and the monitors are connected to both both of them are connected to each monitors. The complete architecture can be seen in Figure 1.

2.3 Simulation challenges

One of the most important requirement of simulation is *accuracy*: while it is theoretically impossible to calculate accurate values, there are many ways to calculate an over-approximation of the error and to keep it below a given amount (Viel, 2014; Arnold et al., 2014a,b). The other important requirement is *efficiency* and – as usual – the two requirements contradict.

In case of iterative methods accuracy can be improved by increasing the number of iterations during a simulation step. In practice the iterations required to comply with the desired tolerance can result in an impermissibly large runtime which makes non-iterative methods favoured in case of co-simulation.

In case of non-iterative methods efficiency depends on the number of steps performed (hence, larger step size yields more efficient simulation) and accuracy depends on the size of the steps (smaller step size yields more accurate results). As an optimization, master algorithms often use *rollbacks* (the rejection of one or more steps) when the simulation error is above the tolerance and then re-simulate with smaller step sizes. Rollbacks have other advantages, e.g. in case of the so-called *threshold-crossing detection* problem, where it has to be detected (and located) when a given variable reaches a certain value.

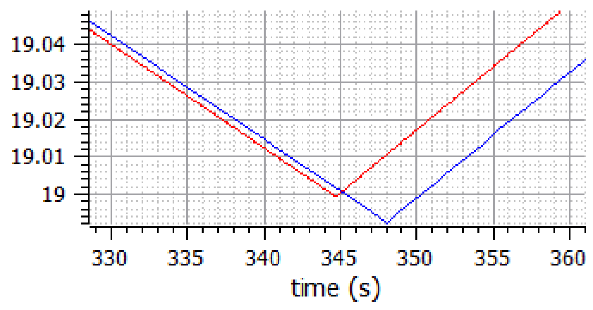


Figure 2. Simulation error caused by step size

Introducing discrete event components into a continuous time environment raises a new challenge, as it becomes more important to detect events with minimal latency. While this problem can also be solved with rollbacks (Galtier et al., 2015) many discrete event FMUs can not handle rollbacks and therefore the only way to ensure the events are detected in time is to use smaller step sizes that makes simulation intractably expensive. It is also possible to attempt to *predict* events (Guermazi et al., 2016).

Example: In case of the thermostat example the *detection of discrete events* becomes important with the monitoring system: when the central monitor gathers information to check if the thermostat works correctly, messages are passed between the monitors. This process is fast and in order to simulate it accurately, the simulation step sizes have to be small, as a number of discrete events (the messages) are triggered by each other, therefore at most one of these events occur in each simulation step. This communication between the monitors takes place when the heating has to be turned on as well as every three minutes when the periodical checks are performed. The former scenario introduces a *threshold-crossing detection* challenge: it is important to detect when the temperature exceeds the bounds of the target interval (initially 19°C and 25°C), and the periodical checks raise the need for *event prediction*.

In order to demonstrate the importance of keeping the latency minimal, we have simulated the first time the temperature falls below 19°C in the thermostat example (excluding the monitor components) with constant step sizes of 0.1 s and 0.01 s. The threshold-crossing happens about 344 seconds from initialization. The results are shown in Figure 2, where the x axis represents the (simulated) time and the y axis represents the simulated temperature. The red line denotes the results of simulation with the smaller step size and blue line corresponds to the larger step size. Before the threshold-crossing the two simulation produces similar results – with an exception of an initialization offset (explained in subsection 6.2). However, after the temperature decreases below 19°C the two simulations produce significantly different results. The reason behind this is that because of the larger step sizes both the threshold-crossing and the discrete reactions are detected with latency. Because of this, in case of the smaller step size the temperature raises over 19°C in less than a second, while in case of the other simulation it takes almost five seconds which is a simulation error caused purely by event detection latency introduced by the large step sizes.

3 Related work

Discrete components in simulation There is extensive existing research on introducing discrete event components to continuous time environments. In (Guermazi et al., 2016) the discrete event components are integrated in the continuous time simulation workflow as a white box, and the communication intervals are adjusted to detect events based on internal information. In (Galtier et al., 2015) rollbacks are used: when an event is detected, the last step is rejected and the simulation step size is adjusted to the minimum amount to locate it. In (Franke et al., 2017) discrete time simulation is supported by introducing *clocks* and corresponding *clocked variables* that only have values when the clock ticks.

Step size control There are many adaptive step size control approaches for enhancing the performance of the simulation (Schierz et al., 2012; Viel, 2014), but most of them rely on internal data and step rejection, with the exception of (Busch and Schweizer, 2011) that uses a non-iterative predictor/corrector error estimator. Adaptive step size control can also be used for threshold-crossing detection and location (Esposito et al., 2001) .

Since the presented algorithms all focus on finding the largest possible step size, it is theoretically possible to combine them.

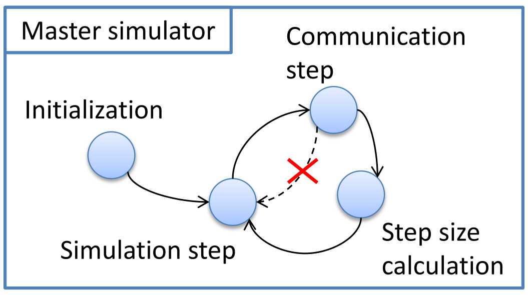
4 Adaptive step size control for Hybrid CT simulation

4.1 Overview of the approach

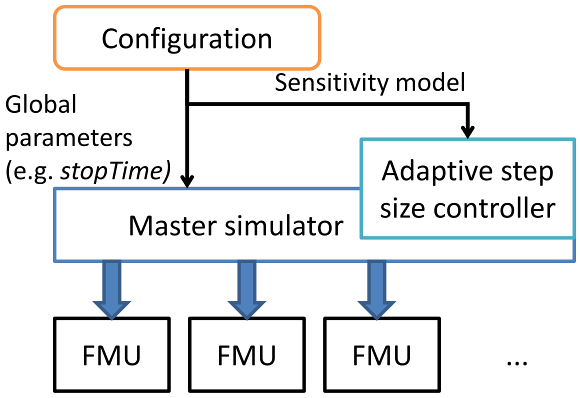
The approach is illustrated in Figure 3.

General idea The step size controller unit is a component of the master simulator, invoked immediately before performing a simulation step, as depicted in Figure 3a. The size of the next simulation step is calculated based on a number of influencing factors, such as values of variables getting near to thresholds, expected occurrence of events, expected event-responses, etc. These parameters are pre-defined in a data structure that we call the *sensitivity model*, that can be considered a configuration parameter of the simulation. Afterwards, the simulation step is performed with the calculated step size. The simulation step is followed by a communication step which is followed by the step size calculation preparing the next simulation step and so on.

Architecture The step size controller can be a component of the master simulator or an additional layer controlling it. The required sensitivity model is an input of



(a) Master simulation algorithm


Configuration
Sensitivity model
Global parameters (e.g. *stopTime)*
Adaptive step size controller
Master simulator
FMU
FMU
FMU
...

(b) Architecture

Figure 3. Overview of approach

the component (or the simulator) – in the current implementation the complete sensitivity model has to be provided together, altough it would be more practical to provide the FMU-specific elements individually, encapsulated with the corresponding FMUs (see subsection 6.2). The architecture can be seen in Figure 3b.

Information to provide In case of FMI-based co-simulation of a large-scale system integration project, the FMUs can originate from different stakeholders who may wish to safeguard their intellectual property. While the sensitivity model does require some information on the operation, this information could also be derived by conducting a limited number of preliminary simulation runs with large (fix) step sizes (see subsection 5.2). Therefore the sensitivity model is a convenient compromise between making the models public in order to simulate them accurately and hiding the models and simulate with impractically small step sizes.

4.2 Sensitivity model

The sensitivity model parametrizes the adaptive step size control approach. It describes the critical scenarios that require accurate calculations or detecting discrete events with low latency – that is, the scenarios where it is important to set the step size small.

In case of FMI-based co-simulation the numerical accuracy is ensured by the internal simulators of the FMUs, but in order to detect a discrete event precisely, the event has to occur at the last moment of the simulation step, otherwise the event is detected with latency.

4.2.1 Described scenarios

The sensitivity model was created to describe various scenarios where the step size need to be adjusted. The data structure of the proposed sensitivity model can be seen in Figure 4. The described scenarios can be classified as follows.

Event reactions Discrete events may trigger other discrete events that have to be simulated with minimal latency. In order to identify these scenarios, a set of *event indicators* – i.e. variables where the change of the value represent an event – have to be declared. During simulation when an event is detected, the step size is set to minimal, so that the reactions can be simulated accurately.

Example: In order to avoid the latencies during the sequences of discrete events during the simulation of the thermostat, all variables representing messages should be included in the set of event indicators. However, this solution does not prevent the latency in the detection of the first message.

Timed events Discrete events may be triggered by the elapse of time. In order to perform a simulation step so that the discrete event is simulated without significant latency, they have to be predicted - i.e. the sensitivity model has to store when to expect an event. A set of variables, called *time indicators* can be given that each indicate when an event will be fired. The values of the variables can be changed during simulation so periodical events only require one indicator variable.

Example: It can be predicted when the central monitor initiates the periodical check based on the variable the component uses for timing the first message.

Threshold-crossing Discrete events may be triggered by continuous variables crossing a given threshold. In order to facilitate the detection of such scenarios with minimal latency the sensitivity model allows the description of auxiliary *threshold intervals* for the variable with corresponding step sizes. Intuitively, the auxiliary intervals should describe when the value is *close* to the threshold and a corresponding step size should be small enough to detect it. Accordingly, the step size controller does not guarantee to precisely detect when a value of a signal gets inside an interval (hence the interval should be appropriately large) but as soon as it is detected, the step size is adjusted accordingly. This way if the limits of the auxiliary interval are appropriate considering the behaviour of the system, the threshold-crossing can be detected with low latency.

Example: In case of the threshold-crossing depicted in Figure 2 the temperature decreases less than 0.003°C in a second, therefore any upper bound over 19.003°C and lower bound below 19°C guarantees that a simulation with a step size of 1 second will surely include a communication step where the temperature is in the interval but more than

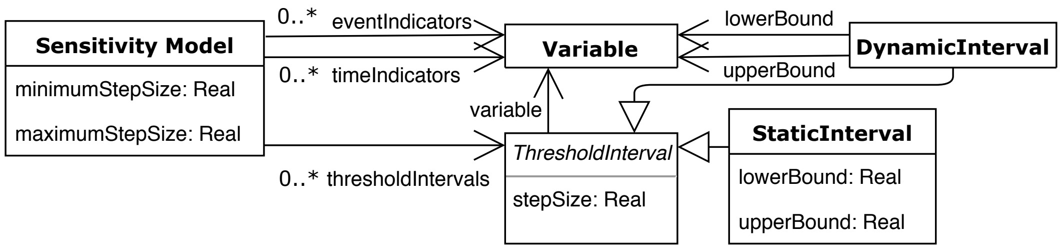


Figure 4. Sensitivity model data structure

19°C. If the step size corresponding to the threshold interval is 0.1 second, then by the time the target threshold-crossing happens the step size will be set to 0.1 second and the threshold-crossing will be detected with a smaller latency. (Naturally, choosing the appropriate upper bound *before* simulation is more difficult and requires domain knowledge.)

4.2.2 Dynamic parameters

Parameters of the model (execution of timed events, thresholds) can be declared both statically and by assigning a variable of the FMU whose value determines the current value of the parameter – the latter can be useful e.g. for declaring the next occurrence of a timed event or when the important threshold to cross can change through time.

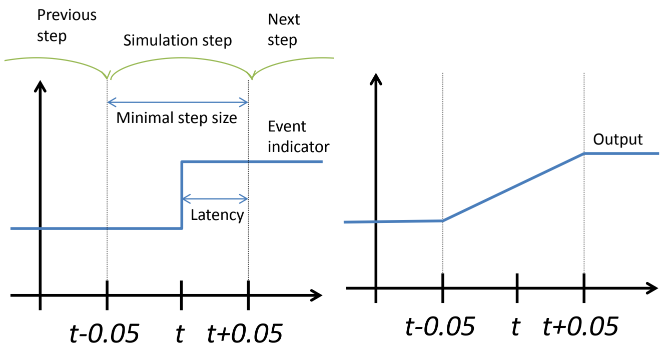
Example: The constant threshold-interval in the previous example only facilitates the threshold-crossing detection until the hysteresis changes. In order to create a general solution, additional variables have to be introduced to the model, e.g. instead of the constant upper bound 19.003°C an additional variable v_{add} can be used with the value $v_{\text{add}} = v_{\text{trg}} - v_{\text{hys}} + 0.01$°C where v_{trg} is the target temperature v_{hys} is the hysteresis and the constant was increased to ensure the interval is large enough.

As demonstrated by the example, additional variables often have to be created in order to describe dynamic parameters. While it is not always possible to modify the FMUs since they are generated, but it is always possible to create an additional FMU with the additional variables that takes the outputs of other FMUs as inputs.

It is important to mention that the sensitivity model describes *expected* scenarios – it does not make a difference during simulation whether the expected events are actually detected, which makes it applicable for non-deterministic models. However, the unnecessary adjustments cause the simulation to be less efficient than it could be if the sensitivity model was more precise.

4.2.3 Minimal and maximal values

As a reference, the minimum and the maximum value of the step size has to be declared before the simulation. It is guaranteed that during simulation the step size will always be between the minimum and the maximum value (except for the very last step that may be smaller than the lower bound). Generally, the main reason for the lower



(a) Event **(b)** Event detection

Figure 5. Event detection latency example

bound is to avoid Zeno behaviour that could otherwise be easy to cause (e.g. it is possible to schedule a sequence of timed events, always with half as much delay as the last one). However, in case of Hybrid CT simulation the lower bound is the guaranteed maximal delay of detecting a timed event, as well as the guaranteed delay between a sequence of discrete events triggered by each other. The upper bound becomes significant when there is no discrete event to expect in the next step – in this case, the step size is set to the given maximum value.

Example In order to demonstrate the significance of the minimal step size, consider the periodical check performed by the central monitor. Let us suppose the next check is scheduled at time stamp t however, because of an active threshold-interval, the current step size is 0.15 s, the simulation time after the last step is $t - 0.05$ s and the minimal possible step size is 0.1 s. Since the size of the next simulation step has to be at least 0.1 s the the event will be detected at $t + 0.05$ s. The latency is illustrated in Figure 5.

The event initiates a sequence of discrete events (responses) and if the event indicators are defined appropriately in the sensitivity model then the step size stays at the minimal value during the simulation of the event sequence.

4.2.4 IP protection concerns

In an industrial environment it is possible that the model constitutes confidential intellectual property and using the step size controller would require disclosing some of the restricted information in the sensitivity model.

In this case the step size controller has to be used differently: in order to protect intellectual property, the model needs to be modified so that the critical scenarios are identified within the FMU. This can be achieved using an auxiliary variable representing the critical scenarios and creating corresponding threshold intervals in the sensitivity model.

Example: Suppose there is a critical scenario that must be simulated precisely. Instead of providing a detailed sensitivity model, an auxiliary variable v can be used to identify the scenario – e.g. $v = 1$ during the scenario and $v = 0$ otherwise. The corresponding interval can be $0.5 \leq v \leq 1.5$ and the corresponding step size can be the minimal value. This way, when the critical scenario is detected (within the FMU) v is set to 1 and the step size controller adjusts the step size to the minimal value.

4.3 Algorithm

In order to calculate the size of the simulation steps, the latest values of event indicators have to be stored. After a communication step, the size of the next simulation step is calculated based on the detected events, the scheduled timed events and the threshold-crossing intervals. Each element of the sensitivity model defines an upper bound on the size of the next step (including the global maximum step size), therefore the actual value of the next step should be the minimum of the given upper bounds (unless it is smaller than the possible minimal value).

An event can be detected by checking if the current value of the corresponding event indicator differs from its previous value. When an event is detected the step size has to be set to the minimal possible value. The stored (previous) values of event indicators always have to be updated, but when an event is detected, no additional calculations are necessary as the step size will certainly be set to the minimal possible amount.

In case when no event is detected the time indicators and the threshold intervals have to be checked. If a value of a time indicator is less than the current simulation time t, it is irrelevant. The time indicator with the smallest value $t_{\text{min}} > t$ denotes the next possible timed event. In order to detect the event precisely the next step can not exceed the difference $t_{\text{min}} - t$, except if it is less than the defined minimum.

The upper bound on the next step based on the threshold intervals can be derived by checking the corresponding variables – if the value of a variable is within the bounds of one of its corresponding auxiliary intervals, the next step can not exceed the corresponding step size described in the sensitivity model.

Example: Let us demonstrate the simulation using adaptive step size control on the thermostat model with the sensitivity model describing the scenarios discussed before and the minimal step size set to 0.01 s and the maximal step size set to 10 seconds. (The bounds of the step size are intentionally unrealistic to demonstrate the operation of the step size control.) The intervals for the threshold crossings are set so that when the difference between the current threshold is less than 0.1°C, the step size is set to 1.0 s and when it is less than 0.02°C, the step size is set to 0.1 second. The model is simulated for 400 seconds. The step sizes are shown in Table 1.

Table 1. Adaptive step sizes of the Thermostat

Start time	Scenario	Size	Steps
0.00 s	Default step size	10.00	18
180.00 s	Message sequence	0.01	8
180.08 s	Default step size	10.00	13
310.08 s	Temp. below 19.10°C	1.00	28
338.08 s	Temp. below 19.02°C	0.10	66
344.68 s	Message sequence	0.01	9
344.77 s	Temp. below 19.02°C	0.10	162
350.97 s	Temp. below 19.10°C	1.00	9
359.97 s	Timed event at 360 s	0.03	1
360.00 s	Message sequence	0.01	8
360.08 s	Temp. below 19.10°C	0.01	15
375.08 s	Default step size	10.00	2
395.08 s	Simulation ends at 400 s	4.92	1

The simulation begins with the maximal step size. The first periodical check happens to be scheduled exactly at the end of the 18th step, therefore, the step size does not have to be adjusted. However, as soon as the first message is detected, the sequence of discrete events caused by the messages is simulated with minimal latency.

In case of the checks caused by the temperature decreasing below 19°C, the first discrete event – the state transition of the thermostat – is detected with (relatively) high latency: the current step size to the corresponding interval, namely 0.1 s. After that, the monitors send the same eight messages which is why there is one more event in this sequence, than in the ones representing the messages passed during the timed checks.

In conclusion the simulation takes 340 steps, which is 99% less than what it takes to simulate it with constant step size of 0.1 s (which would take 40 000 steps) but the results are more precise.

5 Experimental evaluation

We have integrated the proposed adaptive step size controller component in the OMSimulator[1] and run measurements on case studies with constant steps of different step sizes as well as using adaptive step size control. In order to find out how the step size controller affects performance we measured the runtime and then analysed the differences between the efficiency and the results of corresponding simulations.

5.1 Thermostat

The sensitivity model for the Thermostat was presented in section 4. We performed the simulation with various constant step sizes as well as with the step size controller.

[1]https://github.com/OpenModelica/OMSimulator

Table 2. Simulation performance of the Thermostat example

Size	Steps	Runtime	Min temp	Max temp
0.01	300 000	150.94 s	18.9997°C	24.5000°C
0.10	30 000	15.32 s	18.9975°C	24.5004°C
1.00	3 000	1.49 s	18.9734°C	24.5047°C
10.00	300	0.16 s	18.7387°C	24.5356°C
*	808	0.46 s	18.9990°C	24.5005°C

Each time 3000 seconds were simulated. The results are shown in Table 2. The columns of the table contain the step size, the number of steps performed, the runtime of the simulation, and the minimal and the maximal simulated temperature (respectively). The * in the first cell of the last row represents that the simulation includes various step sizes, as chosen by our adaptive algorithm. The simulation with adaptive step size control resulted in 144 steps of 0.01 s, 282 steps of 0.1 s, 93 steps of 1 s, 275 steps of 10 s and 13 steps of other sizes.

In the simulated scenario, the target temperature is 22°C with initially 3°C tolerance (which is why the minimal value is just below 19°C) that is later set to 2.5°C by the user (which is why the maximal value is just above 24.5°C). The results show, that – in case of constant step sizes – an order of magnitude difference in the step sizes yields an order of magnitude difference between the expected and the simulated extreme values. However, in case of adaptive step size control, the difference is almost as small as in case of the 0.01 s steps while the simulation was almost as fast as in case of the 10 s steps.

5.2 Sherpa Automotive demonstrator

The Sherpa Automotive demonstrator is one of the industrial models in the OpenCPS project[2] that served as a basis for required improvements of the simulator.

The case study contains the models representing the mobility aspects of the system presented in (Mokukcu et al., 2017). The physical aspects of a hybrid electric vehicle are simulated to determine the amount of energy required for the vehicle to move with the required speed. The simulated scenario is 1200 time units and the default step size is 0.01. We have performed the simulation with constant step sizes of 0.1 and 0.01. The simulated speed of the vehicle is depicted in Figure 6. Up to 800 time units the results are similar: the largest difference between the result of the simulation with the small (red) and the large (blue) step size is less than 0.55. The biggest differences appear near to the local minimum and local maximum values. In the remaining part of the simulation the differences are much larger (14.0) and additional high frequency sine components appear. The unstable oscillating waveforms show that a step size of 0.1 is too large for accurate simulation.

The goal of adaptive step size control is to improve simulation performance by using larger step sizes where it does not affect precision. After studying the results we have created a simple sensitivity model: braking is considered a discrete event, therefore the the signal representing the brake position is used as an event indicator, and in order to avoid the additional sine components, threshold intervals are used on the variable representing the target speed.

We run the simulation, with 0.1 as maximal and 0.01 as minimal step size three times with the event indicator and the intervals individually and combined. The results can be seen in Figure 7. The runtimes and an evaluation of all performed simulations are shown in Table 3. The *Diff* column of the table denotes the maximum difference between the simulated speed of the vehicle and the speed simulated by the default simulation and *Diff2* denotes the maximal difference in the first 800 s of simulation time.

Using the brake as event indicators causes a runtime almost 90% smaller than that of the original simulation with step size 0.01 while reducing the error of the large step size simulation: in the first part the difference remained under 0.55 and the unstable oscillations of the results disappear decreasing the biggest difference to 2.8. However, some additional low frequency sine waveforms can still be seen.

Using the intervals (without event indicators) causes a runtime 40% smaller than the simulation with the small step sizes and reduces the error of the simulation with the large step sizes: in the first part the difference is less than 0.55 and the obscure parts of the result disappear decreasing the maximum of the difference to 0.7. However, one additional sine waveform remains.

Using both elements of the sensitivity model combines the advantages in accuracy as the results of the simulation are almost the same as that of the simulation with the small step size. However, the resulting runtime is larger than that of the simpler sensitivity models (though not as large as the runtime with a fixed small step size), since in each step the step size is the minimum of those in the previous cases.

The results show, that using the brake as event indicator increased the performance drastically while only introducing small simulation errors. The case study also demonstrated that intervals can be used to identify the critical scenarios.

6 Discussion

6.1 Limitations and opportunities

As demonstrated in section 5 adaptive step size control can be used to improve simulation performance. Step size control has been shown to be effective for event detection as well as decreasing runtime while preserving numerical accuracy.

[2] ITEA3, OpenCPS: Open Cyber-Physical System Model-Driven Certified Development http://www.opencps.eu

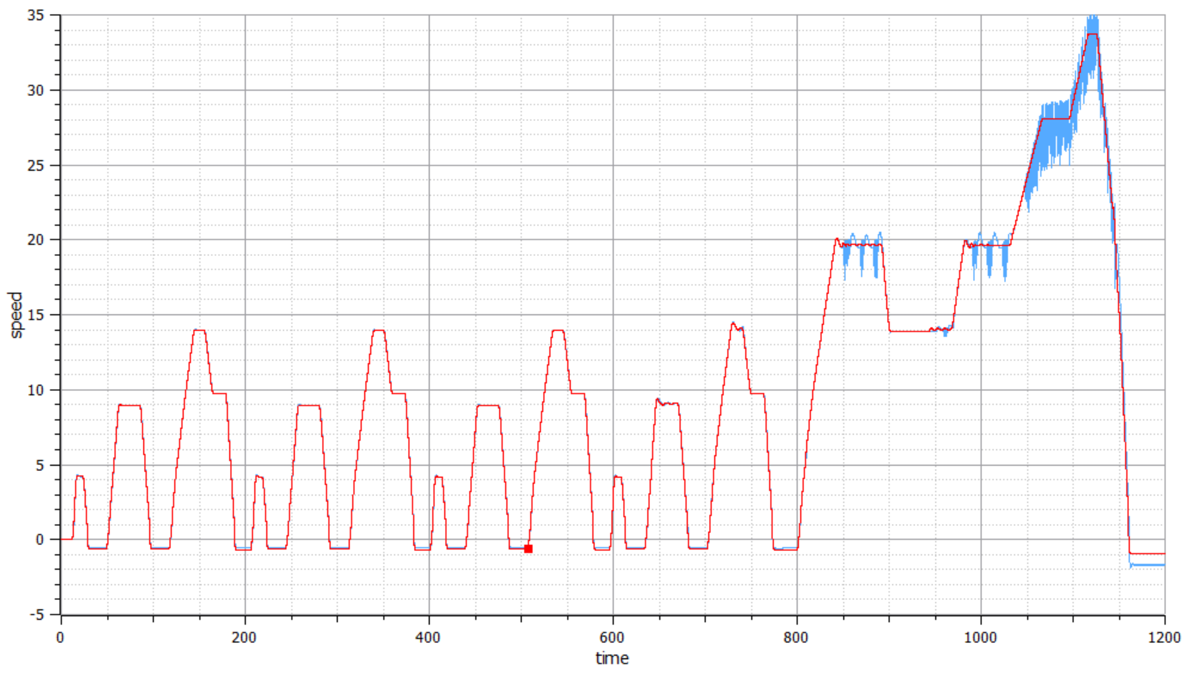


Figure 6. Simulation results of automotive case study

Table 3. Simulation performance of the automotive case study

Events	Intervals	Small steps	Large steps	Total steps	Runtime	Diff	Diff2	Remark
		120 000	0	120 000	31.27 s	–	–	Default simulation
		0	12 000	12 000	3.59 s	14.0	0.55	Unstable oscillations
+		688	11 932	12 620	3.90 s	2.8	0.54	Sine waveforms
	+	34 900	8 511	43 411	18.54 s	0.9	0.55	Sine waveform
+	+	35 005	8 500	43 505	18.81 s	0.7	0.54	Almost identical

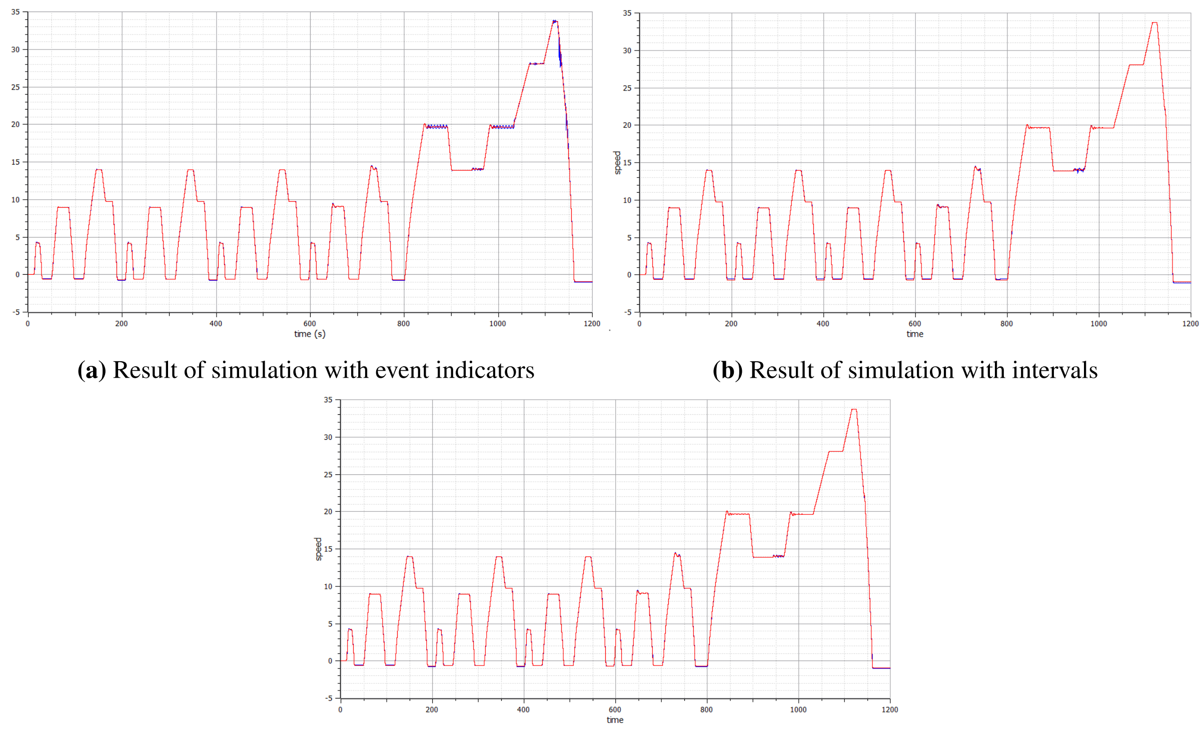
Usability The sensitivity model for the adaptive step size controller requires domain knowledge (e.g. for event prediction), however, some parts of it (e.g. the step sizes assigned to the intervals for threshold-crossing detection) depend on the current simulation configuration. Since both domain-specific and simulation-specific knowledge are required, using the step size controller in its current form may cause difficulties in an industrial environment. A possible solution can be separating the required information and creating the sensitivity model in a final step (see subsection 6.2).

Extensibility The proposed step size controller can be easily extended with additional functionalities: the core of the proposed approach is determining an upper bound of the size of the next step based on several approaches and then choosing the minimum of the calculated upper bounds – it is easy to introduce new analysis methods to the sensitivity model that determine upper bounds based on new aspects. For instance, the algorithm could be combined with the step size controller approach for numerical stability proposed in (Busch and Schweizer, 2011) or the threshold-crossing detection approach proposed in (Esposito et al., 2001).

Possible improvements The proposed solution only considers the possible events in the next step, which – as presented in subsection 4.2 – causes delays. The delays could be minimized using a lookahead method that includes more than one step in the calculation.

Example: In the previously referred example there is a communication step at $t - 0.2$ s. No event is expected in the next simulation step, therefore the next step size is set to 0.15 s. At $t - 0.05$ s the event at t is considered, but since the minimal step size is 0.1 s the event is detected with latency. However, the delay can be prevented by taking two consecutive steps of 0.1 s.

The threshold-crossing detection approach can also be improved by using an advanced solution that uses extrapolation to analyse the possibility of threshold crossing and adjust the step size accordingly. This way it can be avoided to provide intervals and corresponding step sizes.



(a) Result of simulation with event indicators

(b) Result of simulation with intervals

(c) Result of simulation with complete sensitivity model

Figure 7. Results of simulating the automotive case study with step size control

6.2 Lessons learnt

Implementation Throughout the OpenCPS project we have simulated FMUs from various sources, such as OMEdit, Dymola, Simulink, etc. and we have discovered that some FMUs do not comply with the FMI standard completely. As mentioned before, sometimes FMUs can not perform rollbacks. Additionally, in case of the thermostat example the changes in the step sizes introduced odd slopes that later turned out to be caused by the fact that certain defective FMU implementations always perform the step with the previous step size. The difference between the values of the decreasing phase of Figure 2 is caused by the same phenomenon: in the first simulation step (not depicted in the figure) the value does not change and the decrease of the temperature starts in the second simulation step, which is at 0.1 s in one case (denoted by the blue line) and 0.01 s in the other. The time shift causes the difference between the values. This shows that the FMUs must be prepared in order to use adaptive step size control effectively.

Error approximation Currently there are no approximations for the error of the simulation, other than that of the internal simulators of the FMUs. However, – as demonstrated in Figure 5 – discrete events can introduce new types of errors besides numerical stability. The calculation of simulation errors resulting from hybrid co-simulation is a complex theoretical problem and we believe it is an area worth exploring.

Simulation configuration The sensitivity model requires data that can be difficult to acquire, especially when the FMUs to co-simulate originate from different stakeholders. While the *DefaultExperiment* element of the model description file contains simulation-specific information, it only specifies configuration data for simulation with constant step size. Moreover, from a hybrid co-simulation point of view, there are few ways to provide discrete system-specific information in the model description file. A notable exception is the *variability* attribute of *ScalarVariable* elements that can be set to *discrete* thereby denoting an event indicator. However, in case of FMI for co-simulation, timed events and relevant threshold-crossings can not be specified.

Accordingly, in the current implementation the information stored in the sensitivity model is provided by the user as an input of the simulator. However, we believe it would be beneficial to make it possible to provide information specific to discrete/hybrid systems and configuration parameters for simulation with variable step size in the model description file.

7 Conclusions

In this paper we presented a step size controller approach that improves continuous time simulation of discrete event components. The core of the approach is the sensitivity model that describes the simulation scenarios where it is necessary to adjust the step size in order to simulate accurately. The sensitivity model can include sequences of discrete events, timed events and intervals for threshold-crossing detection. The described scenarios can also change dynamically during simulation.

After each communication steps of the master algorithm, the size of the next simulation step is calculated based on the sensitivity model and the current values of variables. The presented method does not require rollbacks.

We have implemented the presented approach within OMSimulator and studied its applicability to the Thermostat example as well as an industrial case study. The results show that the step size controller provides a better compromise between simulation accuracy and efficiency than fixed step sizes by using small step sizes only when is required for simulation accuracy.

We have conducted experiments and found that the step size controller can bridge the gap between continuous time simulation and discrete event components, thereby improving simulation of cyber-physical systems.

Acknowledgement

This project was partially supported by the Hungarian National Research, Development and Innovation Office through the EUREKA_15-1-2016-0008 project as part of the international ITEA3 OpenCPS (14018) project. The authors would like to thank Magnus Eek, Lennart Ochel, Philippe Fiani and Krisztián Mócsai for their assistance in this research.

Emberi Erőforrások Minisztériuma

Gábor Bergmann was partially supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and by the ÚNKP-18-4 New National Excellence Program of the Ministry of Human Capacities.

References

Martin Arnold, Christoph Clauß, and Tom Schierz. Error analysis and error estimates for co-simulation in fmi for model exchange and co-simulation v2. 0. In *Progress in Differential-Algebraic Equations*, pages 107–125. Springer, 2014a.

Martin Arnold, Stefan Hante, and Markus A Köbis. Error analysis for co-simulation with force-displacement coupling. *PAMM*, 14(1):43–44, 2014b.

Torsten Blochwitz, Martin Otter, Johan Akesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 076, pages 173–184. Linköping University Electronic Press, 2012.

Martin Busch and Bernhard Schweizer. An explicit approach for controlling the macro-step size of co-simulation methods. *Proceedings of The 7th European Nonlinear Dynamics, ENOC*, pages 24–29, 2011.

Joel M Esposito, Vijay Kumar, and George J Pappas. Accurate event detection for simulating hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 204–217. Springer, 2001.

Rüdiger Franke, Sven Erik Mattsson, Martin Otter, Karl Wernersson, Hans Olsson, Lennart Ochel, and Torsten Blochwitz. Discrete-time models for control applications with fmi. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, number 132, pages 507–515. Linköping University Electronic Press, 2017.

Virginie Galtier, Stephane Vialle, Cherifa Dad, Jean-Philippe Tavella, Jean-Philippe Lam-Yee-Mui, and Gilles Plessis. Fmi-based distributed multi-simulation with daccosim. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pages 39–46. Society for Computer Simulation International, 2015.

Sahar Guermazi, Saadia Dhouib, Arnaud Cuccuru, Camille Letavernier, and Sébastien Gérard. Integration of UML models in fmi-based co-simulation. In *TMS/DEVS 2016*, page 7. ACM, 2016.

Mert Mokukcu, Philippe Fiani, Sylvain Chavanne, Lahsen Ait Taleb, Cristina VLAD, and Emmanuel Godoy. Control Architecture Modeling using Functional Energetic Method: Demonstration on a Hybrid Electric Vehicle. In *14th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, Madrid, Spain, July 2017. doi:10.5220/0006413300450053. URL `https://hal.archives-ouvertes.fr/hal-01719924`.

Tom Schierz, Martin Arnold, and Christoph Clauß. Co-simulation with communication step size control in an fmi compatible master algorithm. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 076, pages 205–214. Linköping University Electronic Press, 2012.

Antoine Viel. Implementing stabilized co-simulation of strongly coupled systems using the functional mock-up interface 2.0. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, number 096, pages 213–223. Linköping University Electronic Press, 2014.

Steady State Initialization of Vapor Compression Cycles Using the Homotopy Operator

Christian Schulze[1] Andreas Varchmin[1] Wilhelm Tegethoff[2]

[1]TLK-Thermo GmbH, Germany, {c.schulze,a.varchmin}@tlk-thermo.com
[2]Institut für Thermodynamik, University of Braunschweig, Germany, w.tegethoff@tu-braunschweig.de

Abstract

This paper presents a concept how a hybrid DAE of a vapor compression cycle can be initialized in steady state using the homotopy method. A simplified equation system for a vapor compression cycle is described and its computational causality explained. It is discussed how additional boundary conditions can be applied to the simplified equation system, which do not apply to the actual equation system. The robustness and CPU time for different cases is examined and discussed based on transition plots.

Keywords: Vapor Compression Cycle, Homotopy, TIL, ThermalSystems

1 Introduction

Modelica is nowadays widely used in industry and research for object oriented modelling and transient simulation of cyber physical systems. Several Modelica compilers are available and the compatibility between them is continuously improving.

Although Modelica is used for transient simulation of dynamic models, the user is often only interested in the steady state results. And even if the transient simulation is wanted, the initial state of the model is preferred to be in steady state. These arguments particularly apply to vapor compression cycles, because they are computationally very expensive.

Models which were implemented for steady state simulation are fundamentally different from transient models, because simplifications or analytic solutions such as the NTU method (see Verein deutscher Ingenieure (2013)) can be applied. From a user's perspective it would be most convenient to have just one model for both cases. If it is not possible to merge both models, then the two different models should provide the same level of detail and precision.

A typical simple model optimized for steady state simulation of a vapor compression cycle could have two (algebraic) state variables. A dynamic model using finite volume method may have more than 100 (continuous time) state variables, which have to be brought into a steady state. So the dynamic models used for transient simulation are structurally more complex than models which have been optimized for steady state calculations.

If a dynamic model is initialized in steady state, all continuous time states and other initial unknowns (e.g. `fixed =false` parameters) are calculated from an nonlinear systems of equations of the initialization problem. This initialization equation system is a result of the initial equation `der()=0` for all continuous time states.

Most Modelica tools translate the hybrid DAE described by the Modelica equations to an explicit hybrid ODE to solve it. Nonlinear sub-systems are solved inline e.g. using Newton's method. Some tools also provide a DAE Solver to handle both, the differential equations and the algebraic equations. However, the focus of these solving methods is the transient integration rather than solving large nonlinear systems. Often the nonlinear solvers fail to solve the nonlinear equation system of the initialization problem.

One solution to improve the chance for a convergence of the nonlinear system of the initialization problem would be to improve the root finding method. Another solution would be to simplify the model. The homotopy operator is targeted at the second option. The nonlinear equation system of the initialization problem can be simplified to make sure that also less sophisticated solvers find a solution.

Vapor compression cycles are computationally very expensive. The fluid properties used in these models are highly nonlinear and based on complex equations (multiparameter equations of state). The fluid properties also have a very limited numerical range of validity, e.g. evaluating these properties for a negative pressure, temperature or density is impossible. As the equations have been estimated to describe measurement data, they also have an even more restrictive physical range of validity. So it is essential that the system state always is within the range of validity.

2 Homotopy Operator

2.1 Rationale

If a dynamic model is initialized in steady state, many start values are required. From an engineer's perspective, reasonable start values are either obvious and easily defined, or they are almost impossible to provide, because they are actually the desired result of the model. The solving procedure is obscure because it is intellectual property of the tool vendor, and often it is not clear if the solving process has failed because of physical or numerical problems.

Usually the engineer can observe that parts of the system seem to diverge to problematic working conditions, but there is no way to influence the solving procedure.

The larger algebraic nonlinear systems are, the harder it is to trace back the reason for convergence problems of nonlinear systems. It is very difficult to provide good feedback to the user about the underlying problem.

Homotopy is a concept to increase the robustness and simplify the solving procedure for algebraic nonlinear systems. The idea is to use a simplified model (to replace complex dependencies by simplified ones), to calculate a first guess for the result of the actual equation system, and to make use of the similarity of the systems during the transition from the simple to the complex equation system.

For simple nonlinear systems a common Newton solver will be more efficient than using homotopy method. The larger the nonlinear systems are, the more effort has to be invested into the numeric root finding method. The homotopy method enables the engineers to find better ways to define start values, and to focus the solving procedure on relevant aspects. So in other words, it enables to describe the aspects which are obvious to the engineer with a physical understanding of the system. An engineer always crosschecks the plausibility of the calculated system states - the solver cannot do that.

An engineer would also try to generate guess values based on a levelled approach. So first the focus should be on the top level, to set general working conditions including guess values for interface values between subsystems. After that each subsystem has to be processed using the interface values. Interdependencies between subsystems can and have to be broken completely at the interfaces between the subsystems.

The homotopy method requires a simple equation system, which can be used to calculate the same state variables that are calculated from the actual equation system. Additionally it is essential that the resulting solution of the state variables only changes continuously during the transition from the simple to the actual equation system. So the causality must be compatible, and the transition must be continuous. If these two requirements are fulfilled, then the homotopy method can be applied.

2.2 Homotopy in Modelica

The Modelica implementation of homotopy as presented in Sielemann et al. (2011) uses one global dimensionless transition factor λ which is zero for the simple equation system and one for the actual equation system.

The homotopy operator looks like a function in Modelica. It outputs a linear combination of the two inputs:

```
function homotopy
  input Real actual;
  input Real simplified;
  output Real val;
end homotopy;
```

The homotopy function switches term-wise between actual and simple equations. A small example could look like this:

$$y = \text{Term}_{\text{actual}} \cdot \lambda + \text{Term}_{\text{simple}} \cdot (1-\lambda) \quad (1)$$

$$z = y^2 \quad (2)$$

Even though the term for the calculation of y is switched linearly in eq. 1, the eq. 2 to calculate z is switched nonlinearly because it is nonlinearly dependent on y.

In general all derivatives have to be set to zero, to initialize a system in steady state, consequently there will be a large initialization nonlinear system. The homotopy function can be used anywhere in the model equations or initial equations. The function is intended to be used to support solving nonlinear systems and it will be ignored in other cases.

In Dymola there is a separate symbolic analysis of the simplified equation system ($\lambda = 0$) that may have a different computational causality and structure. E.g. the former iteration variables could be calculated explicitly. There is also a symbolic analysis for the initialization nonlinear equation system dependent on the parameter λ which represents the `der()=0` initial equations. The latter equation system has the same structure, dimension and computational causality as if the homotopy method was not used, although the homotopy function calls were inlined.

The solving procedure often implements a few basic steps:

1. λ is set to 0
2. The (separately analysed) simplified equation system is solved
3. The initialization equation system with the current λ is solved using a common root finding method with start values from the last evaluation
4. λ is increased
5. if $\lambda < 1$, repeat steps 3-5, else final execution of 3

Instead of solving one equation system, an equation system has to be solved for each λ-value. So by design the homotopy method is more computationally expensive than solving the actual equation system directly. However, comparing the CPU time is difficult, because the start values have a huge impact, and the chance of convergence for different boundary conditions is also important. Common root finding method usually fail to initialize a larger system in steady state without using homotopy method. Considering robustness and the lack of good start values, it is worth investigating the homotopy method.

Dymola usually uses a common root finding method for nonlinear systems and it is required to activate the homotopy method use from the beginning (this can be done with an Advanced-flag, or an annotation in the model). By default, the initial lambda step size is 0.1 and it remains unchanged until one evaluation fails. If that happens, the step size is reduced and the solver will try again. But the step size will not be reset to 0.1, it remains reduced for the rest of the solving procedure. This sometimes has a negative effect on the computation time, if the transition is highly nonlinear for low λ-values.

Initialization in a partially steady state is usually not meaningful. Parts of a model which are not initialized in steady state can be seen as a dynamic boundary condition to the connected other model part which shall be initialized in steady state. If the time derivative of all states in a subsystem are zero given dynamic boundary conditions, then usually that subsystem is not in a steady state. The time derivative will be zero at the simulation start, but will be unequal to zero as soon as $t > t_{\text{start}}$.

The Modelica homotopy operator is supported by different Modelica compilers such as OpenModelica, Dymola, and SimulationX. But currently not many libraries extensively use this method. E.g. the ThermoPower library (Casella and Leva, 2006) provides models with homotopy as discussed in Casella et al. (2011). The ClaRa library (Gottelt et al., 2017) also applies the homotopy operator, but this feature is not implemented completely. The ThermoCycle (Quoilin et al., 2014) uses homotopy. The TIL library (Gräber et al., 2010), which is also known as ThermalSystem library, does not yet support homotopy, but is the basis for this publication.

2.3 Simple Example

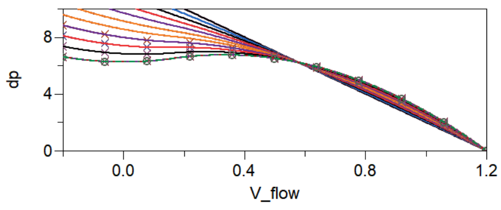


Figure 1. Nonlinear pump characteristic with transition to a simplified solution. The simplified model is linear, can therefore be solved symbolically and has only one solution.

In figure 1 the transition between a nonlinear pump characteristic and linear approximation equation is shown. The Modelica code is listed in section A.

These nonlinear pump characteristics can cause problems because algebraic equation systems based on them may have several solutions. E.g. if the pressure difference is known and not dependent on the volume flow rate, there might be three possible volume flow rates (e.g. at $dp = 6.5$). In fact this is also a common problem for real systems. Homotopy can help to find the wanted (rightmost) solution without giving a start value.

First the simple equation system can be rearranged symbolically to calculate the mass flow rate because it is a linear relationship. Then this result will be used to solve the nonlinear equation system $6.5 = dp(V_{\text{flow}})$ with $\lambda = 0$. So the residual should be equal to zero for the calculated start values. Subsequently lambda is increased (e.g. $\lambda = 0.1$) and the equation system is solved again. But since the new equation system is almost the same as the last one, it is easy for the root finding method to find a solution close to the start value.

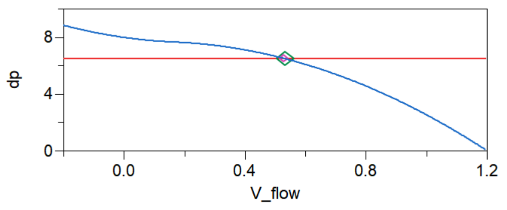


Figure 2. Solving the pump characteristic for $dp = 6.5$ at $\lambda = 0.7$. The last result of the nonlinear system is used as start value (large diamond marker) to find the next result (small diamond marker).

In figure 2 the curve for $\lambda = 0.7$ is shown. The large diamond marker represents the start value from the last solution for $\lambda = 0.6$. The small diamond marker represents the result of the equation system. The start value and the solution are very close to each other, and by reducing the λ step size, the values will be even closer.

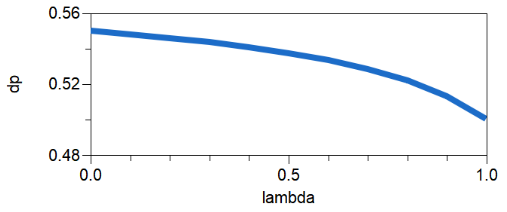


Figure 3. Solutions of the equation system plotted over λ. The transition is continuous and describes the transition from the simple equation system to the actual one.

If the transition is continuous, then solutions calculated for different lambda form a continuous solution path from the simple to the actual equation system. The transition of the resulting algebraic state is shown in figure 3. There must not be a discontinuity or pole in this λ-plot. The λ-plot could also be considered as a root locus plot - the solutions for different λ-values are connected to a line.

3 Initializing Vapor Compression Cycles with Homotopy

We are focusing on finite volume models with balance equations for mass, energy, and momentum. Some components such as the valve and compressor have steady state balance equations. The dynamic heat exchangers are dicretized one-dimensionally. In contrast to that the separator model is a 0-D model with dynamic mass and energy balance. For more details see Schulze (2013). If the dynamic component models shall be initialized in steady state, then an additional initial equation has to be added to set the time derivative of the continuous time state to zero. The continuous time state variables are pressure and specific enthalpy.

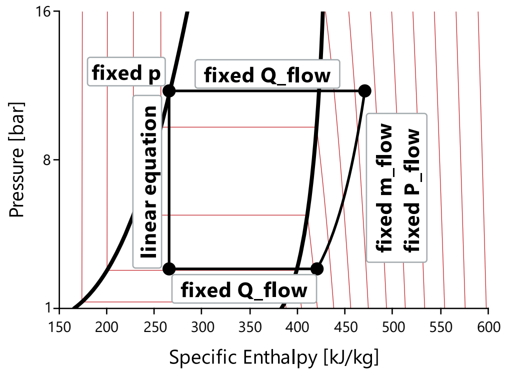


Figure 4. User-defined boundary conditions for the simplified equation system. Heat flow rates in each component are predefined, mass flow rate, separator pressure, and a linear valve characteristic.

In figure 4 the boundary conditions used for this work are shown. The basic concept for the simple equation system is to describe the state of a whole vapor compression cycle using a number of simple conditions:

- Fixed heat flow rates in each heat exchanger
- Fixed mass flow rate and power in the compressor
- Fixed pressure and filling level in the separator
- Linear characteristic in the valve

For this simplified nominal working state, all mass flow rates, enthalpy and pressure states can be calculated. Due to the steady state mass balance all mass flow rates are equal to the one set in the compressor. The high pressure is set by the user in the separator. The low pressure is calculated from the mass flow rate, the high pressure, and linear valve characteristic. Starting from the separator outlet enthalpy in a saturated state (which is known due to the fixed pressure), all enthalpies can be calculated. In a discretized heat exchanger the heat flow rate density is assumed to be constant, so if the control volumes have the same size, the enthalpy difference between two neighboring volumes is constant.

The valve characteristic is a linear equation which connects high pressure, low pressure, mass flow rate, and valve opening area. The valve opening area is usually controlled.The valve opening area must not be removed from the simple equation system, because the controller state will be calculated from it.

The above described simplified equation system does not require the calculation of fluid properties. Usually the heat flow rate is calculated from a temperature difference, but as the heat flow rates are predefined, the temperature has no influence on the result of the simplified equation system. If in contrast a complex heat transfer model would be replaced only by a constant heat transfer coefficient, then the temperature-enthalpy relation is not broken (Casella et al., 2011).

During the transition between the simplified and actual equation system the states must stay within the range of validity of the fluid properties. So the definition of the simple system and the transition to the actual equation system are the most important challenges when using homotopy.

Because the simple equation system is analysed separately, it is solved symbolically and no start values are required. This is one of the main advantages when using homotopy. The user only has to provide the values listed above and nothing more. Usually the user knows how to choose this nominal working state.

This concept also works for more complex vapor compression cycles. In case there are more than two pressure levels, there has to be an additional separator or a component such as an ejector. An additional separator would set the pressure, and an ejector would set the mass flow rate ratio between suction inlet and driving inlet. Also internal heat exchangers do not cause problems, because the predefined heat flow rate decouples the two fluid pipes. If a system consists of multiple connected vapor compression cycles and/or heating/cooling liquid cycles, this simple equation system decouples the cycles and the above shown computational causality can be applied to each cycle.

The level of abstraction of the presented approach is higher than in other publications such as (Casella et al., 2011), many relations have been replaced completely by predefined values, not only by linear relations.

4 Loop Breaker Component

A vapor compression cycle consists of at least 4 components connected to a cycle. If dynamic models are used, then it is no problem to connect these components to a cycle. If only steady state models are used, then the mass balance causes circular dependencies. Each component has a mass balance that basically sets the outlet mass flow rate equal to the (negative) inlet mass flow rate:

$$\dot{m}_\mathrm{A} = \dot{m}_\mathrm{B} \tag{3}$$
$$\dot{m}_\mathrm{B} = \dot{m}_\mathrm{C} \tag{4}$$
$$\dot{m}_\mathrm{C} = \dot{m}_\mathrm{D} \tag{5}$$
$$\dot{m}_\mathrm{D} = \dot{m}_\mathrm{A} \tag{6}$$

This equation system is singular. The equation $\dot{m}_\mathrm{A} = \dot{m}_\mathrm{A}$ can be derived, and no value is set.

To solve this problem using pure steady state models, an additional component called loop breaker is used. This component does not have a mass balance and is therefore underconstrained. The circular dependency is no longer a problem. However, when initializing a dynamic model in steady state, the mass balance cannot be removed. Only for the initialization phase the circular dependency has to be broken.

The locally overdetermined equation system for the mass flow rates can be brought into a balanced form, if one degree of freedom is added to the this part of the initialization problem. This degree of freedom has to be set by the mass balance equations. There are only a few ways to add a degree of freedom to the initialization equation system without changing the continuous time equation system:

1. A parameter with `fixed=false`
2. A discrete state variable
3. A continuous time state variable with `der()=0` as continuous time definition

One possible implementation for a loop breaker can be interpreted as a junction model. Starting point is a component model with one inlet connector and one outlet connector. In this model an additional mass flow rate `mDot_loopbrk` is added to the mass balance. If this component is integrated to the closed cycle of steady state components, `mDot_loopbrk` can be calculated. As no mass is added in the other components, no mass will leave the loop breaker component, therefore `mDot_loopbrk` is equal to zero:

```
  parameter Real mDot_loopbrk(fixed=false);
initial equation
  der(density)=0;
equation
  mDot_in + mDot_out + mDot_loopbrk =
    volume*der(density);
```

It is important to notice, that the whole initialization problem is balanced. It is only locally overdetermined. So by adding a degree of freedom another initial equation can be added.

The presented approach is similar to the one used by (Casella et al., 2011).

5 Separator Model

The separator model has to be treated different from the other models. Generally the separator has two main purposes. First, it separates liquid from vapor in a normal operating condition. Second, it is used to store refrigerant without changing the state of the system. This component is also special because its state cannot be calculated from the constraint `der()=0`, because in normal operating condition, the filling level of this component does not influence the outlet state (pure liquid or vapor). An additional information about the initial filling level or the total mass in the system is required. The additional degree of freedom added for the mass balance loop breaker is used for this purpose. So actually there is a `der(h)=0` initial equation, and a `fillingLevel=initialFillingLevel` initial equation.

The above presented approach to use predefined heat flow rates may lead to another problem: The sum of the predefined heat flow rates and powers might not sum up to 0. This is by definition not a steady state, since more (or less) heat is put into than taken of the cycle. This circular dependency is not properly detected by Dymola, rather the problem is solved numerically. If the energy balance is fulfilled numerically, then the solution can be found. If the energy balance is not fulfilled, then evaluating the simple solution fails. Similar to the mass balance, the system of equations is not in a locally balanced state.

To overcome this overdetermined energy balance and to smooth the transition from simple to actual solution, an additional degree of freedom is added to the separator model: An energy balance loop breaker. Similar to the mass balance loop breaker, an additional variable is added to the balance equation, and it is calculated from all initial conditions - namely the `der(h)=0`. So if the heat flow rates sum up to zero, then the simple equation system result for this energy flow is zero. But if the heat flow rates are unbalanced, then this additional energy flow is equal to the energy balance error for the simple equation system.

For this additional degree of freedom an additional initial equation has to be added. But this initial equation should not be used to define this degree of freedom, but rather set something else which had not been defined yet: the total pressure level. Up to now the pressure is not yet defined, only pressure difference due to the valve characteristic. Of course in the actual equation system the energy balance loop breaker variable should be zero.

The following initial equation has been chosen:

```
homotopy(deltah_loopbrk, k*(p-pInitial))=0;
     k=1e-2;
```

The loop breaker variable `deltah_loopbrk` with the unit $[J/kg]$ is set to zero as actual solution, and for the simplified solution the pressure is set to a fixed value. k is used to define the transition shape between pressure difference and energy boundary condition. As a result of this the energy balance does not have to be fulfilled for the prefixed user values in the simple equation system. However, in the actual equation system the loop breaker enthalpy difference is zero.

6 Specific Enthalpy Breaker Models

Similar to the separator model, it is useful to add additional breaker models, to define the fluid state i.e. specific enthalpy at certain positions. This is possible by adding more degrees of freedom to the initialization problem which disappear at simulation time.

For example the superheat after the evaporator is often controlled to a constant value. So assuming the controller will be successful, the fluid state at that position is known and can be set to a constant value for the simple solution. The enthalpy difference `delta_h` in the model has to be 0 for the actual solution, and for the simple solution it has to be calculated from the constraint `portB.h_outflow= superheatedEnthalpy`. `superheatedEnthalpy` is the enthalpy difference to the saturated enthalpy. So one possible implementation is:

```
Real superheatedEnthalpy=...;
```

```
  parameter Real delta_h(fixed=false);
initial equation
  0 = homotopy(delta_h,
      portB.h_outflow-superheatedEnthalpy);
equation
  portB.h_outflow = inStream(
      portA.h_outflow) + delta_h;
```

7 Controller

In the presented simplified solution, the superheating after the evaporator depends on the mass flow rate, but the outlet specific enthalpy does not. So the solver will try to find a low pressure value, that has the desired superheating temperature for a given specific enthalpy. This equation system is hard to solve and often has zero or two solutions.

However, the controlled variables can be changed for the simple solution. Instead of passing the actual superheating temperature to the controller, a pressure difference to a desired low pressure value can be passed to it (of course the setpoint of the controller has to be added to this difference). A simple model to modify the measured signal could look like this:

```
  parameter Real replacement_desired = ...;
  parameter Real original_setpoint = ...;
  parameter Real k = 1e-6;
  RealInput u "original value";
  RealInput replacement_measured;
  RealOutput y = homotopy(u,
      original_setpoint + (
      replacement_measured -
      replacement_desired)*k);
```

The `replacement_desired` is the desired low pressure, `replacement_measured` is connected to the current low pressure. `k` is used to relate the order of magnitude of a pressure difference to a temperature difference.

Similar to this the capacity controller in the system has to be modified. The controller usually modifies the compressor displacement and consequently the mass flow rate. The cooling capacity is fixed for the simple solution and so is the air outlet temperature. If the controller is not modified, then the output will be limited and the integral part will may be defined by the anti-windup implementation. This issue can be fixed similarly by replacing the measured air outlet temperature with an homotopy term that depends on the controller output itself. So using the above described model the `replacement_desired=0.9` is the desired relative displacement, `replacement_measured` has to be connected to the current relative displacement. `k` is set to 10.

8 Software Experimental Results

In the following a common R-134a automotive vapor compression cycle is examined. The system is shown in figure 5. The cycle has a valve, an evaporator, a compressor, and a condenser with separator and build-in subcooling section. The superheating after the evaporator is used to control the expansion valve (superheating setpoint 7 K). The evaporator air outlet temperature is used to control the relative displacement of the compressor (air temperature setpoint 3°C). The whole system including the controllers is initialized in steady state. The evaporator air inlet temperature and the condenser inlet air temperature are 30°C. The compressor speed is set to 50 Hz (=3000 rpm). All results have been calculated using Dymola 2019.

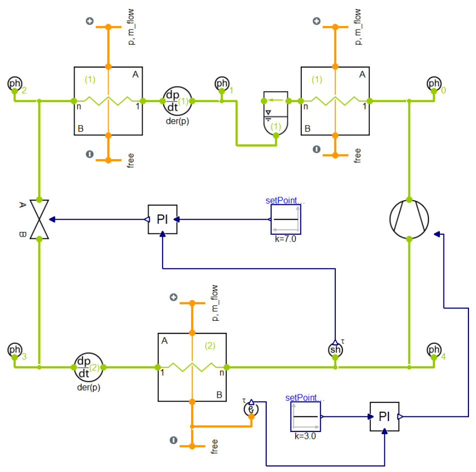


Figure 5. Automotive air conditioning cycle used for simulative experiments, based on TIL. The condenser is implemented using two separate heat exchangers.

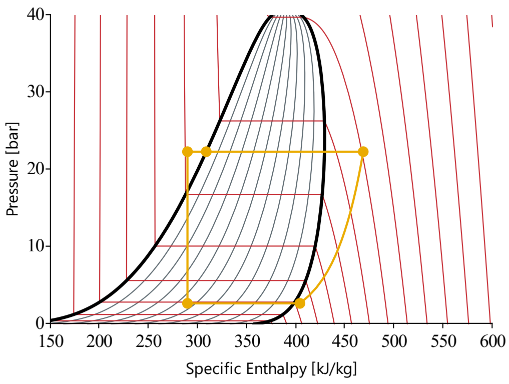


Figure 6. Steady state of the automotive air conditioning cycle used for simulative experiments shown in the ph diagram.

The predefined values in the simplified equation system are:

- High pressure: 25 bar
- Mass flow rate: 0.05 kg/s,
- Condenser heat flow rate: 7000 W condensation + 1000 W subcooling
- Evaporator heat flow rate: 6000 W
- Compressor power: 2000 W

- Nominal linear valve characteristic: mass flow rate = 0.05 kg/s at 24 bar pressure difference
- Setpoint for superheat controller replacement (low press.) = 1 bar
- Setpoint for capacity controller replacement (rel. disp.)= 0.9

The predefined values are enough to replace all initial and start values in the model, if the system is initialized in steady state.

The robustness of the solving procedure is influenced by two aspects:

1. Plausibility of the simplified nominal working state.
2. Similarity between the simplified and nominal operating condition.

8.1 Transition from Simplified Nominal System State to Actual System State

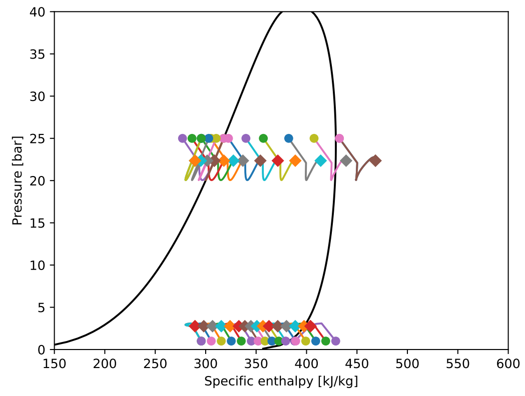


Figure 7. Transition from simplified nominal system state to actual system state. Each marker represents a thermodynamic state in a control volume of the heat exchanger. Circles mark the simplified state. diamonds mark the actual state.

Figure 7 shows the transition between the simplified and the actual control volume states in a ph-diagram. It is clearly visible that the transition of the states is continuous, but the shape of the transition is not linear. There is no simple explanation for transition form. In the simple equation system the pressures are predefined, in the actual equation system the pressures are defined by the refrigerant mass in the components, the temperatures, and heat transfer to the other medium.

As mentioned before, it is important that all enthalpy and pressure states remain in a reasonable range during the transition. Otherwise the fluid properties would cause problems. E.g. it is not possible to provide reasonable property data for a negative pressure or if temperature are below the triple temperature. As can be seen the transition stays well within a reasonable range. Pressures stay below the critical point and above the triple point. The specific enthalpies stay around the two phase region. If the simplified nominal working state is not close to the actual state, this does not seem to be a problem.

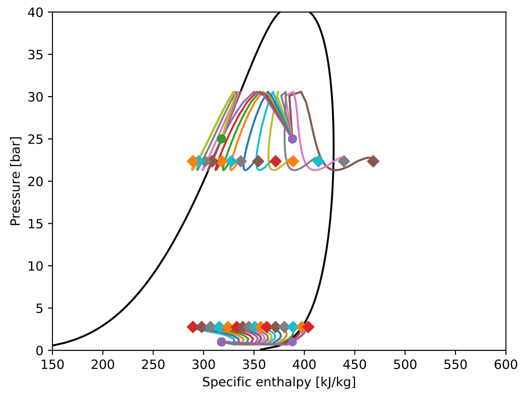


Figure 8. Transition from simplified nominal system state with zero heat flow rate to actual system state.

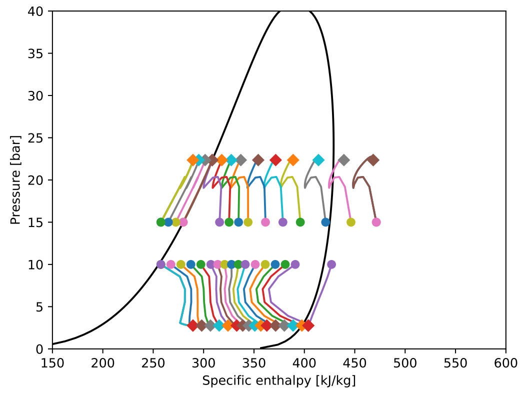


Figure 9. Transition from simplified nomnial system state with low pressure difference to actual nominal system state.

Even the zero heat flow rate use case shown in figure 8 is working fine, although the transition is nonlinear. The pressure levels of the simplified nominal working state seem to be very important for the plausibility of the system state as is visible in the nonlinear transition in figure 9.

8.2 Robustness against Operating State

Often a normal transient simulation from an arbitrary initial state is done to find the steady state of a dynamic model. Homotopy method is an alternative to that. To enable a fair comparison, the simulation times for both cases have been tested. In this section the results of a batch run for different boundary conditions are discussed. The operating states of the system are shown to illustrate the robustness. The following conditions have been varied:

- compressor speed: 10 Hz to 50 Hz (600 to 3000 rpm)
- condenser air inlet temperature: 10°C to 60°C
- evaporator air inlet temperature: 10°C to 50°C

All the dynamic simulations start from the same initial state. The simulation stop time was set to 1000 s, DASSL was used with a tolerance of 1e-4. To increase the chance of convergence, the supheating controller has a comparably low gain value. The results calculated with homotopy method are all based on the same nominal state ("normal"). The model was only initialized with homotopy at $t = 0$s, but not simulated. The total number of cases is 60.

To examine how these results depend on the model size, the same batch run has been done with different levels of dicretization. "2 x volumes" indicates, that the number of control volumes in the heat exchangers has been doubled.

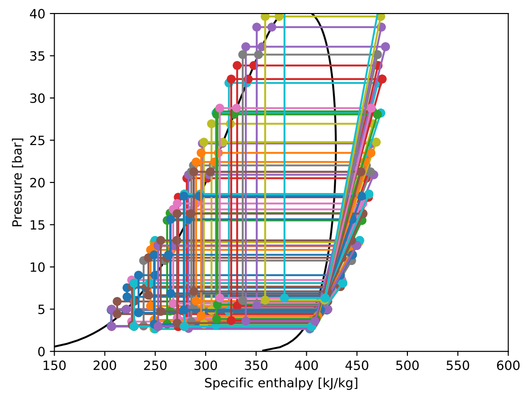


Figure 10. Steady state results of the cycle of the parameter variation. The results vary from very low pressures and temperatures up to the critical point.

Figure 10 shows the steady state results of the batch run. In all cases the superheating was reached, and the filling level of the separator was around 40 to 50%. So the system was always in a normal operating condition, which simplifies the simulation. The pressures are ranging from 2.5 bar to 43 bar.

For a wide range of operating states the above presented homotopy method is capable of finding a steady state. It proved to be very robust and easy to parametrize. However, if the controller limits would be active, the picture might change.

8.3 Computational Effort

The batch run discussed in section 8.2 is now examined regarding its CPU time and function evaluations. The measurements were taken on an i7 (2. gen), each calculation as been executed 5 times to get an average execution time.

In figure 11 the CPU times for the different cases are shown. For the three variations (three levels of dicretization) of the system homotopy method is computationally less expensive than a simulation by a factor of 5-10.

Since the CPU time is difficult to measure precisely and is highly dependent on the CPU an additional indicator was chosen: For the dynamic simulations the number of F-evaluations (evaluations of the RHS of the hybrid ODE), and for the homotopy method the number of evaluations of the residual of the initialization problem. These measures

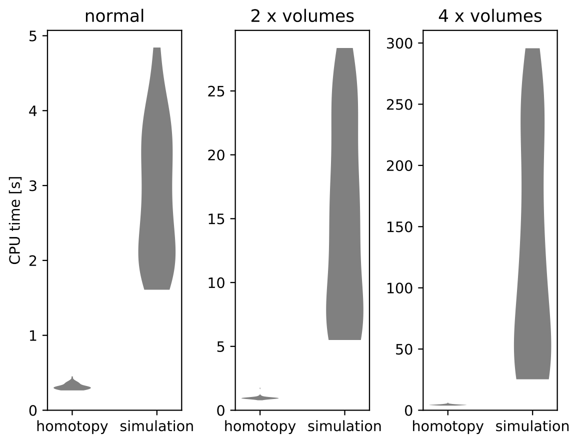


Figure 11. CPU time of a parameter variation compared between simulation with Dassl (tolerance = 1e-4) and homotopy initialization. "2 x volumes" indicates that the number of control volumes has been doubled compared to the normal example. The thickness of this violin plot indicates the density of occurrence.

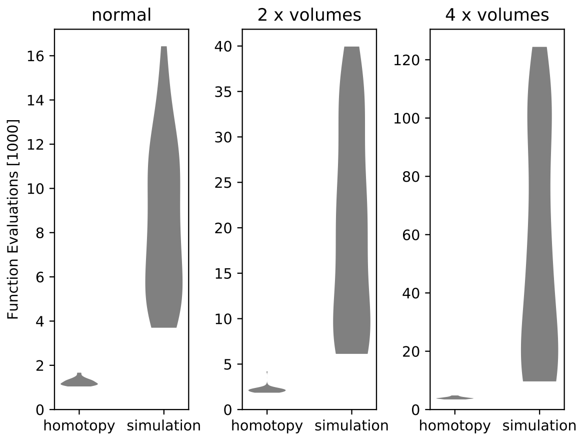


Figure 12. Number of function evaluations of a parameter variation compared between simulation with Dassl (tolerance = 1e-4) and homotopy initialization. "2 x volumes" indicates that the number of control volumes has been doubled compared to the normal example. The thickness of this violin plot indicates the density of occurrence.

usually are proportional to the CPU time. The results are shown in figure 12.

The two figures look very similar, so the CPU time has been measured with a sufficient precision. But comparing the simulation with the homotopy shows that a RHS evaluation of the ODE is not equivalent to a residual evaluation of the intialization problem. The F-evaluation is computationally more expensive.

However, larger parameter studies and user tests have to be done to clearly evaluate the benefits from using homotopy for different initial states, simplified nominal working states, and larger systems.

9 Conclusion

The simplified equation system to describe a vapor compression cycle that was presented in this paper is easy to parametrize, and defines a reasonable system state. The simple model is very abstract but it particularly enables separation of different flow paths and different cycles, so it is potentially able to handle large scale problems, even though this still has to be proven.

The mass and energy balance require a loop breaker to handle the different causality of the simplified model. The energy balance loop breaker turned out to have a positive influence on the convergence. Measured values for controllers have to be modified the define the integral part.

The experiments show that initialization using the presented approach is very robust, and neither the operating state of the system, nor the boundary conditions have to be close to the simplified solution.

Homotopy method lead to a reduction of the computational effort. The transition of the system state including the controllers have a huge impact on the result. A bad homotopy implementation is likely to fail or be computationally more expensive. More time has to be invested to evaluate the user-friendliness, robustness, and computational speed also for other cycles.

10 Acknowlegdements

This work has been partially supported by the BMBF in the VEOTOP project (FKZ: 01LY1809A).

A Appendix

Code for a simple homopty example:

```
model HomotopyPumpLine
  Real V_flow(start=0);
  parameter Real dp0=6;
initial equation
  der(V_flow) = 0;
equation
  der(V_flow) =
    1*(
      6.5 - homotopy(
        actual = (dp0 + 3*V_flow - (3*
            V_flow-0.5)^2*sign(3*V_flow-0.5
            )),
        simplified = (2*dp0 - 10*V_flow)
        )
      );
end HomotopyPumpLine;
```

In Dymola the flag `Advanced.OnlyUseHomotopyMethod` has to be activated to find the rightmost solution. If it is not activated, homotopy is only used if the default algebraic solver fails. If the flag `Advanced.DebugHomotopy` is activated, a csv file with the `V_flow` over lambda will be generated in the current working directory.

References

Francesco Casella and Alberto Leva. Modelling of thermo-hydraulic power generation processes using modelica. *Mathematical and Computer Modelling of Dynamical Systems*, 12 (1):19–33, 2006. doi:10.1080/13873950500071082.

Francesco Casella, Michael Sielemann, and Luca Savoldelli. Steady-state initialization of object-oriented thermo-fluid models by homotopy methods. 2011. doi:10.3384/ecp1106386.

Friedrich Gottelt, Timm Hoppe, and Lasse Nielsen. Applying the power plant library clara for control optimisation. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, number 132, pages 867–877. Linköping University Electronic Press, Linköpings universitet, 2017. doi:10.3384/ecp17132867.

M. Gräber, K. Kosowski, C. Richter, and W. Tegethoff. Modelling of heat pumps with an object-oriented model library for thermodynamic systems. *Mathematical and Computer Modelling of Dynamical Systems*, 16(3):195–209, 2010. doi:10.1080/13873954.2010.506799.

Sylvain Quoilin, Adriano Desideri, Jorrit Wronski, Ian H. Bell, and Vincent Lemort. A modelica library for the simulation of thermodynamic systems. 2014. doi:10.3384/ECP14096683.

Christian Schulze. *A Contribution to Numerically Efficient Modelling of Thermodynamic Systems*. PhD thesis, Dec 2013. URL `https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00057492`.

Michael Sielemann, Francesco Casella, Martin Otter, C. Clauss, Jonas Eborn, Sven Erik Mattsson, and Hans Olsson. Robust initialization of differential-algebraic equations using homotopy. 2011. doi:10.3384/ecp1106375.

Verein deutscher Ingenieure. *VDI-Wärmeatlas*. VDI-Buch. Springer, Berlin Heidelberg, 11 edition, 2013. ISBN 978-3-642-19981-3. doi:10.1007/978-3-642-19981-3.

SESSION 5A: BUILDINGS 3

Co-Simulation Through Exchange of Time-Series Data Applied to an Energy System Model and Detailed Ground Heat Exchanger Model

Hauke Hirsch, Andreas Nicolai, Hans Petzold

Institute of Building Climatology, Technical University Dresden, Germany,
hauke.hirsch@tu-dresden.de

Abstract

In recent years, building energy systems have become an important area of application for Modelica. However, few components, such as ground heat exchangers, remain difficult to implement in Modelica, and thus require co-simulation with an external model. We present a method for coupling a building energy system modeled in Modelica with an external ground heat exchanger model. The so-called waveform relaxation method (WRM) realizes co-simulation by exchanging arbitrary time-series data, instead of constant/polynomial values, as currently possible with the FMI standard. This may allow for performance improvement compared to FMI under certain conditions. A major advantage of this method is the applicability to simulation tools that do not yet support FMI. First, we briefly explain the energy system model (implemented in Modelica) as well as the ground heat exchanger model (implemented in external software DELPHIN). Next, we present different implementations of the WRM coupling method and their results. Finally, we discuss the performance of WRM under certain conditions and compare it to the FMI-co-simulation approach.

Keywords: Co-Simulation, FMI, Building Simulation, HVAC System, Ground Heat Exchanger, Wave Form Relaxation

1 Introduction

As with the development of various libraries such as AixLib (Müller et al, 2016), IDEAS (Jorissen et al., 2018), Buildings (Wetter et al., 2014) and BuildingSystems (Nytsch-Geusen et al., 2012), Modelica has become an important tool for simulation of building energy systems. The Modelica approach is suitable for most HVAC components, as they can usually be described by differential equations with a low number of spatial dimensions. However, there remain physical problems in building energy systems, which are difficult to model in Modelica. One example are heat pump systems with borehole heat exchangers or horizontal ground heat exchangers (HGHX). These systems strongly depend on the two- or three-dimensional temperature distribution in the ground. Until now, there have been few studies addressing these problems. In (Picard and Helsen, 2014) a borefield heat exchanger model was developed, based on a TRNSYS model. The ground is discretized in radial and vertical direction and the results are close to the original model. However, for horizontal ground heat exchangers, cylindrical coordinates are not suitable and therefore a much finer spatial discretization is needed. In (Sangi and Müller, 2018) a model for horizontal slinky coil heat exchangers was developed. The soil is discretized in three dimensions and coupled to a dynamic pipe model. The model does not take into account moisture transport and freezing of soil water content and is limited to the particular slinky coil geometry. Furthermore, the presented results cover only a few days without taking into account real weather conditions such as convection and radiation at the soil surface. These effects, however, are crucial for an accurate prediction of the heat pumps efficiency at a given climate and HGHX size. This is due to the fact that electrical energy demand and heat output of heat pumps strongly depends on their source temperature.

Accurate modeling of HGHX, taking into account all relevant physical processes, requires a fine spatial discretization, usually about <10 mm, as shown in (Ramming 2007, Hirsch 2016). This leads to huge systems of differential equations, which need to be solved efficiently using dedicated software. DELPHIN (Grunewald 1994, Nicolai 2007) is a hygro-thermal simulation software, commonly used for component modeling in building physics. As the physical processes are similar, it can also be applied to transient heat and moisture transport in soils.

A common method for runtime coupling of different software is the FMI standard. It allows for two different coupling methods: the simulation under one common solver (ModelExchange) and coupling of independent solvers (Co-Simulation). While a number of existing simulation tools support this standard, there are still many tools, such as DELPHIN, without available FMI interface.

In the present paper, we propose an alternative approach for the coupling of independent simulation tools, called waveform relaxation method (WRM). WRM was originally established for solving systems of differential equations (Crow and Ilic, 1994), which can be broken into subsystems. Each subsystem is solved over the whole time domain using the time-dependent solution of another coupled subsystem. The method is repeated iteratively until convergence is reached. (Maciejewski et al, 2017) used WRM for the co-simulation of a digital power controller with fixed time step and an electrical circuit model with adaptive time stepping scheme. Both components can be integrated over the entire time span separately, which maintains their characteristics and avoids any change in their numerical implementation. This appears to be an advantage over classical co-simulation algorithms, where both solvers have to stop and communicate at a common time point, and commence integration after exchange of variables.

In our case the WRM will be conducted by coupling Modelica and DELPHIN and thus obtaining a detailed building energy supply system simulation with appropriate modeling of heat and moisture transport in the soil.

2 Ground Heat Exchanger Model in DELPHIN

DELPHIN is a simulation program for coupled heat, moisture and matter transport, commonly used for simulation of porous building materials. In a recent research project we implemented different soil types in DELPHIN, which enables us to model ground heat exchangers as well.

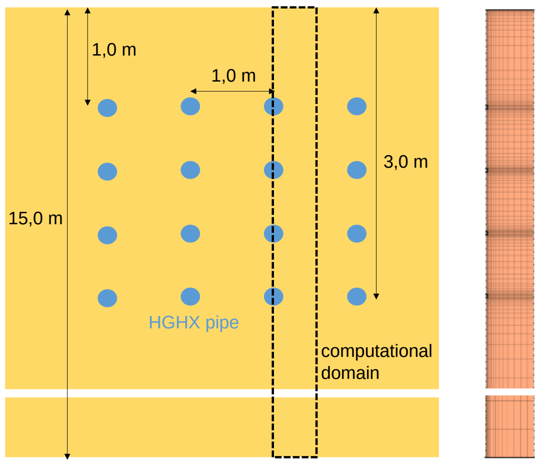


Figure 1. HGHX geometry (left) and discretization of computational domain (right)

2.1 General Assumptions and Geometry

The present study investigates a multi-layer horizontal ground heat exchanger shown in **Figure 1**, hereafter called HGHX. It consists of four horizontal layers of pipes, filled with water-glycol mixture and placed between 1 m and 3 m underneath the surface. The number of vertical rows as well as the pipe length are parameters. Most important modeling assumptions are:

- The model is reduced to the two-dimensional plane perpendicular to the HGHX tubes. Heat and moisture transport along the flow direction is neglected.
- The HGHX is considered indefinitely wide, neglecting lateral boundaries. This allows reducing the effective computational domain to the area showed in **Figure 1** due to the symmetry of the temperature field.

2.2 Spatial Discretization

The right hand side of **Figure 1** shows the discretized computational domain. A fine mesh size is used close to the soil surface and close to the pipes, while the mesh is considerably coarser further away from these boundaries. The minimum mesh size is 4 mm and its maximum is 70 mm. This leads to a total number of 2944 elements. We found these mesh sizes as a good compromise between accuracy and simulation performance, based on a parameter study.

2.3 Boundary and Initial Conditions

At the soil volume surface, the following effects are considered:

- thermal convection
- long wave radiation
- absorption of short wave radiation
- vapor diffusion
- precipitation

The respective climate data is the test reference year TRY 2010 (Zone 4) from (Deutscher Wetterdienst, 2010). We assume constant temperature at the lower boundary, which is located at a depth of 15 m below the surface. This assumption is valid, since the influence of climate conditions at the surface is negligible at this depth. The lower boundary is in direct contact with water. Both lateral boundaries are considered adiabatic, due to the temperature field symmetry.

Heat exchange between the porous material and fluid inside the pipe is approximated by a special boundary condition model. It assumes steady state flow and a constant soil temperature along the pipe. The analytical solution of the pipe outlet temperature reads

$$T_{out} = T_s + (T_{in} - T_s)\exp\left(-\frac{kA}{\dot{m}c_p}\right) \tag{1}$$

with T_{in} and T_s being the inlet and adjacent soil temperature, k refers to the heat transfer coefficient, A is the outer pipe area and $\dot{m}$ and c_p are the fluid mass flow rate and its specific heat capacity.

Before carrying out coupled simulations, we conducted an undisturbed simulation over a range of five years without any heat exchange with the HGHX to make sure quasi-steady state is reached. The resulting temperature and moisture fields are then used as initial values for the coupled simulations.

3 Building Energy System Model in Modelica

The investigated building is a small office building without domestic hot water demand. Heating energy is supplied by a heat pump and the building can be cooled through passive cooling using the HGHX. In a first step, we determined the building heating and cooling demand by simulating the building without considering energy supply and distribution. These values are used as inputs for the energy supply system. The Modelica model is built using components from the AixLib library (Müller et al, 2016) and Modelica Standard libraries and simulated using DYMOLA.

3.1 Generic Building Model

The considered building model was created using the tool TEASER, which allows for generation of archetype building models based on few parameters (Lauster et al, 2016). It is a small office building with an area of 400 m^2 divided into the following zones:

- Office (50%)
- Corridor (25%)
- Storage (15%)
- Meeting (4%)
- Toilets (4%)
- Equipment (2%)

Each zone consists of a 4K model taking into account exterior wall, interior wall, roof and indoor air. During heating season, the temperature set point is 21°C in the daytime and 18°C at night (between 7 pm and 5 am) and during cooling season (between May and September) it is constant at 26°C. This leads to a total heating demand of 13900 kWh/a and a cooling demand of 2900 kWh/a.

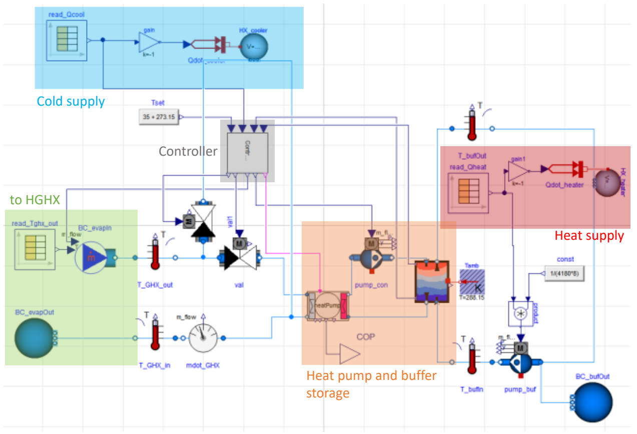


Figure 2. Energy supply system model in Modelica

3.2 Energy Supply System Model

The Modelica model of the energy supply system is shown in **Figure 2**. Basis is a brine/water heat pump, which charges a buffer storage for heat supply. The heat pump operates only in two states: on or off, as this is common practice in real systems. Its electrical energy demand and heating power are calculated through interpolation using manufacturer data tables. A two-point controller determines the heat pump operation based on the buffer storage temperature. The buffer storage model assumes one-dimensional stratification taking into account buoyancy and heat losses to the environment.

The heating and cooling demands as well as the HGHX outlet temperature (calculated in DELPHIN) are implemented as data tables. A controller determines whether the brine delivers heat (in case of heat pump operation) or is supplied with heat (in presence of cooling demand). The temperature at which the brine exits the energy supply system serves as input for the HGHX model.

4 WRM Co-Simulation of Modelica and DELPHIN

4.1 Simple Approach

The WRM differs from FMI-type co-simulation, as the coupled components are computed independently over the whole time domain of interest, rather than solving them for limited time steps. This allows the numerical solvers to run independently from each other. No changes are needed to the code of either simulation tool. For the present study, we coupled the described DELPHIN and Modelica models as shown in Figure 3.

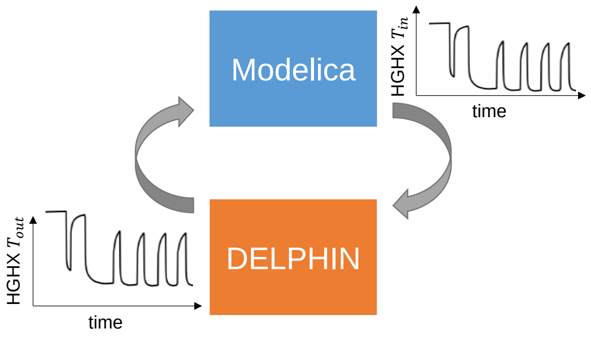


Figure 3. Simple WRM coupling scheme of Modelica and DELPHIN

First, we simulate the energy supply system over the whole time domain, typically one year, assuming a constant HGHX outlet temperature. Thus, we obtain the HGHX inlet temperature from the Modelica system model as time series for one year, which we use now as an input for the DELPHIN simulation. The obtained HGHX outlet temperature from DELPHIN is now again

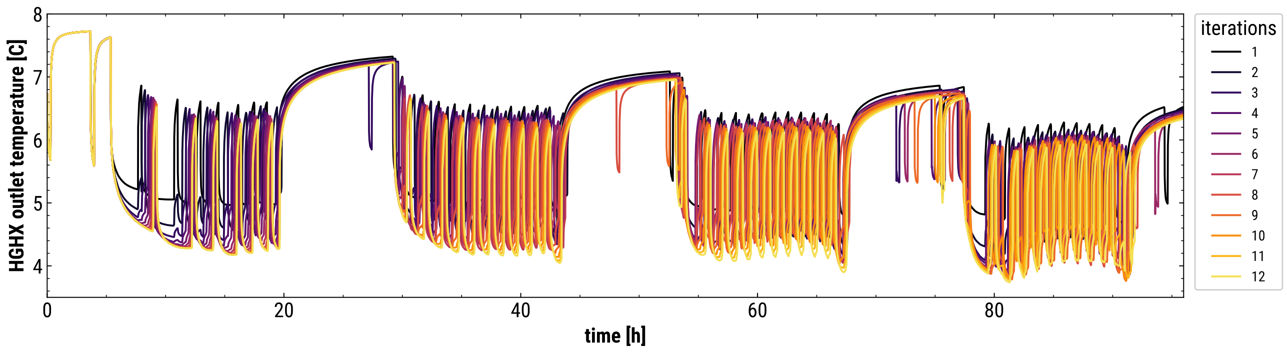


Figure 4. HGHX outlet temperature over four days using simple WRM approach

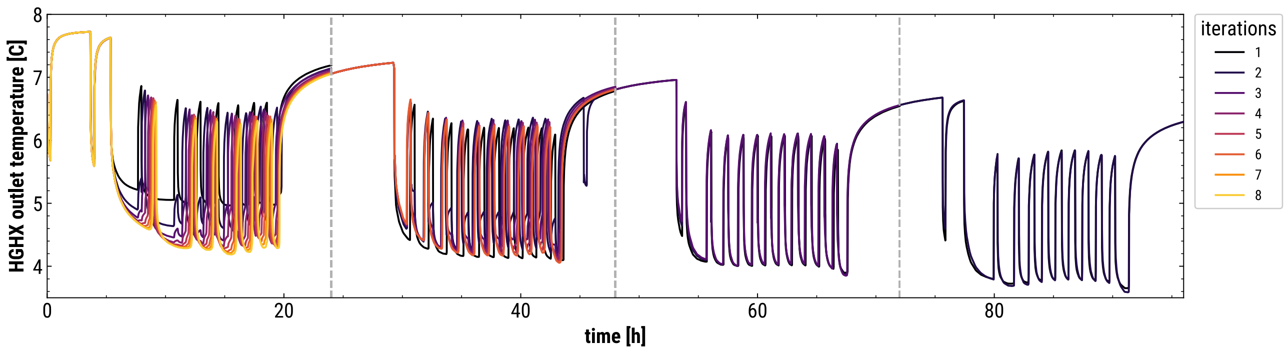


Figure 5. HGHX outlet temperature over four days using stepwise WRM approach with an interval of 24 h

used as input for Modelica in the next iteration. The temperature may changes drastically within short periods of time, due to the intermittent behavior of the heat pump. Hence, in order to provide sufficient accuracy, both models write outputs every minute (sampling rate of the input/output signals). We implemented the method as Python script, which starts DYMOLA and DELPHIN through command line and exchanges values using txt-files. Figure 4 depicts the HGHX outlet temperature for the first four days of the year. We ran the simulation with 12 iterations.

The heat pumps on/off characteristic controlled by a two-point controller leads to a typical behavior of the fluid temperature: When the heat pump switches on, the fluid temperature drops drastically, while when the heat pump switches off, the fluid temperature approximates the temperature of surrounding soil. However, the heat pump heating power strongly depends on its source temperature. A lower HGHX outlet temperature causes a lower heating power, and thus the heat pump needs more time to charge the buffer storage. Due to this non-linear behavior, the system converges for a limited number of iterations only within short simulation times. This can be observed in the example in Figure 4, where convergence appears to be achieved within the first day (gradually smaller differences between lines from red to yellow). However, within the following days, there is still no convergence after 12 iterations. This is due to the time shift of the operation cycles. The problem persists for the remaining simulation period. We expect the simulation to converge for all points in time after an unknown (large) number of iterations, however, simulation effort of DELPHIN is relatively high and thus, the total CPU time becomes too large to be practical usable. Hence, this simple approach to WRM co-simulation does not appear to be suitable.

4.2 Stepwise Approach

In order to handle the problems revealed, we implemented a new scheme, where WRM is carried out in multiple time intervals, rather than simulating over the whole time domain. As depicted in Figure 6, in each time interval, we conduct WRM as described in section 4.1. After convergence is achieved, we store the final model state. In the next time interval, both programs start, using their final states from the previous time interval and the process is repeated. This leads to a temporal decoupling and thereby clearly improves the convergence behavior of the system.

We use the root mean square norm

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(T_{out,i} - T_{out,i}^{prev}\right)^2} \qquad (2)$$

in order to determine the state of achieved convergence. Here, T_{out} is the HGHX outlet temperature in the current iteration and T_{out}^{prev} is the one in the previous iteration.

It should be noted that this method requires the capability of both models to load an initial state at the beginning of a simulation and store their final state when the simulation is finished. In DYMOLA, this can be achieved using the *dsin.txt*, which contains the initial values of the system and the *dsfinal.txt*, which contains the systems final state. Both files have the same structure, so they can simply be replaced. In DELPHIN, the final state is contained in the restart file, which is created at the end of each simulation. It provides the possibility of restoring the state of the simulation model and continuing the simulation from this point in time, when DELPHIN is executed in restart mode.

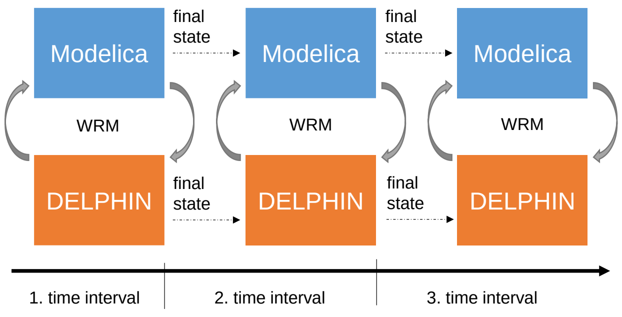


Figure 6. Stepwise WRM coupling scheme of Modelica and DELPHIN

Figure 5 shows the resulting HGHX outlet temperature with stepwise WRM using a time interval of 24 h. In the first day, the system behaves identical to the example from chapter 4.1 and converges after eight iterations. For the next time interval, there is a distinct advantage for the iteration compared to the first time interval: since the soil temperature changes very little within one day and all other variables of Equation 1 remain unchanged, we are able to estimate the initial value of HGHX outlet temperature with relatively high accuracy.

This allows for a noticeable reduction of iterations, so that the second time interval converges within six iterations, the third time interval within three iterations and the fourth time interval within two iterations. Thus, with stepwise WRM, we achieved a total CPU time of around 150 min for one year simulation time, depending on chosen parameters (Core i5-7200). We consider this as practically usable, as it enables us to carry out numerous parameter studies with reasonable simulation effort.

4.3 Assumptions and restrictions

The presented method can be applied to the Co-Simulation of arbitrary models, with the restriction that the problem converges. Moreover, the sampling rate must be significantly below the characteristic time constant of the system, to ensure that important events are sufficiently taken into account. In our study, an operation cycle of the heat pump lasts around 30 min, hence we used a sampling rate to 1 min. Finally, stepwise WRM requires the capability of both models (respectively their implementation), to store their final state and use this as an initial state for the following simulation.

5 Discussion of Performance

5.1 Impact of Time Interval on Performance

The main parameter of the presented stepwise WRM approach is the time interval each WRM is conducted for. Theoretically, shorter time intervals require less iterations, which means an improvement of simulation performance. However, there is a noticeable overhead in starting a simulation model (initialization phase). Further, both Modelica and DELPHIN have variable timestep solvers based on error estimates. When started, both solvers reinitialize and start with a tiny initial time step and only gradually enlarge this time step. This constitutes a fairly large slowdown of the simulation, which in total lowers the performance for shorter time intervals.

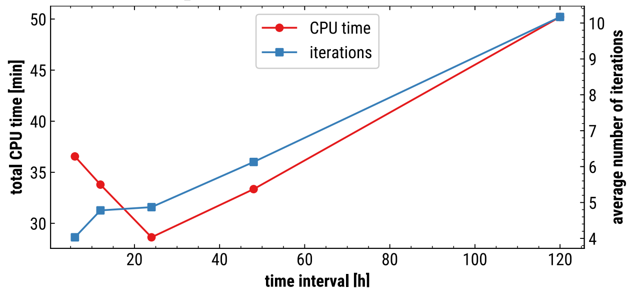


Figure 7. Total CPU time and average number of iterations using different time intervals.

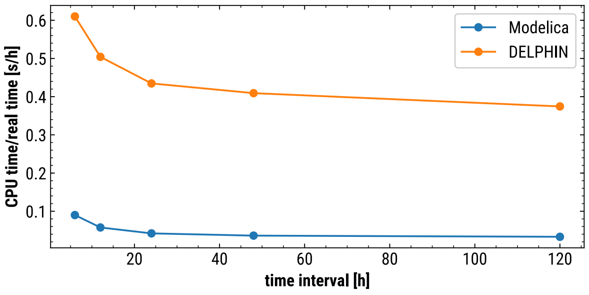


Figure 8. CPU time in relation to real time for both models using different time intervals.

We investigated both effects, by carrying out WRM co-simulations over 30 days with different time intervals ranging from 6 h to 120 h. Figure 7 shows the total CPU time and the number of iterations averaged over all time intervals. As assumed, shorter time intervals require less iterations. However, the total CPU time only decreases between 120 h and 24 h, while very short time intervals between 24 h and 6 h cause an increasing CPU time. This can be explained when considering the CPU time to real time relation for both models, depicted in Figure 8. When using short time intervals both models require significantly more CPU time for the same real time, due to reasons explained. Thus, a time interval of 24 h appears to be a good compromise between both effects.

5.2 Generalization

The WRM coupling technique is very similar to the FMI Co-Simulation with rollback as defined in version 2 of the standard. The WRM corresponds to a Gauss-Seidel-type co-simulation master algorithm. However, with FMI v.2 the input and output signals are eithers constants (as supported by most FMI implementations) or polynomial functions expressed by the derivatives of a Taylor-series expansion. Complex input/output signal shapes as used in the WRM cannot be expressed by polynomials, thus communication step sizes typically need to be much smaller than those used for the WRM. Following the previous argumentation, this would incur a significant restart overhead and yield an overall slower simulation.

However, if an FMI slave were to implement a full rollback, including all variables related to the time integration and error handling, a time interval could be repeated without falling back to tiny initial step sizes (a process also known as hot-restart). In this case, utilizing FMI Co-Simulation for a WRM-type signal input/output handling can be an interesting approach with potentially much higher performance, since the FMI rollback feature can help avoid the excessive overhead in restarting/reinitializing simulations. Currently, only the SimulationX Modelica environment has documented such a rollback functionality.

6 Summary

We investigated the application of the waveform relaxation method (WRM) for a co-simulation between Modelica and an external software. Therefore, we used the example of a building energy system simulation (Modelica), which was coupled to a ground heat exchanger model (DELPHIN). We showed that the non-linear behavior of the Modelica model causes convergence problems when a simple WRM approach is used. The simulation converges only for short time domains within reasonable number of iterations. In order to tackle that problem, we introduced a stepwise approach, where WRM is carried out in limited time intervals. This provides temporal decoupling, so that the simulation converges with noticeably less iterations.

We revealed that the simulation performance of the stepwise WRM strongly depends on the chosen time interval. While long time intervals cause many iterations, short time intervals increase the overhead of restarting/reinitializing the individual simulations. Eventually, a comparison to FMI Co-Simulation has been undertaken. We propose that the possibility of using complex input/output-signals in FMI-coupling may yield a significant performance improvement, in particular if used with a full rollback, where excessive overhead due to reinitializing is avoided.

Acknowledgements

We gratefully acknowledge the support and funding received from the German Federal Ministery for Economic Affairs and Energy in the research project "EnEff:Wärme – Erdeisspeicher und oberflächennahe Geothermie" #03ET1382A.

References

M.L. Crow and M.D. Ilic. The Waveform Relaxation method for systems of differential/algebraic equations. Mathematical and Computer Modelling, Volume 19, Issue 12, June 1994, Pages 67-84.

Deutscher Wetterdienst. Testreferenzjahre von Deutschland für mittlere und extreme Witterungsverhältnisse TRY. DWD, Offenbach am Main, Deutschland, 2010.

John Grunewald. Diffusiver und konvektiver Stoff- und Energietransport in kapillarporösen Baustoffen. PhD thesis, Dresden, TU Dresden, 1997.

F. Jorissen, G. Reynders, R. Baetens, D. Picard, D. Saelens, and L. Helsen. Implementation and Verification of the IDEAS Building Energy Simulation Library. Journal of Building Performance Simulation, 11 (6), 669-688, doi: 10.1080/19401493.2018.1428361, 2018.

Hauke Hirsch, Fabian Rüsing, Gunter Rockendorf. Modellierung oberflächennaher Erdwärmeübertrager für Systemsimulationen in TRNSYS. BauSim 2016, Dresden, Germany. 2016.

M. Lauster et al. Design-Driven Parameterization of Reduced Order Models Using Archetype Buildings with TEASER. BauSIM 2016, Dresden, Germany. 2016.

M. Maciejewski, I. Cortes Garcia, S. Schöps, B. Auchmann, L. Bortot, M. Prioli, and A.P. Verweij. Application of the Waveform Relaxation Technique to the Co-Simulation of Power Converter Controller and Electrical Circuit Models. 22nd International Conference on Methods and Models in Automation and Robotics (MMAR). Miedzyzdroje, 2017. pp. 837-842. doi: 10.1109/MMAR.2017.8046937

D. Müller, M. Lauster, A. Constantin, M. Fuchs, P. Remmen. AixLib - An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework. BauSIM 2016, Dresden, Germany. 2016.

Andreas Nicolai. Modeling and numerical simulation of salt transport and phase transitions in unsaturated porous building materials. PhD thesis, New York, Syracuse University, 2007.

Christoph Nytsch-Geusen; Jörg Huber; Manuel Ljubijankic; Jörg Rädler. Modelica BuildingSystems - Eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme. BAUSIM 2012 IBPSA Germany, 26.-28. September. Conference Proceedings. Berlin, 2012.

Damien Picard and Lieve Helsen. Advanced Hybrid Model for Borefield Heat Exchanger Performance Evaluation, an Implementation in Modelica. 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden. doi: 10.3384/ecp14096857}

Klaus Ramming. Bewertung und Optimierung oberflächennaher Erdwärmekollektoren für verschiedene Lastfälle. PhD thesis, Dresden, TU Dresden, 2007.

Roozbeh Sangi and Dirk Müller. Dynamic modelling and simulation of a slinky-coil horizontal ground heat exchanger using Modelica. Journal of Building Engineering 16 (2018) 159–168. https://doi.org/10.1016/j.jobe. 2018.01.005.

Michael Wetter, Wangda Zuo, Thierry S. Nouidui and Xiufeng Pang. Modelica Buildings library. Journal of Building Performance Simulation, 7(4): 253-270, 2014.

Greenhouses: A Modelica Library for the Simulation of Greenhouse Climate and Energy Systems

Queralt Altes-Buch[1*] Sylvain Quoilin[1,2] Vincent Lemort[1]

[1]Energy Systems Research Unit - Thermodynamics Laboratory, Aerospace and Mechanical Engineering Department, University of Liege, Liege, Belgium
[2]Smart Thermal Energy Systems, KU Leuven, Geel, Belgium
[*]Corresponding author: qaltes@uliege.be

Abstract

This paper presents the results of an on-going project to develop Greenhouses, an open Modelica library for the simulation of greenhouse climate and energy systems. The Greenhouses library is one of the few open-source modeling frameworks for greenhouse climate and crop growth simulation, and the first able to handle simulating the energy integration of greenhouses coupled to thermal systems (e.g. generation and storage units). The proposed modeling framework can be used for multiple purposes, such as the optimal control of the greenhouse actuators, the optimal sizing of the heating appliances, or the optimal integration of the units in the power system. The Greenhouses library also comprises multiple example models, making it readily usable for both research and industrial applications.

Keywords: Greenhouse climate, CHP, Crop yield, Thermal systems, Climate Control, Dynamic modeling

1 Introduction

Greenhouses present the peculiarity of requiring heating, electricity and CO_2. As an energy consumer, they contribute to the depletion of non-renewable energy sources and to global warming through energy-related emissions (e.g. CO_2 emissions from fossil fuel combustion gases). Their energy sources should therefore provide the combined demands in a competitive but also sustainable way. Up to now, the use of combined heat and power (CHP) is proposed as an efficient technology for that purpose: the CHP thermal generation is used for heating purposes, the electricity covers the consumption of the appliances and the CO_2 from the exhaust gases can be recovered to activate photosynthesis. In most cases, there is an excess electricity generation that is fed back to the grid. CHP units in greenhouse horticulture are highly flexible, with the ability to go to full load in less than one hour (Buck et al., 2014). Therefore, when coupled to thermal storage, CHP units can be valuable for the power system by providing services such as load balancing, ancillary services or decentralised storage capacity (Jiménez-Navarro et al., 2018). For example, in a country like the Netherlands, the CHP units dedicated to greenhouse horticulture produced 7.8% of the national production in 2016 (cbs, 2018). Greenhouses can also be coupled to district heatings, in which case activities such as heat recovery from the industry are made possible.

To evaluate the potential of such activities, the complex energy flows within greenhouses must be understood, which also requires ad hoc greenhouse climate models. In addition, a platform for dynamic simulation of the thermal flows interacting between greenhouses and external thermal systems (e.g. district heating networks, generation units, thermal storage) is required. In the current literature, a small number of models are openly available for grenhouse climate simulation and crop growth. Although researchers openly present model structures and simulation scenarios, an open-source simulation platform is still lacking. In fact, the most common climate simulation softwares (e.g. CASTA, KASPRO, VirtualGreenhouse) are not open-access and are not able to handle the integration of greenhouses with external thermal systems. The Greenhouses Modelica library aims at filling this gap by providing an open-source modeling framework capable of simulating greenhouse climate as well as its complex interactions with thermal systems. To that end, the library proposes models covering the following aspects:

- Greenhouse climate, to compute the energy consumption of a greenhouse given its specific design, outdoor conditions and a specific control.
- Thermal systems, with models ranging from heat distribution systems in greenhouses to generation and thermal storage units.
- Crop yield, to account for crop requirements as well as crop behavior (e.g. transpiration and photosynthesis), which influence the indoor climate and thus, the greenhouse energy consumption.

Climate control systems (heating, ventilation, CO_2 enrichment and supplementary lighting) are also included in the library. Furthermore, several numerical methods are developed and implemented in order to enhance the robustness and the simulation speed of the models during initialization and integration.

The library is simple to implement and intuitive to use. The required information for a new user to get started is provided in this paper. Moreover, an additional documentation including a user guide with the required steps to run the models and extended documentation of the library content is available online (cfr. Section 5 for more details).

The library shows potential for both research and industry applications. On the one hand, it can be useful for greenhouse operators when it comes to optimizing the control of the actuators or the sizing of the HVAC appliances. On the other hand, it can be used for purposes such as optimizing the integration of CHPs in the electricity markets.

2 The Greenhouses Modelica library

The Greenhouses Modelica library aims at providing a robust framework to simulate greenhouse climate and its integration with energy systems. The goal is to provide an integrated and fully open-source solution ranging from the computation of energy flows in a greenhouse, to the simulation of complex systems with their control strategy. The Modelica language is thus well adapted to the formulation of this problem, mainly because of its acausal characteristic language that allows inter-connecting the models in a 'physical' way (Casella et al., 2007). The key features of the library are the following:

- Designed for system level simulations.
- Full compatibility (connector-wise) with the Modelica Standard Library and libraries such as ThermoCycle, ThermoPower or Buildings.
- Various numerical robustness strategies implemented in the components and accessible through Boolean parameters.
- High readability of the models (limited levels of hierarchical modeling).

The components provided in the library are designed to be as generic as possible. For example, the detailed geometry records of the greenhouse structure are not compulsory. Instead, only the floor area, the mean greenhouse height and the roof tilt are required. In all the models, default values relative to the most commonly used greenhouse structure (e.g. the Venlo greenhouse) are proposed for all the parameters.

2.1 Modeling of greenhouse climate

Greenhouse climate models have been the object of a substantial literature. While many models have been developed (Bot, 1983; De Zwart, 1996; Impron et al., 2007; Luo et al., 2005; van Ooteghem, 2010), most of them can only be used for a single location and for a specific greenhouse structure and climate. Recently, a more generic greenhouse climate model combining the work of Bot (1983) and De Zwart (1996) was developed. For the purpose of this work, this model, which was developed by Vanthoor et al. (2011b) and validated for a range of climates and greenhouse designs, has been implemented.

The model describes the indoor climate of a greenhouse resulting from the greenhouse design, the outdoor climate and a specific climate control. The indoor climate is characterized by the air temperature, water vapor pressure (to account for the air relative humidity) and CO_2 concentration. Besides, the variables with an indirect influence on the climate are also modeled. These are mainly the characteristics relative to the canopy and the envelope (i.e. the cover, the floor and the thermal screen). In order to compute the indoor climate, the modeling approach consists in applying the energy conservation principles on each greenhouse component and the mass balance on the air. To that end, all the existent energy and mass flows must be modeled. A detailed description of the latter can be found in Altes-Buch and Lemort (2018). Using the encapsulation capabilities of the Modelica language, the balances and flows are defined in independent models that should be inter-connected to build the greenhouse system. The Modelica language offers a high degree of flexibility to the user because:

(i) the greenhouse structure and energy systems are not predefined, i.e. the model can easily be adapted to match different types of greenhouses

(ii) the models are parametrizable i.e. the user can define the materials and system sizes.

The main models of the library are described in the following sections. For a full description of the equations of the models, please consult the online documentation of the library in `https://greenhouses-library.readthedocs.io`. An example of greenhouse model is shown in Figure 1. As it can be distinguished, the greenhouse modeled in this example consists of a two-level heating circuit, roof windows (no side vents), natural ventilation (no forced ventilation) and a movable thermal screen. It should be noted that, when the screen is drawn, the air of the greenhouse is divided in two zones, i.e. below and above the screen. These zones are modeled separately and their respective climate is assumed to be homogeneous.

2.1.1 Surfaces

This section describes the modeling approach used to model the cover, the floor, the canopy and the thermal screen. The energy balance on these surfaces is defined by equation (1):

$$\rho c V \frac{dT}{dt} = \sum \dot{Q} + \sum \dot{Q}_L + P_{Sun} + P_{Light} \quad (1)$$

which takes into account the following exchanges:

- Sensible heat flows ($\dot{Q}$), including convection with the indoor or outdoor air, long-wave radiation between all surfaces or to the sky, and conduction through the soil.
- Latent heat flows ($\dot{Q}_L$), such as the heat exchanged by condensation on the inner side of the cover, condensation or evaporation on the screen, or evaporation on the leaves. These flows can be treated as forced flows, since they are determined by the mois-

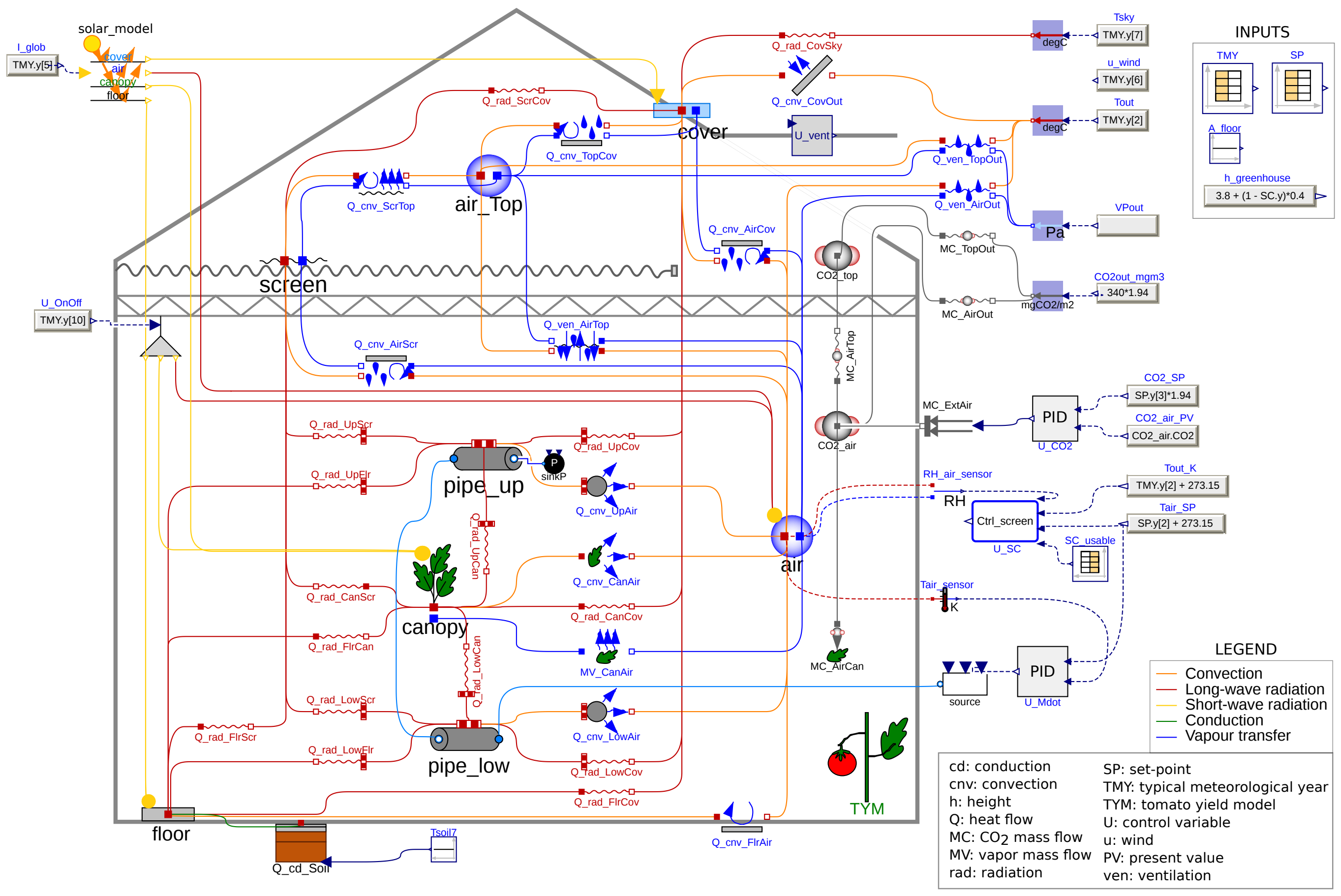


Figure 1. Graphical interface of the greenhouse climate simulation model (*Greenhouse1* in the *Examples* package)

ture mass flow rate caused by condensation or evaporation ($\dot{M}_v$) and the heat of evaporation (Δh_{fg}):

$$\dot{Q}_{L,12} = \Delta h_{fg} \cdot \dot{M}_{v,12} \tag{2}$$

- Short-wave radiation inputs, such as the absorbed radiation from the sun (P_{Sun}) and/or supplementary lighting (P_{Light}).

The water vapor pressure at a surface is defined as the saturated vapor pressure at the surface temperature. No mass balance is applied on the modeled surfaces.

Cover

The cover is the only surface exchanging with both the inside and outside air. The model (*cover* in Figure 1) can be parametrized for any type of glazing (single-glass, double-glass, polycarbonate, etc.). For single glazing, since glass thickness is commonly small (4 mm), conduction is neglected. Depending on the vapor pressure difference, condensation may take place at the inner side of the cover. Evaporation of moisture from the cover to the air is neglected since the condensate is commonly drained.

Canopy

The magnitude of the energy exchanged by the canopy depends on the size of the leaves, which is increased with crop growth and decreased by leaf pruining. To take this into account, the leaf area index (LAI), defined as the leaf area per unit of ground area, is used. The LAI is computed in the crop yield model and input in the canopy model (*canopy* in Figure 1). The heat capacity per unit of leaf area is the main parameter. The canopy temperature has an impact on its photosynthesis and transpiration, which decrease the CO_2 concentration and increase the moisture content of the air, respectively.

Floor

The floor model (*floor* in Figure 1) can be parametrized for a range of floor materials (e.g. soil, concrete). Conduction through the soil is modeled by a nodal model, dividing it into several layers. The temperature of the deepest layer is a boundary condition. Vapor transfer is not modeled.

Thermal screen

The thermal screen (waved line in Figure 1) is a membrane used to reduce the energy requirement to heat the greenhouse. When drawn, thermal losses to the outside are reduced by 38 to 60%, depending on the nature of its material (Bailey, 1988). The screen model (*screen* in Figure 1) can easily be parametrized to cover the wide variety of commercial screens nowadays used by horticulture growers. The screen thickness, commonly less than 1 mm, implies a very low heat capacity. Since the screen is mostly drawn at night (i.e. when there is no sunlight), the absorbed heat from short-wave radiation is neglected.

Given the porous nature of the screen, air and moisture are exchanged through its fabric. The present model assumes that the thermal screen is capable of transporting water from the lower side to the upper side. The storage of moisture in the screen is however neglected. This implies that the vapor that condenses at the screen is either evaporated at the upper side or drips from the screen. The rate of evaporation is therefore lower or equal to the rate of condensation.

2.1.2 Air

The energy balance on the indoor air (*air* in Figure 1) is defined by equation (3):

$$\rho c_p V \frac{dT}{dt} = \sum \dot{Q} + P_{Sun} + P_{Light} \tag{3}$$

which takes into account the following exchanges:

- Sensible heat flows ($\dot{Q}$), including convection at surfaces and ventilation flows (natural, forced or leakage) with the outside air.
- Forced heat inputs, including the short-wave radiation from the sun (P_{Sun}) or supplementary lighting (P_{Light}), which are first absorbed by the greenhouse construction elements and later released to the air.

The moisture content of the air is increased by the transpiration of the canopy and decreased by ventilation and by condensation on the cover and the screen. In the model, it is characterized by the water vapor pressure of the air (P_v), which is determined by the vapor mass balance defined in equation (4):

$$M_H \frac{V}{RT} \frac{dP_v}{dt} = \sum \dot{M}_v \tag{4}$$

where M_H is the molar mass of vapor and $\dot{M}_v$ is the vapor mass flow rate.

The CO_2 concentration of the greenhouse air, being independent from the heat and vapor exchanges, is computed in a separate model (*CO2_air* in Figure 1). Its value is decreased by ventilation processes and by the CO_2 consumption of the canopy, and increased by the CO_2 supply from an external source controlled by the climate controller. The CO_2 mass concentration (γ_{CO2} [mg{CO2} m^{-3}{air}]) of the air is determined in the CO_2 mass balance, defined in equation (5).

$$V \frac{d\gamma_{CO2}}{dt} = \sum \dot{M}_c \tag{5}$$

where $\dot{M}_c$ is the CO_2 mass flow rate.

The top air zone has a very low heat capacity and is only modeled when the screen is drawn (i.e. mostly at night, to mitigate losses in the lack of sunlight). For this reason, its heat and vapor balances are computed in a simplified version of the air model (*air_Top* in Figure 1), in which the heat input from short-wave radiation (P_{Sun} in equation (3)) is neglected. The CO_2 balance (*CO2_top* in Figure 1) is done in the same manner as for the main zone.

2.1.3 Heating pipes

The fluid in the heating pipes from the greenhouse heating ciruit is modeled by means of the discretized model for incompressible flow described in Section 2.4. Heat is transferred by long-wave radiation to the canopy, floor and cover, and by convection to the air. Since the thermal resistance from the outer pipe surface to the air is about 100 times greater than the thermal resistance from the inner surface to the outer one (De Zwart, 1996), the temperature of the pipe surface can be assumed equal to the water temperature.

Greenhouse heating circuits are commonly made of several parallel heating loops. The main parameters of the model (*pipe_low* in Figure 1) are the pipe diameter, the installed length per unit of ground area per loop, and the number of parallel loops. The nominal mass flow rate and the number of nodes in which each loop is discretized are also parameters of the model.

2.2 Modeling of heat flows

Several models are proposed for computing the different types of heat transfer. It should be noted that convection and long-wave radiation are modeled separately.

2.2.1 Free convection at surfaces

The upward or downward heat exchange by free convection from an horizontal or inclined surface is modeled. The heat exchange coefficients are modeled based on the Nuselt-Rayleigh (Nu-Ra) relation (Balemans, 1989). The model can be used for convection at the cover (upward flow, inclined surface), the floor (upward/downward flow, horizontal surface) or the screen (upward flow, horizontal surface). The bi-direction nature of the convective flow on the floor is due to the fact that the latter can be warmer or colder than the air above it. The different natures of the flows lead to different Nu-Ra relations for each surface. Therefore, the user should indicate (by means of the Boolean parameters) which surface is being modeled.

Depending on the status of the thermal screen, the heat flow to the cover can originate either from the top or the main air zone, and the heat flow to the screen can have a different magnitude. Therefore, when the model is used for the cover or the screen, the screen closure (control variable in the global system) is a required input.

2.2.2 Free convection at the leaves

The heat exchange coefficient on the leaves of tomato crop was derived experimentally by Stanghellini (1987). Because of the lack of required input data to compute it, in the present model it is however simplified to a constant value. This coefficient is expressed per unit of leaf area.

In order to compute the global heat exchange coefficient, the LAI is thus a required input.

2.2.3 Free convection at heating pipes

The magnitude of convective heat from the heating pipes to the air depends on the pipe position, which implies a free exchange (i.e. pipes in free air) or a hindered exchange (i.e. pipes situated close to the canopy and near the floor). The free exchange is modeled based on the Nu-Ra relation. The hindered exchange, considered to be forced, is modeled by experimental correlations derived by Bot (1983). The user should indicate which exchange should be modeled by means of a Boolean parameter. The diameter of the pipes and the installed pipe length per unit of ground area are also required parameters.

2.2.4 Forced convection with the outside air

The convection at the outer side of the greenhouse cover is modeled according to the experimental work of Bot (1983), who characterised the heat exchange coefficient at this saw-tooth surface as a function of the wind speed. The wind speed is an exogenous input of the model. The main parameter is the cover tilt.

2.2.5 Natural ventilation

The heat transfer between the inside and outside air due to natural ventilation is computed as a function of the air exchange rate. This rate, derived by Boulard and Baille (1993), depends mainly on two factors. The first one is the window opening, a required input which is set by the climate controller. The second one is the window characteristics (e.g. the wind pressure coefficient and the coefficient of energy discharge caused by friction at the windows), which in order to simplify the model, are set to constant values relative to standard roof windows.

Depending on the status of the thermal screen, the heat flow can originate either from the top or the main air zones. Therefore, the screen closure (control variable from the climate controller) is also a required input.

This model also takes into account the leakage rate through the greenhouse structure, which is dependent on the wind speed (exogenous input of the model) and the leakage coefficient of the greenhouse (parameter of the model, characteristic of its structure).

2.2.6 Forced ventilation

The heat flow from forced ventilation is computed as a function of the air exchange rate between two air volumes, which depends on the capacity of the ventilation system (parameter of the model) and the position of the control valve (required input set by the climate controller).

2.2.7 Ventilation through the screen

Analogously to the other ventilation models, the heat transfer caused by air exchange between the main and top air zones is computed as a function of the air exchange rate, which is the sum of the air rates caused by two mechanisms. The first one is the air exchange through the openings in the fabric of the screen, which is temperature driven and was derived experimentally by Balemans (1989). The second one is the exchange through the gap when the screen is opened, which is caused by density difference and was theoretically modeled by Miguel (1998) using the Navier-Stokes equation. The main required input is the screen closure (control variable from the climate controller).

2.2.8 Long-wave radiation

The long-wave infrared radiation flows are modeled for each exchange between all the surfaces in the greenhouse (red lines in Figure 1). These flows are modeled by the Stefan-Boltzmann equation. The emission coefficients, characteristic of the surfaces, are parameters of the model for which a standard value is proposed in the documentation of the model. The view factor of each surface is computed according to De Zwart (1996) in its component model and is an input of the model.

2.2.9 Short-wave radiation

Short-wave radiation in a greenhouse can be originated from the sun or from supplementary lighting.

Solar model

The main input is the solar radiation incident in a greenhouse, which can be split in three spectral parts: ultra violet (UV, from 0.3 to 0.4 μm), visible light (from 0.4 to 0.7 μm) and near infrared light (NIR, from 0.7 to 3 μm). The visible light has an interest for biological growth and is referred as photosynthetically active radiation (PAR) in greenhouse modeling. The fraction of UV and PAR in the global radiation is 6-10% and 45-60%, respectively (Coulson, 1975). However, for plant growth it is common to assign 50% to PAR, neglect the UV and assign the other 50% to NIR (De Zwart, 1996). Besides the spectral division, the solar radiation can be divided in direct and diffuse radiation. The solar model of this work is simplified by making no distinction between diffuse and direct solar radiation and by assuming that the transmission coefficient of the greenhouse cover does not depend on the solar angle. It should be remarked that the optical properties of the greenhouse elements differ for PAR and NIR.

On the cover, the incident radiation from the sun is partially reflected, absorbed and transmitted inside the greenhouse. The transmitted radiation is absorbed by the construction elements, the canopy or the floor. The transmitted PAR to be absorbed by the canopy or the floor is defined by:

$$\dot{q}_{PAR,\tau} = (1 - \eta_{Glob,Air}) \cdot \tau_{Cov,PAR} \cdot \eta_{Glob,PAR} \cdot I_{Glob} \quad (6)$$

where $\eta_{Glob,Air}$ is the ratio of the radiation that is absorbed by the greenhouse construction elements, $\tau_{Cov,PAR}$ is the transmission coefficient of the cover and $\eta_{Glob,PAR}$ is the fraction of PAR in the outside global radiation (I_{Glob}). When the thermal screen is closed, $\tau_{Cov,PAR}$ is a lumped transmission coefficient of the greenhouse cover and the movable thermal screen.

For instance, the PAR absorbed by the canopy is the sum of the PAR transmitted by the cover and directly absorbed by the canopy and the PAR reflected by the floor and later absorbed by the canopy. In a homogenous crop, this is described by an exponential decomposition of light with the LAI (Ross, 1975):

$$\dot{q}_{PAR,Can} = \dot{q}_{PAR,\tau}(1-\rho_{Can,PAR})\left(1-e^{-K_{PAR}\cdot LAI}\right)+ \dot{q}_{PAR,\tau}\cdot e^{-K_{PAR}\cdot LAI}\cdot \rho_{Flr,PAR}(1-\rho_{Can,PAR})\left(1-e^{-K_{PAR}\cdot LAI}\right) \tag{7}$$

where $\rho_{Can,PAR}$ and $\rho_{Flr,PAR}$ are the reflection coefficients for PAR of the canopy and the floor, and K_{PAR} is the extinction coefficient for PAR of the canopy.

Supplementary lighting

Although the contribution of supplementary lighting is very small during summer, in winter it can double the sun input during a day and thus, have an important impact on crop growth. The illumination model is designed for high intensity discharge lamps (e.g. high pressure sodium (HPS) lamps) and the main parameter is the installed power per unit of ground area. For these lamps, only 17% and 25% of the electrical power is converted into NIR and PAR, respectively. The remaining 58% is released to the greenhouse air (Urban and Urban, 2010). The fraction of radiation absorbed by the greenhouse components is computed similarly than in the solar model.

2.3 Modeling of moisture and CO_2 flows

This section presents the modeling approach for the computation of moisture and CO_2 flows.

2.3.1 Condensation and evaporation

The mass exchange coefficients for condensation and evaporation at the screen and the cover are linearly related to their convective heat exchange coefficients by a conversion factor (De Zwart, 1996). As previously stated, evaporation from the cover and from the screen's lower side is not modeled. Therefore, the mass flow rates due to condensation are prohibited from being negative. Condensation on the upper side of the screen is prohibited as well. Negative flows are avoided by setting the mass transfer coefficients to zero when the water vapor pressure difference between the air and the surface is negative.

2.3.2 Mass transfer through ventilation

Mass transfer occurs in ventilation processes, i.e. between the main and top air zones, and between these and the outside air. The moisture and CO_2 flows accompanying an air exchange are function of the air flow rate, which is computed as explained in Sections 2.2.5 and 2.2.7.

2.3.3 Mass transfer at the canopy

The canopy transpiration originates from a phase interface somewhere inside the cavities of a leaf. The resistance to moisture transport from the leaves to the air was derived by Stanghellini (1987) as a function of leaf temperature, CO_2 concentration of the air, water vapor pressure difference and absorbed solar irradiation. These variables, computed elsewhere, are inputs of this sub-model. Furthermore, transpiration is also function of the dimension of the leaves. The LAI is therefore an input of the model.

The CO_2 flow from the air absorbed by the canopy depends on the canopy photosynthesis rate and the respiration processes. It is computed in the crop yield model and input in this model.

2.4 Modeling of fluid flows

Fluid flows are modeled using the finite volume approach by means of a discretized model for incompressible flow, adapted from Quoilin et al. (2014). The model distinguishes between two types of variables: cell and node variables. The main features and hypothesis of the model can be summarized by:

- Dynamic energy balance and static mass and momentum balance are applied in each cell
- Upwind or central differences discretization scheme
- Uniform velocity through the cross section and constant pressure
- Axial thermal energy transfer is neglected

The overall flow model can be built by connecting several cells in series. The model is compatible with the *Media* package of the Modelica Standard Library, at the condition that the considered fluid is incompressible.

2.5 Modeling of HVAC systems

In the Greenhouses library, several HVAC models are provided in order to enable system-level simulations such as the energy integration of greenhouses with generation and storage units. To that end, performance-based models of CHP units, heat pump and thermal storage units are developed. Although the number of modeled HVAC systems remains limited, the full compatibility (connector-wise) of the Greenhouses library allows the connection with other libraries more specialized in modeling thermal systems (e.g. Buildings, ThermoCycle, ThermoPower, etc.). In all the developed HVAC models, fluid flow is modeled by means of the fluid model described in Section 2.4. To illustrate the modeling possibility of the Greenhouses library, two system-level simulations are included in the *Examples* package.

2.5.1 CHP

The CHP model does not consider part-load operation (ON/OFF regulation is assumed). Thus, constant natural gas consumption and total efficiency are assumed. The electrical efficiency is computed assuming a constant second-law efficiency, whose value is obtained using the nominal operating conditions.

2.5.2 Heat pump

For heat pumps, two models are proposed. First, a peformance-based model similar to the CHP model is developed, in which the second-law efficiency is assumed to remain unchanged in part-load operation.

A second more detailed model is also implemented, in which the heat pump performance are predicted at both full- and partial-load operation by three polynomial laws fitted through manufacturing data (Bolther et al., 1999).

2.5.3 Thermal energy storage

The thermal energy storage model is a nodal model of a stratified tank with an internal heat exchanger and ambient heat losses, adapted from Quoilin et al. (2014). The water tank is modeled using the energy and mass conservation principles and assuming thermodynamic equilibrium at all times inside the control volume. The following hypothesis are applied:

- No heat transfer between the different nodes.
- The internal heat exchanger is discretized in the same way as the tank: each cell of the heat exchanger corresponds to one cell of the tank and exchanges heat with that cell only.
- Incompressible fluid in both the tank and the heat exchanger.
- Axial thermal conductivity is neglected.

2.6 Modeling of crop yield

Several inputs used in the computation of the greenhouse climate (e.g. the LAI, the CO_2 flow absorbed by the canopy) are characteristics of the crop and should be quantified by a crop growth model. Moreover, with a crop growth model, the yield and hence, the profitability (e.g. savings in energy) from different control strategies can be compared. For those reasons, a dynamic crop yield model is implemented. Given that yield models differ between crops, the model implemented in this work is only valid for tomato crop.

Crop growth is related to photosynthesis and most of the existent crop yield models directly relate these two variables without considering a carbohydrate buffer. The buffer is a storage system of the crop, whose function is to store the carbohydrates from the photosynthesis (inflow) before they are distributed to the plant organs (outflow). It has a maximum capacity, above which carbohydrates cannot be stored anymore, and a lower limit, below which the carbohydrate outflow stops. Thus, the in- and out-flows depend on the level of carbohydrates in the buffer and thereby, may not be simultaneous. For instance, crop growth may continue after dusk, when photosynthesis has stopped but distribution can still be possible if the buffer content has not yet reached its lower limit. The presence of a carbohydrate buffer is thus important when modeling crop growth, as suggested in Dayan et al. (1993); Heuvelink (1996); Linker et al. (2004); Marcelis et al. (1998); Seginer et al. (1994).

In this work, a recent yield model developed and validated for a variety of temperatures (Vanthoor et al., 2011a) is implemented. The model structure is shown in Figure 2. The carbohydrate assimilation is modeled by distinguishing three crop parts: the leaves, the fruits and the stems (and roots). Mass balances are applied on each part and on the buffer. For instance, the mass balance on the buffer is described by:

$$\frac{dC_{Buf}}{dt} = \dot{M}_{C,AirBuf} - \dot{M}_{C,BufFruit} - \dot{M}_{C,BufLeaf} - \dot{M}_{C,BufStem} - \dot{M}_{C,BufAir} \quad (8)$$

where C_{Buf} is the availability of carbohydrates in the buffer and $\dot{M}_C$ are the carbohydrate flows, which are computed as a function of fixed parameters related to the tomato crop. The inputs of the model are the instantaneous temperature of the canopy, the CO_2 concentration of the greenhouse air and the PAR absorbed by the canopy. Their values are retrieved from the greenhouse climate simulation model. The main outputs of the model are the LAI, the harvested dry matter, the photosynthesis rate and the respiration rates.

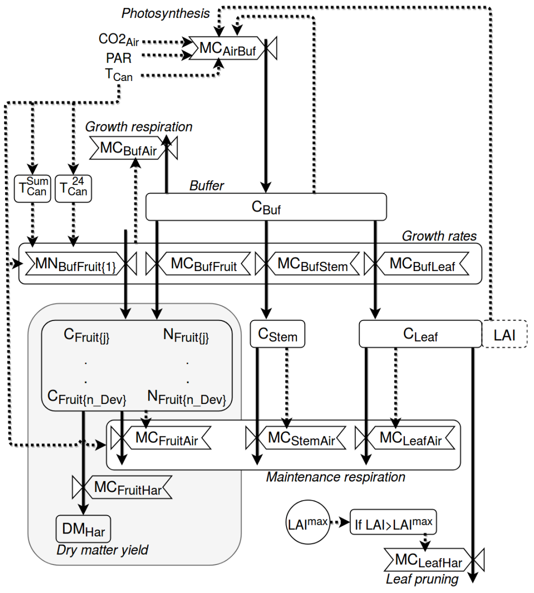


Figure 2. Schematic representation of the crop yield model. Boxes define state variables (blocks), semi-state variables (dotted blocks) and carbohydrate flows (valves). Arrows define mass flows (solid lines) and information flows (dotted lines). Adapted from Vanthoor et al. (2011a).

3 Numerical aspects

The complexity of the final model largely depends on the selected discretization scheme for the piping and for the ground. However, for a typical complete greenhouse example model (e.g. the model *Greenhouse1* in the *Examples* package), the system of equations comprises 4222 unknowns, among which 197 are differentiated variables. After the symbolic manipulation, the size of the non-linear systems of equations is 236 for the initialization problem and 3 for the integration. The typical solving time is 48 minutes for a one-year simulation with a 3 GHz I7 processor.

Because of the important time constants involved in some parts of the model (e.g. the vapor content of the air within the greenhouse), most equations are initialized in steady-state. While this adds some complexity to the initialization problem (in the current example, a system of 236 non-linear equations), it avoids long and unnecessary transients at the beginning of the simulation.

Some equations of the model include conditional statements (in the form of equation (9)) which, during integration, generate state events and therefore decrease the computational efficiency of the model (Jorissen et al., 2015).

$$y = \begin{cases} y_1 & \text{if } k > k_s \\ y_2 & \text{otherwise} \end{cases} \tag{9}$$

In order to increase the computational efficiency of the model, these conditional statements have been replaced by a differentiable switch function. For the general case where $y_1, y_2 \in \mathbb{R}$, the statement is replaced by:

$$y = y_1 \cdot S_k + y_2 \cdot (1 - S_k) \tag{10}$$

where S_k is the value of a differentiable switch function that is determined by the state variable k, which is defined by:

$$S_k = \frac{1}{1 + e^{s_k (k - k_s)}} \tag{11}$$

where k_s is the value of k where S_k is 0.5, and s is the slope of the differentiable switch at k_s. The sign of s is set according to if S_k increases ($s < 0$) or decreases ($s > 0$) with an increasing k. For instance, in the case where some crop parameters differ between day and night, k is the global irradiation, k_s is equal to zero, and y_1 and y_2 are the values of the parameter at daytime and nighttime, respectively.

The model also includes conditional statements in which the output value is equal to the indicator function, defined by equation (12).

$$y = \begin{cases} 1 & \text{if } k \in [k_{s1}, k_{s2}] \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

These conditional statements are approximated by:

$$y = S_k^1 \cdot S_k^2 \tag{13}$$

where S_k^1 and S_k^2 are two differentiable switch functions, which are defined according to equation (11) for k_{s1} and k_{s2} and have opposite slope signs (i.e. the former is negative, the latter is positive).

4 Library implementation

4.1 Library structure

The Greenhouses library is hierarchically structured into different packages, including:

- *Components*, is the central part of the library. It is organized in three sub-packages:
  - *Greenhouse*, contains models from the simple greenhouse components (i.e. all the models described in Section 2.1) to already-build greenhouse models ready to use (similar to Figure 1);
  - *HVAC*, contains the models for generation and storage units presented in Section 2.5;
  - *CropYield*, contains the yield model for tomato crop described in Section 2.6.
- *Flows*, contains models of the flows that are encountered in a greenhouse system. It is organized in seven sub-packages that model the heat, moisture and CO_2 mass transfer, as well as fluid flow. These models are described from Section 2.2 to 2.4.
- *ControlSystems*, organized in two sub-packages, contains control units to control *Climate* (i.e. the thermal screen closure, the operation of supplementary lighting and the window's aperture) and *HVAC* (i.e. the operation of generation units, the storage (dis-)charge) (cfr. Section 4.2 for more details).
- *Examples*, contains examples that demonstrate the usage of this library. It includes simulations of greenhouses (e.g. Figure 1) and two system-scale simulations of a greenhouse connected to a thermal storage, a CHP and a heat pump (e.g. Figure 4).
- *Interfaces*, contains all the type of connectors used in the library.
- *Functions*, contains the empirical correlations used to characterize some of the models presents in the library.

Figure 3 shows an overview of the library structure.

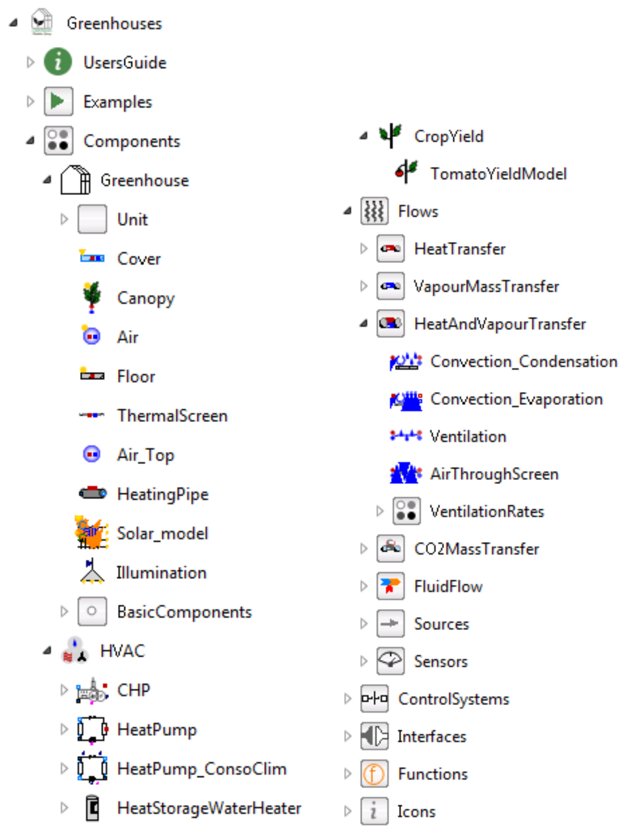

- Greenhouses
  - UsersGuide
  - Examples
  - Components
    - Greenhouse
      - Unit
      - Cover
      - Canopy
      - Air
      - Floor
      - ThermalScreen
      - Air_Top
      - HeatingPipe
      - Solar_model
      - Illumination
      - BasicComponents
    - HVAC
      - CHP
      - HeatPump
      - HeatPump_ConsoClim
      - HeatStorageWaterHeater
    - CropYield
      - TomatoYieldModel
  - Flows
    - HeatTransfer
    - VapourMassTransfer
    - HeatAndVapourTransfer
      - Convection_Condensation
      - Convection_Evaporation
      - Ventilation
      - AirThroughScreen
      - VentilationRates
    - CO2MassTransfer
    - FluidFlow
    - Sources
    - Sensors
  - ControlSystems
  - Interfaces
  - Functions
  - Icons

Figure 3. An overview of the library structure from the Dymola graphical user interface

4.2 Control Systems

Greenhouses have high requirements on indoor climate control. The control strategies used in commercial climate controllers differ from manufacturers and are commonly private-access. For this reason, several control strategies for the control of climate systems are developed. The implemented control strategies are based on a literature review on climate requirements and control practices (Aaslyng et al., 2003; Bailey, 1988; De Zwart, 1996; Dieleman and Kempkes, 2006; Grange and Hand, 1987; Grisey and Brajeul, 2007; Urban and Urban, 2010; Vanthoor et al., 2011a). In the library, depending on the nature of the strategies, two implementation approaches are distinguished: proportional-integral (PI) and state graph based controllers. The library includes models for the control of:

- *Supplementary lighting*: ON/OFF operation determined by a state graph based controller. The strategy sets up a time window for lighting, during which a lighting set-point condition is applied. To prevent cycling, natural light levels must be below or above the set-point for a proving time, and once turned on, lights must remain on for a minimum time.
- *Natural ventilation*: a PI controller sets the windows' aperture based on air sanitation and air cooling, i.e. the air relative humidity and temperature are not allowed to increase above a certain value.
- *Thermal screen*: the screen's closure is set by a state graph based controller model. The screen deployment is done progressively as a function of the outside irradiation. Depending on the night, a small temporary opening of the screen may be required to regulate humidity or temperature.
- *Heating*: a PI controller adjusts the heating power output by varying the supply mass flow rate of the heating pipes according to the difference between the air temperature set-point and actual value.
- *CO_2 external source*: a PI controller adapts the CO_2 supply rate to attain the set-point. In high ventilation conditions, CO_2 enrichment is commonly reduced due to the high exchange rate to the outside air.

The developed control strategies remain relatively simple compared to some state-of-the-art commercial climate controllers. Users are therefore encouraged to develop their own controls systems adapted to their climate requirements.

5 Open-source implementation

Quality of science relies upon basic principle such as reproducibility, transparency or peer-review, which are greatly facilitated by open-source and open-data approaches (Pfenninger et al., 2017). For this reason, the presented library is released as open-source (using the permissive Modelica License 2). The required documentation for a new user to use the models is described in this paper. The library can be downloaded from `https://github.com/queraltab/GreenhouseLibrary`.

In addition to this paper, an online documentation of the library is available in `https://greenhouses-library.readthedocs.io`. Apart from an overview of the library, the online documentation includes a user guide with the required steps for a new user to get started. Furthermore, it includes an extended description of each model of the library, in which the main modeling assumptions and equations are stated. To demonstrate the usage of the library, the example simulations from the *Examples* package are also commented.

6 Conclusion

The development of the Greenhouses library is an ongoing process aiming at providing a completely open-source tool for the simulation of greenhouse climate and its energy integration with thermal systems or the power system. The library comprises a number of components that can be used to simulate a wide range of greenhouse structures and climates. Moreover, the crop growth model allows determining the yield, and hence, the profitability of different control strategies. The components can finally be used to simulate the coupling of greenhouses with generation units and thermal storage, as proposed by the authors in a previous publication Altes-Buch et al. (2018) and illutrated in Figure 4. In that work the library was used to optimize the control of a greenhouse connected to a CHP, a heat pump and a storage system in such a way to maximize self-consumption, leading to significant savings (9 % of the total operation cost) compared to the baseline.

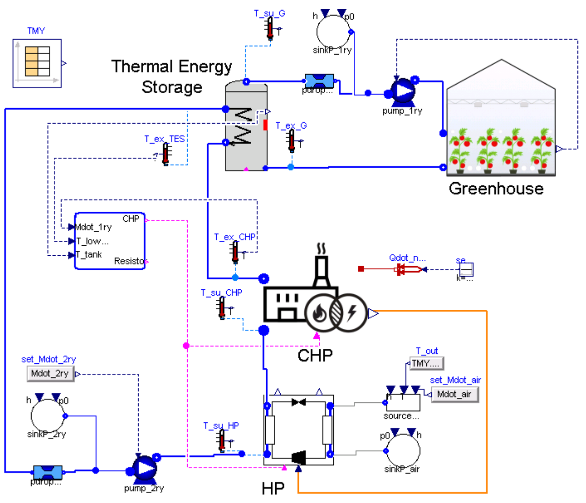


Figure 4. Diagram of a simulation example

The full compatibility (connector-wise) of the library allows the connection with other libraries more specialized in modeling thermal systems, thus increasing the simulation possibilities of the Greenhouses library. The library is released as open-source, ensuring a proper reproducibility and re-usability of this work. Ongoing and future works will mainly focus on the integration of new components and on the validation of the proposed models.

Acknowledgements

The authors thank the Walloon Region of Belgium, which has funded this research in the context of the EcoSystemePass project (convention 1510610).

References

CBS StatLine. Electricity; production and means of production, 2018. URL `https://opendata.cbs.nl/statline`. Last accessed 16 January 2018.

J. M. Aaslyng, J. B. Lund, N. Ehler, and E. Rosenqvist. IntelliGrow: a greenhouse component-based climate control system. *Environmental Modelling & Software*, 18(7):657–666, September 2003. ISSN 1364-8152. doi:10.1016/S1364-8152(03)00052-5.

Q. Altes-Buch and V. Lemort. Modeling framework for the simulation and control of greenhouse climate. In *Proceedings of the 10th International Conference on System Simulation in Buildings*, Liege, December 2018.

Q. Altes-Buch, S. Quoilin, and V. Lemort. Modeling and control of CHP generation for greenhouse cultivation including thermal energy storage. In *Proceedings of the 31st international conference on efficiency, cost, optimization, simulation and environmental impact of energy systems*, Guimaraes, Portugal, June 2018.

B. J. Bailey. Control strategies to enhance the performance of greenhouse thermal screens. *Journal of Agricultural Engineering Research*, 40(3):187–198, July 1988. ISSN 0021-8634. doi:10.1016/0021-8634(88)90206-5.

L. Balemans. *Assessment of criteria for energetic effectiveness of greenhouse screens*. PhD thesis, Agricultural University, Ghent, 1989.

A. Bolther, Casari R., Fleury E., D. Marchio, and J.R. Millet. Méthode de calcul des consommations d’énergie des bâtiments climatisés ConsoClim. Technical Report CSTB ENEA/CVA-99.176R, Ecole des Mines, Paris, 1999.

G.P.A Bot. *Greenhouse climate : from physical processes to a dynamic model*. PhD thesis, Wageningen University, 1983.

T. Boulard and A. Baille. A simple greenhouse climate control model incorporating effects of ventilation and evaporative cooling. *Agricultural and Forest Meteorology*, 65(3):145–157, August 1993. ISSN 0168-1923. doi:10.1016/0168-1923(93)90001-X.

A. Buck, S. Hers, M. Afman, H. Croezen, F. Rooijers, W. van der Veen, P. van der Wijk, and T. Slot. The Future of Cogeneration and Heat Supply to Industry and Greenhouse Horticulture. Technical Report 14.3D38.67, CE Delft, Delft, October 2014.

F. Casella, J. G. van Putten, and P. Colonna. Dynamic Simulation of a Biomass-Fired Steam Power Plant: A Comparison Between Causal and A-Causal Modular Modeling. pages 205–216, January 2007. doi:10.1115/IMECE2007-41091.

K. L. Coulson. *Solar and Terrestrial Radiation*. Elsevier, 1975. ISBN 978-0-12-192950-3. doi:10.1016/B978-0-12-192950-3.X5001-3.

E. Dayan, H. van Keulen, J. W. Jones, I. Zipori, D. Shmuel, and H. Challa. Development, calibration and validation of a greenhouse tomato growth model: I. Description of the model. *Agricultural Systems*, 43(2):145–163, January 1993. ISSN 0308-521X. doi:10.1016/0308-521X(93)90024-V.

H.F. De Zwart. *Analyzing energy-saving options in greenhouse cultivation using a simulation model*. PhD thesis, Wageningen University, 1996.

J.A. Dieleman and F.L.K. Kempkes. Energy screens in tomato: determining the optimal opening strategy. *Acta Horticulturae*, (718):599–606, October 2006. ISSN 0567-7572, 2406-6168. doi:10.17660/ActaHortic.2006.718.70.

R. I. Grange and D. W. Hand. A review of the effects of atmospheric humidity on the growth of horticultural crops. *Journal of Horticultural Science*, 62(2):125–134, January 1987. ISSN 0022-1589. doi:10.1080/14620316.1987.11515760.

A. Grisey and E. Brajeul. *Serres chauffées: réduire ses dépenses énergétiques*. Centre technique interprofessionnel des fruits et légumes (CTIFL), 2007.

E. Heuvelink. *Tomato growth and yield : quantitative analysis and synthesis*. PhD thesis, Wageningen University, 1996.

I. Impron, S. Hemming, and G. P. A. Bot. Simple greenhouse climate model as a design tool for greenhouses in tropical lowland. *Biosystems Engineering*, 98(1):79–89, September 2007. ISSN 1537-5110. doi:10.1016/j.biosystemseng.2007.03.028.

J. P. Jiménez-Navarro, K. C. Kavvadias, S. Quoilin, and A. Zucker. The joint effect of centralised cogeneration plants and thermal storage on the efficiency and cost of the power system. *Energy*, 149:535–549, April 2018. ISSN 0360-5442. doi:10.1016/j.energy.2018.02.025.

F. Jorissen, M. Wetter, and L. Helsen. Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation. In *Proceedings of the 11th International Modelica Conference*, Versailles, France, September 2015. Lawrence Berkeley National Lab. (LBNL), Berkeley, CA (United States).

R. Linker, I. Seginer, and F. Buwalda. Description and calibration of a dynamic model for lettuce grown in a nitrate-limiting environment. *Mathematical and Computer Modelling*, 40(9):1009–1024, November 2004. ISSN 0895-7177. doi:10.1016/j.mcm.2004.12.001.

W. Luo, H.F. de Zwart, J. DaiI, X. Wang, C. Stanghellini, and C. Bu. Simulation of Greenhouse Management in the Subtropics, Part I: Model Validation and Scenario Study for the Winter Season. *Biosystems Engineering*, 90(3):307–318, March 2005. ISSN 1537-5110. doi:10.1016/j.biosystemseng.2004.11.008.

L. F. M Marcelis, E Heuvelink, and J Goudriaan. Modelling biomass production and yield of horticultural crops: a review. *Scientia Horticulturae*, 74(1):83–111, April 1998. ISSN 0304-4238. doi:10.1016/S0304-4238(98)00083-1.

A. A. F. Miguel. *Transport phenomena through porous screens and openings : from theory to greenhouse practice*. PhD thesis, Wageningen University, January 1998.

S. Pfenninger, J. DeCarolis, L. Hirth, S. Quoilin, and I. Staffell. The importance of open data and software: Is energy research lagging behind? *Energy Policy*, 101:211–215, February 2017. ISSN 0301-4215. doi:10.1016/j.enpol.2016.11.046.

S. Quoilin, A. Desideri, J. Wronski, I. Bell, and V. Lemort. ThermoCycle: A Modelica library for the simulation of thermodynamic systems. In *Proceedings of the 10th International Modelica Conference 2014*, 2014.

J. Ross. Radiative transfer in plant communities. In *Vegetation and Atmosphere (Ed. J. L. Monteith)*, pages 13–55. Academic Press, London, UK, 1975.

I. Seginer, C. Gary, and M. Tchamitchian. Optimal temperature regimes for a greenhouse crop with a carbohydrate pool: A modelling study. *Scientia Horticulturae*, 60(1):55–80, December 1994. ISSN 0304-4238. doi:10.1016/0304-4238(94)90062-0.

C. Stanghellini. *Transpiration of greenhouse crops : an aid to climate management*. PhD thesis, Wageningen University, 1987.

L. Urban and I. Urban. *La production sous serre: La gestion du climat*, volume 1. Lavoisier, 2nd edition, 2010.

R. J. C. van Ooteghem. Optimal Control Design for a Solar Greenhouse. *IFAC Proceedings Volumes*, 43(26):304–309, January 2010. ISSN 1474-6670. doi:10.3182/20101206-3-JP-3009.00054.

B. H. E. Vanthoor, P. H. B. de Visser, C. Stanghellini, and E. J. van Henten. A methodology for model-based greenhouse design: Part 2, description and validation of a tomato yield model. *Biosystems Engineering*, 110(4):378–395, December 2011a. ISSN 1537-5110. doi:10.1016/j.biosystemseng.2011.08.005.

B. H. E. Vanthoor, C. Stanghellini, E. J. van Henten, and P. H. B. de Visser. A methodology for model-based greenhouse design: Part 1, a greenhouse climate model for a broad range of designs and climates. *Biosystems Engineering*, 110(4):363–377, December 2011b. ISSN 1537-5110. doi:10.1016/j.biosystemseng.2011.06.001.

Modeling of Low Temperature Thermal Networks Using Historical Building Data from District Energy Systems

Ryan Rogers[1] Vickram Lakhian[1] Marilyn Lightstone[1] James S. Cotton[1,2]

[1]Department of Mechanical Engineering, McMaster University, Canada,

[2]Corresponding Author: `cottonjs@mcmaster.ca`

Abstract

A Modelica library for modelling and comparing District Energy Systems (DES) and Low Temperature Thermal Networks (LTTN) has been developed. The library consists of six unique models and a series of replaceable sub-models that allow for different scenarios for thermal energy generation. The fluid transport model and losses have been tuned using an empirical data set of a district energy system in operation.

An analysis was performed to compare the performance of an existing, operational four-pipe DES against an alternative design that consists of two one-pipe LTTN. The results show that the LTTN implementation can drastically reduce the natural gas usage and in turn the carbon emissions of a district energy plant by over 90% during a two-week period in the transitional month of October for a thermal microgrid in Southern Ontario, Canada.

Keywords: thermal microgrid, district energy, thermal transport systems, carbon emissions

1 Introduction

District Energy Systems have historically been implemented in areas with high heating demand and/or cooling to increase energy efficiency due to the utilization of larger industrial generation stations and allow the sharing of energy resources (Lund *et al.*, 2014). These systems use a centralized plant that heats and cools a thermal transport fluid to a set temperature before distributing the conditioned fluid to different buildings within the community using a piping network. Traditionally these piping networks consist of four pipes, a supply and return for heating, and a similar two for cooling. In this way, each building has access to centralized conditioning, and can access energy through an Energy Transfer Station (ETS) that consists of a heat exchanger that interfaces with the district pipes (Figure 1.A).

Although these systems have proven effective at consolidating a community's energy production, especially within large scale institutional campuses, due to increased awareness of the effects of greenhouse gases, there is significant research interest in developing thermal microgrids and a shared thermal economy as a means to reduce emissions.

A thermal microgrid does not depend entirely on this centralized plant, but distributes the load to decentralized energy generation, from traditional units or renewable sources (solar thermal, deep lake cooling, etc. (Li *et al*, 2016)). The control and balancing of the system is done by an Energy Management Centre (EMC) that allows for better utilization of energy by coupling the energy production resources with demand management strategies. The system can be further exploited for increased efficiency and reduced emissions with the modification of the distribution loop from a high temperature distribution to a Low Temperature Thermal Network (LTTN).

A Low Temperature Thermal Network replaces the two fluid, four-pipe system present in traditional District Energy Systems with a singular low temperature working fluid. Like District Energy Systems, this low temperature fluid is supplied to buildings within a community using a thermal network, which interfaces with each building through an ETS. Although these systems can feature a four-pipe thermal network, traditionally one- or two-pipe thermal networks are used (Bünning *et al,* 2018). LTTN also differ from the traditional system because they require the use of heat pumps at the building to receive/ inject thermal energy from/into the distribution thermal fluid temperature to match the needs of the internal distribution of the building. This allows for customization within each building as each system is not constrained to match the characteristics of the thermal loop. This necessitates that at every building interface, one or two heat pumps are required to provide the building's heating and cooling demands, depending on whether there is a need for simultaneous heating and cooling.

The implication of this is that the low temperature of the thermal loops allows for a higher level of thermal energy heat capture from sources within the DES or from rejected heat from cooling demand, as the lower temperature within the DES allows for greater temperature ranges for capture, and thus allowing for greater system level energy utilization. Additionally, because the LTTN features a low temperature transport

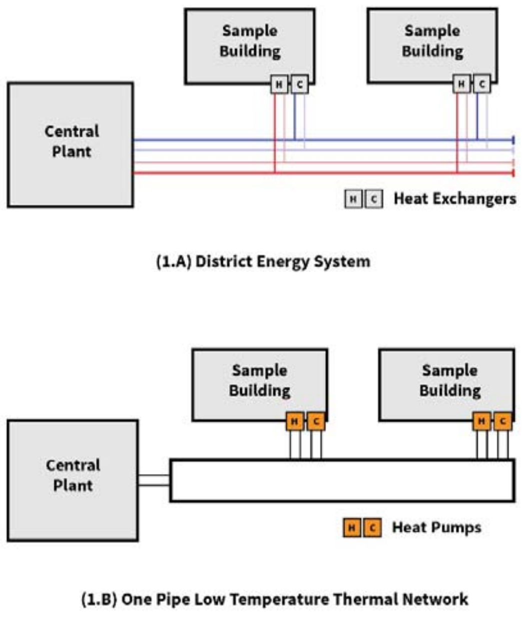


Figure 1 Different thermal network systems

fluid, the resulting system energy losses to the environment are reduced due to the lower temperature difference between the piping network and the ground, which results in a lower rate of heat transfer.

A consequence of this LTTN is that each of these heat pumps will have an additional electrical load. However, the countering benefit of LTTN's ability to capture low-grade waste energy from a community provides a potential method to off-set further heating demand. (Lund *et al*, 2014).

In the LTTN framework (Figure 1.B), the building load is connected to the ETS through a heat pump. The heat pump interacts with the LTTN through a heat exchanger, where both the supply and return connections are connected to the same pipe. This results in a singular pipe which connects all buildings in series and can in turn share energy between buildings. Unlike, the four-pipe District Energy System, after the working fluid leaves the ETS, instead of being returned to the centralized plant, it continues on to the next building. During seasons with simultaneous heating and cooling loads, these series connections allow buildings to act as producers. For example, if a building requires cooling, it will transfer energy into the thermal network, and raise the temperature of the working fluid. This additional energy can be used towards another building's heating demand and, as such, the energy demand upon the centralized plant is reduced. This works similar in reverse, as heating demands cool the temperature of the thermal network and as such benefit other buildings that require cooling. As this pipe is at one temperature, it does not give a preference to either heating or cooling demands of buildings but allows for both. It is the role of the equipment at the central plant to overcome any thermal imbalances and control the setpoint temperature of the thermal fluid.

Due to this tradeoff between higher thermal energy utilization and increased electrical consumption, there exists a need to understand the system dynamics of LTTN systems to determine the impacts on total energy usage and implications on greenhouse gas emissions. Modelica, due to its object-oriented nature and its ease in modeling fluid systems, was chosen to fulfill this need. Using a combination of both existing Modelica equipment models and implementing new analytical models, a modeling hierarchy was created. These models were then tuned using empirical data of an existing traditional four-pipe District Energy Systems before implementing these models as a one-pipe LTTN for the same community, as is described in Section 2. The performance of both these systems was compared by contrasting total resource utilization, peak electrical power requirements and carbon emissions of both systems.

2 Models

In order to accurately model both District Energy Systems and LTTN, a Modelica library was created to model the piping, ETSs, and a simplified EMC within the system. These models incorporate components from both the base Modelica Standard Library as well as the AixLib Building systems library (Müller *et al.*, 2016).

2.1 Thermal Pipe Loss Model

The thermal pipe loss model, *DistrictPipe* is utilized to simulate the heat losses from a buried pipe to the environment. The model is based on an analytical model for steady state pipe losses developed by Wallentén (1991). The model considers pipe diameter, insulation diameter, buried depth and ground surface temperature taken from climate data. It calculates a heat transfer resistance between the working fluid and the ground using the buried depth, pipe size, and the thermal conductivity properties for the ground and pipe (1). Using this resistance, the temperature between the fluid and the ground surface can then be used to calculate the energy lost across the pipe.

$$q = 2\pi\lambda_g(T_1 - T_o) \cdot h_1(H, r_{o,} \beta) \qquad (1)$$

Where q is the pipe heat losses
λ_g is the thermal conductivity of the ground
T_1 is the temperature of the fluid in the pipe
T_0 is the temperature at the ground surface
h_1 is the heat transfer coefficient for the losses
H is the buried pipe depth
r_o is the outer radius of the pipe insulation
β is a dimensionless parameter relating the ground's thermal conductivity to that of the pipe's insulation.

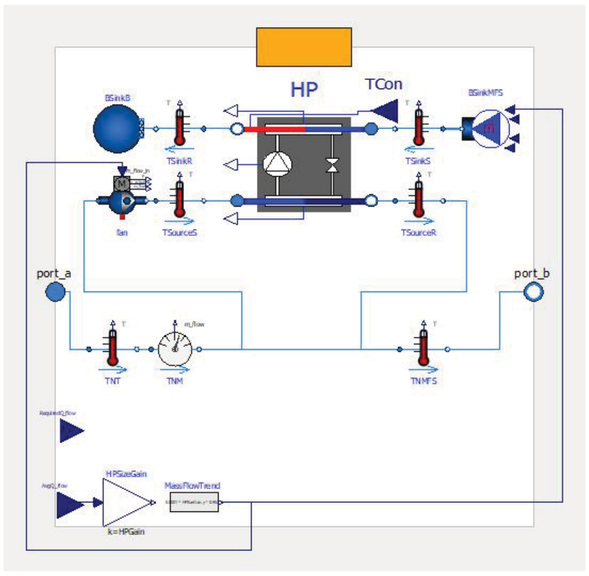


Figure 2. One-Pipe Energy Transfer Station model

Using the Modelica standard library, the resistance and temperature parameters were integrated in the *DynamicPipe* model which then created a discretized heat transfer model for the system. The *DynamicPipe* model also calculates the pipe pressure losses for the *DistrictPipe* model allowing comparisons to be made between different pipe network configurations pump power requirements.

2.2 Four-Pipe Energy Transfer Station

The four-pipe ETS is a standard energy transfer station that consists of a pump, heat exchanger with unity efficiency, and a return outlet. The pump is used to direct a set amount of working fluid away from the main header and into the heat exchanger. This diverted mass flow set point is calculated using a relationship derived from equipment datasheets that relates average buildings heating or cooling requirements to a heat exchanger's nominal mass flow rate.

The building load interfaces with the ETS through two Modelica *RealInput* interfaces. The first connector supplies the seasonal average energy requirements for the building which is used to calculate the mass flow rate to the heat exchanger. The second is the energy required by the building from the ETS. This is used to dynamically calculate the inlet temperature to the heat exchanger on the building side, changing the heat exchangers performance and describing the load within the building itself. A simple control strategy based on a building set temperature and equipment trends gathered from datasheets is also implemented which controls the energy transfer, as well as enforces maximum and minimum energy draw constraints on the heat exchanger.

2.3 One-Pipe Energy Transfer Station

The one-pipe ETS expands on the four-pipe ETS by replacing the heat exchanger with a heat pump model (Figure 2). Since the thermal output of this model is equipment dependent, two models have been made, one for heating and one for cooling. In both cases, Carnot based heat pumps were used from AixLib (Müller *et al*, 2016). These models condition the working fluid to a fixed outlet temperature and calculate the Coefficients of Performance (COP) and power consumption for the equipment based on a Carnot efficiency.

$$COP = \eta_{Carnot} \frac{T_{hot} - T_{cold}}{T_{hot}} \tag{2}$$

Where T_{hot} is the temperature of the condenser
T_{cold} is the temperature of the evaporator
η_{Carnot} is the Carnot efficiency of the heat pump when compared to the ideal cycle

Although this model is not fully based on standard equipment, the Carnot models allow for a range of buildings to be tested without the reliance on equipment look up tables. To further improve the accuracy of the models, a building set temperature control system as well as minimum and maximum energy draw constraints were enforced on the heat pump. These constraints ensured that the COP operates within a reasonable range of existing commercial equipment (COP = [3.09 – 4.20]).

2.4 Energy Management Centre

The Energy Management Centre model simulates the centralized plant production for both the District Energy System and the LTTN. It contains the fluid conditioning system, sensory models used to determine the total energy draw and an equipment dispatcher for the energy production. This dispatcher connects to four replaceable sub-models. These models break down the energy production into four categories: on-peak heating (x hours/day), off-peak heating (y hours/day), on-peak cooling (z hours/day) and off-peak cooling (w hours/day), where *peak* refers to the hours within a day when the equipment would be operated during the hours when there is electrical peak demand upon the electrical grid. This is of interest in areas where baseload electricity production is primarily emissions free, except for peak usage where there is fossil fuel generation (e.g. Ontario or British Columbia, Canada). This typically refers to the between the hours of 7:00 to 19:00 per day during the month of October.

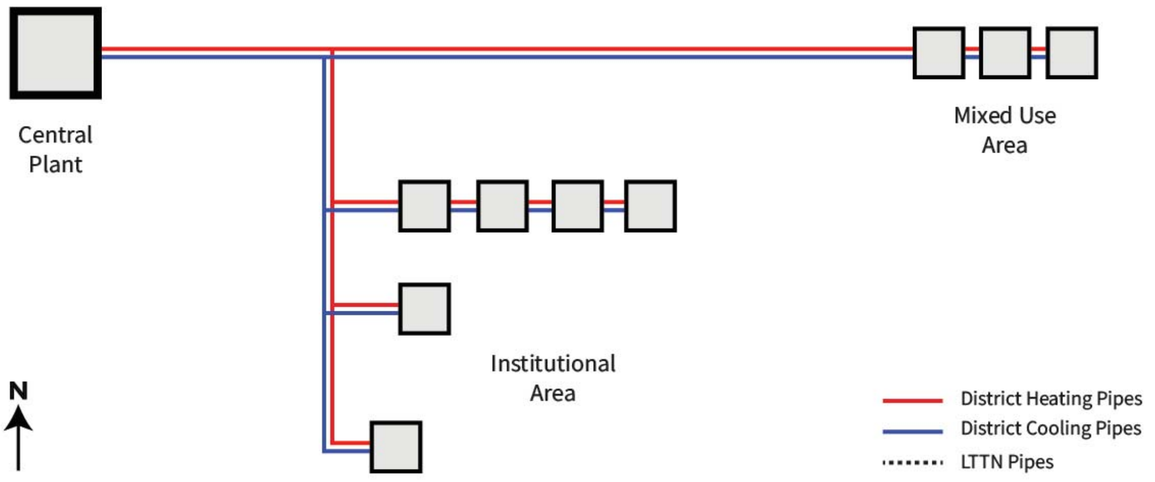


Figure 3. District Energy System case study

These components within the model can easily be replaced with four different models of mechanical equipment: a boiler, a combined heating and power plant (CHP), an air source electric chiller and a ground source heat pump. By implementing these models within the four different energy production categories, they can convert the EMC's thermal energy production requirements to electrical power requirements, natural gas demand and carbon dioxide emissions which can then be used for system comparisons.

2.5 Supplemental Models

Additionally, various sub-models were also created to simulate the thermal network system. One of these sub models is a building load model, that incorporates real building data within the Modelica environment through a timetable. This system can handle data at a variety of time steps and also produces the buildings average energy requirements to the other models.

Another sub-model that was created, the *IsoPipe* model, which was used to model the heat energy and pressure losses for district energy pipes that were not buried underground. These could exist within the interior of buildings or within underground civil infrastructure such as parking garages or utility corridors.

Additional to these components, an array of connectors was also made in order to reduce the total number of connections required within the high-level thermal network models.

3 District Energy Case Study

To validate the four thermal energy loop models, a case study was chosen for simulation. This case study focuses on a nine-building district energy system located in a high-density community within Southern Ontario, Canada. The community contains mixed use, institutional, commercial and residential building types. Figure 3 shows the general layout of this system; the total foot print of the community is approximately 0.40 km^2.

3.1 Design and Layout

The District Energy System can be portioned into two major sections. The first area is the mixed-use area located on the east end of the community. These three buildings contain both office space and residential apartments all which require both heating and cooling from the district system.

The second area located in the southern section of the community consists of six institutional buildings. The four buildings located to the north, consist of an office building and three community institutional buildings. Additional to this, a large municipal building is located in the southernmost part of the campus while a medical research facility is centrally located.

Both areas are connected to the centralized plant using a thermal network. The heating pipes are 15.24 cm diameter steel pipes with an insulated diameter of 24.51 cm. While the cooling pipes are larger diameter 20.32 cm HDPE without insulation.

3.2 Comparison between Model and Observations

Real-world observations were used to tune the model to more closely match reality. To ensure the model recreated the existing system as closely as possible, information was obtained for the existing physical four-pipe system.

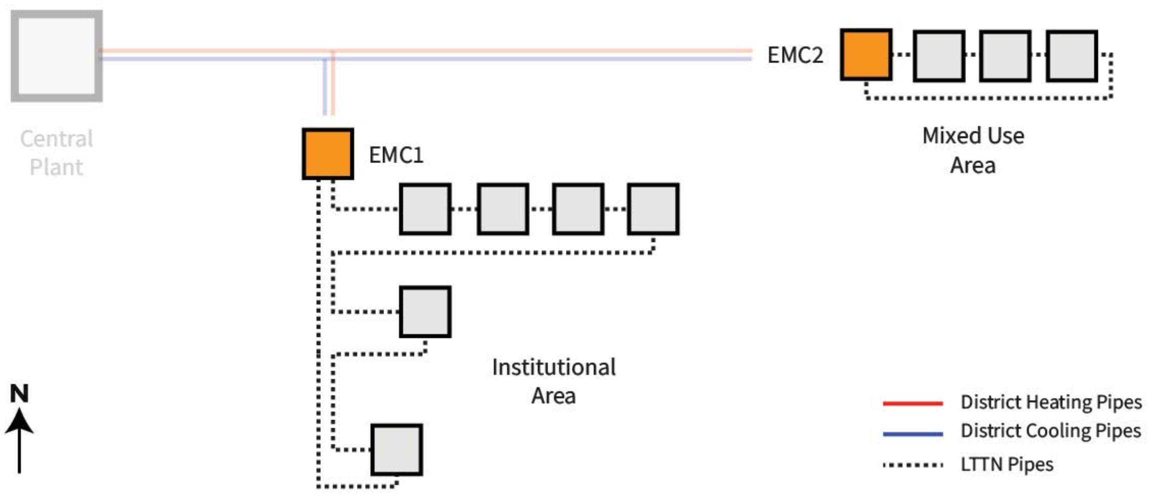


Figure 4. Low Temperature Thermal Network implementation

This included all pipe lengths, equipment efficiencies and the building load data within the system. This data, captured at five-minute intervals, was then used to reconstruct the physical plant as a Modelica model and comparisons were then made between EMC performance of the simulated model and the physical plant.

The existing system observes a 24% loss in energy from the total energy generation of the system at the central plant, to where the energy is delivered to the building loads. The system was simulated in Modelica with the library described in Section 2. This indicated that approximately half of the total losses (13%) were heat loss from the buried pipes to the environment, using expected values for the thermal conductivity of the soil and heat exchanger efficiencies. Both the thermal conductivity of the soil and the heat exchanger efficiency were tuned to assess the sensitivity of the error. This showed that while step changes in ground thermal conductivity had little effect on the pipe losses, the heat exchanger efficiencies could potentially account for the error. This led to the decision that to best address the remaining 11% error more experimental work is needed to determine the true nature of these losses before the library can be better tuned.

4 LTTN Implementation

To showcase an example of a LTTN implementation, the same case study that was used to validate the four-pipe District Energy System was decomposed into two communities that were then modeled with two, one-pipe LTTNs (Figure 4).

4.1 Design and Layout

In this layout, the two distinct areas outlined with the District Energy System have been separated and outfitted with their own LTTN systems. Both of these systems consist of individual one-pipe networks. For this example, the distribution pipe was modelled using the insulated pipe present in the physical system (15.24 cm pipe diameter, 24.51 cm insulation diameter). Thermal losses to the environment were taken into account, as with the observed data for the four pipe District Energy case.

5 Results and Discussion

The results from operating the two different systems for a two-week period in October of 2017 are discussed in this section. It should be noted that there are simultaneous demands for heating and cooling in this month, typical of October in Southern Ontario, Canada. For this comparison, the pipe specifications outlined in the previous sections were used. For both the District Energy System and the LTTN, the heating production was supplied using a boiler and cooling production was supplied with a chiller regardless of peak timing. For the two different areas in the LTTN, the average thermal energy requirements throughout the two-week period are outlined in Figure 5 with the total community energy requirements equal to approximately 52.96 MWh for the period. Both

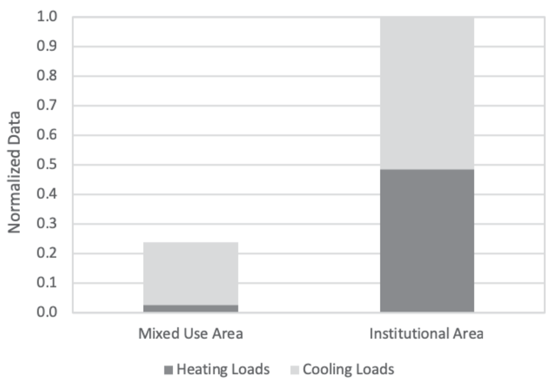


Figure 5. Energy demand distrubution for the case study community

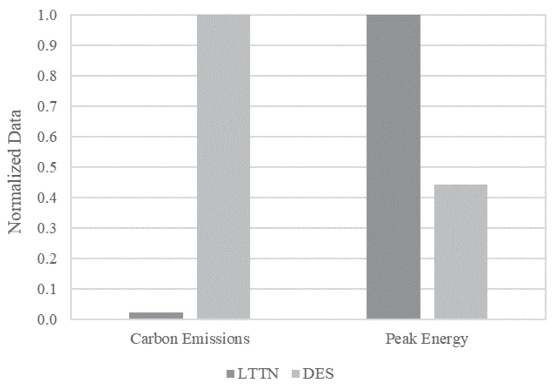


Figure 6. Carbon emission and peak power comparisions

systems were then compared based on total power consumption, total electrical costs and carbon emissions.

5.1 Total Energy Usage:

5.1.1 Peak Electricity Usage

The total peak electrical power use for this two-week period in October with the traditional four-pipe District Energy system is 5.8 MWh, while in the case of the LTTN it is 13.0 MWh (Figure 6). This is due to the additional heat pump utilization with the LTTN.

5.1.2 Total Resource Usage

The four-pipe system uses 1.2 MWh of energy in the form of electricity for pumping and requires 9.2 MWh of electricity for cooling. Additionally, it also uses 23.0 MWh of thermal energy in the form of natural gas for heating at the centralized plant.

Comparatively, the LTTN requires 0.7 MWh of electricity for pumping, 5.4 MWh of electricity for cooling and 0.8 MWh of natural gas energy for heating. Additionally, the system utilized 15.1 MWh of electricity for the heat pump operations, however this additional electrical energy made the system more efficient and resulted in a 34% total energy reduction when compared to the District Energy System.

Figure 7 and Figure 8 show the dynamic energy generation of both the DES and LTTN system respectively. Although the LTTN utilizes 34% less energy, it still has high generation periods equivalent to that of the DES system during periods where cooling is dominate. This 34% decrease in energy is a direct result of the energy sharing which is taking place between capturing rejected heat from cooling loads, and conversely for heating loads.

Additionally, the LTTN model also exhibited much lower pipe losses than the DES model. Using the same insulated piping geometry for both cases, the lower temperature set point of the LTTN (25°C) led to a 95% reduction in pipe losses when compared to the higher DES set point (75°C).

Figure 7 also gives insight into the nature of the mixed loads within this DES. Although the community has almost equivalent heating and cooling generation requirements, these generation periods are offset by a standard 12-hour period.

5.2 Carbon Dioxide Emissions

Considering the carbon dioxide emissions, it was found during the transitional month of October, due to a mix of heating and cooling demands (Figure 5), the LTTN had much lower carbon emissions than the traditional District Energy System (Figure 6). This stems from the heat pumps ability to reduce total centralized plant production by capturing rejected thermal energy from periods that require both heating and cooling. During periods of heating demand, the heat pumps within the LTTN provide heating and in return cool the thermal network. Similarly, during periods of cooling demand, the heat pumps provide cooling and in turn heat the thermal network. This rejected energy coupling reduces the total production at the centralized plant which results in lower emissions from natural gas

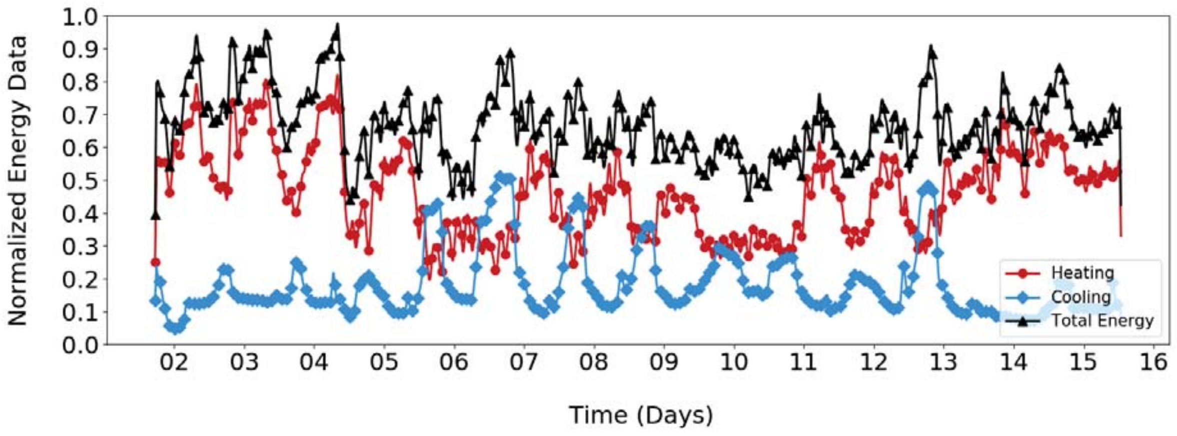


Figure 7. DES Energy Generation Comparison: Heating (Natural Gas Heating Load), Cooling (Electrical Load)

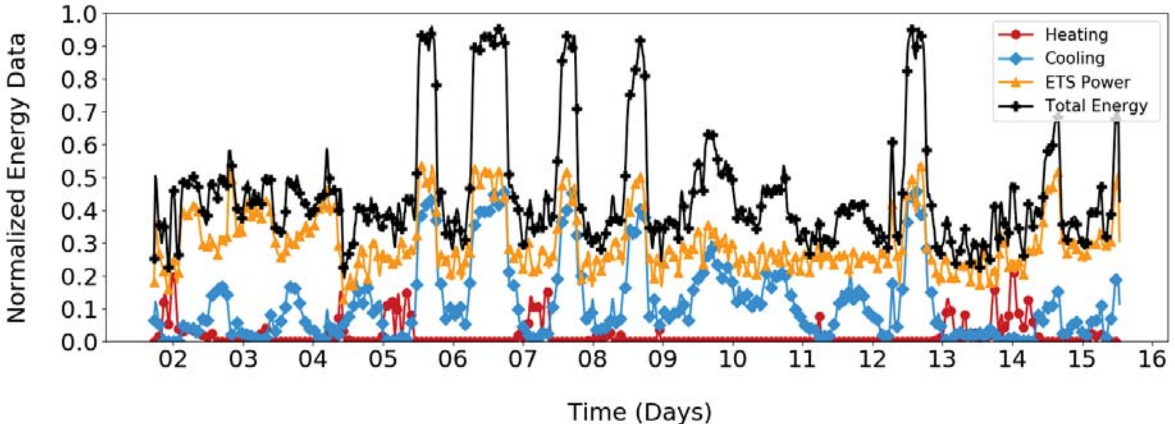


Figure 8. LTTN Energy Generation Comparison: Heating (Natural Gas Heating Load), Cooling (Electrical Load), ETS Power (Electrical Load)

heating. This reduction remains true even when accounting for the additional carbon dioxide production from using heat pumps when there is fossil fuel generation on the electrical grid during peak times. During these peak periods the impact of the waste energy recovery benefit still reduces carbon emissions by at least 56% based on calculations from Environment Canada (2017) (51 kg CO_2/Gj).

6 Conclusions

A library for modelling Low Temperature Thermal Networks has been developed to help characterize the performance of these new energy systems. The library consists of six unique models for the simulation of both traditional four-pipe District Energy Systems and one-pipe LTTNs. The library models were used to simulate an existing four-pipe District Energy System which provided real energy data for both energy demand and generation for a case community. This comparison indicated an 11% error between the predicted values and the historical values of the thermal losses in the distribution grid, an error that is being addressed through additional experimental analysis.

Beyond this comparison, a LTTN design was then implemented in lieu of an existing District Energy System in a nine building, high density community to demonstrate the effectiveness of the LTTN. By simulating the LTTN, it was found that although the single pipe design's integration of heat pumps increases the peak electrical power requirements of the community by 7.2 MWh, the waste energy recovery potential of the system can reduce the carbon emissions for the system by at least 56%. Additionally, the system reduced the total energy utilization by 34% by utilizing electrical energy to improve the efficiency of the heating and cooling process.

7 Future Work

To further improve the Modelica library presented, some revisions are still necessary to account for the additional 11% energy losses in the validation case. These steps include experimentally testing the DES system to determine the nature of these losses and then adjusting the Modelica library to account for them.

Acknowledgements

Natural Sciences and Engineering Research Council of Canada, Ontario Centres of Excellence and the team from HCE Energy Inc., Shawn Forbes, Ankur Mehrotra and Savan Trivedi.

References

Felix Bünning, Michael Wetter, Marcus Fuchs, and Dirk Müller. Bidirectional low temperature district energy systems with agent-based control: Performance comparison and operation optimization. *Applied Energy*, 209, pp. 502-515, 2018.

Mengyu Li, Xiongwen Zhang, Guojun Li, and Chaoyang Jiang. A feasibility study of microgrids for reducing energy use and GHG emissions in an industrial application. *Applied Energy*, 176, pp. 138-148, 2016.

Henrik Lund, Sven Werner, Robin Wiltshire, Svend Svendsen, Jan Eric Thorsen, Frede Hvelplund, and Brian Vad Mathiesen. 4th Generation District Heating (4GDH) Integrating smart thermal grids into future sustainable energy systems. *Energy*, 68, pp. 1-11, 2014.

D. Müller, M. Lauster, A. Constantin, M. Fuchs, and P. Remmen. AixLib - An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework. *BauSIM*, pp. 3–9, 2016.

P. Wallentén. Steady-state heat loss from insulated pipes. *Byggnadsfysik LTH, Lunds Tekniska Högskola*, pp. 13-15, 1991.

Environment Canada, Greenhouse Gas Sources and Sinks in Canada Part 2: Canada's Submission to the United Nations Framework Convention on Climate Change. *National Inventory Report 1990–2015*, 2017.

SESSION 5B: POWER & ENERGY 4

Robust Calibration of Complex ThermoSysPro Models using Data Assimilation Techniques: Application on the Secondary Loop of a Pressurized Water Reactor

Luis Corona Mesa-Moles[1] Jean-Philippe Argaud[1] Audrey Jardin[1] Amine Benssy[1] Yulu Dong[1]

[1]EDF R&D, France, {luis.corona-mesa-moles, jean-philippe.argaud, audrey.jardin, amine.benssy, yulu.dong}@edf.fr

Abstract

ThermoSysPro (TSP) is a library for the modeling and simulation of power plants and energy systems. It has been developed by EDF and it is released under open source license. When developing models with TSP it is necessary to ensure that they match reality. In practice, this operation is performed by adjusting the value of the parameters appearing in the model. This major step corresponds to model calibration.

Calibration can be performed through various methods. A classical way to do so with Modelica models is by model inversion. The major inconvenience of this method, in addition of potential convergence problems for complex models, is that it is necessary to have exactly the same number of measurements as parameters to be calibrated, which is not often the case in practice.

This paper shows how data assimilation techniques can robustly be used for calibration of complex TSP models avoiding the inconveniences associated to calibration by model inversion while ensuring an optimal use of the available measurements. A complex TSP model of the secondary loop of a Pressurized Water Reactor (PWR) is considered for this purpose.

Keywords: Modelica, ThermoSysPro, ADAO, data assimilation, model calibration, thermal-hydraulics, pressurized water reactor.

1 Introduction and context

Physical models of energy systems such as power plants can be advantageously used for the engineering of these systems all along their lifecycle from the design phase till the operation phase. They can be employed to test different design or retrofit alternatives, to evaluate the impact of changes in safety or environmental rules, to validate the performance of new components during their commissioning, to train operators, or even to help diagnose component's failures or sensor's drifts during operation and predict the system evolution in these conditions.

Modelica (Modelica, 2018) is a language perfectly suited for this kind of modelling thanks to its equation-based and acausal features:

(1) The engineer can use physical equations to capture in the same model the different phenomena governing the system behavior from the mechanical, hydraulic, thermal, electrical, and so on points of view;
(2) The equations are expressed in an acausal (i.e. non-oriented) way such that the engineer can reuse the same model for different computation purposes. From the same equation, one may, for instance, deduce the perfect sizing of a component to match a given operating point or compute the resulting operating point given the characteristics of on-shelf component.

A generic Modelica library, called ThermoSysPro (TSP), has been developed by EDF to model and simulate power plants and other kinds of energy systems. It is released under open source license and freely distributed with the OpenModelica simulation tool (OpenModelica, 2018) downloadable here: https://openmodelica.org/download/download-windows#.

Numerous organizations and individuals worldwide now use TSP and a large spectrum of use-cases exist from nuclear, thermal, to combined-cycle through biomass or even concentrated solar plants (El Hefni B. and Bouskela D., 2017).

In the design phase, the engineer has no other choice than calibrating such models with design assumptions and theoretical performances of each component issued from manufacturer data.

In the operation phase, when measurements within the modelled system are available, it is possible to use them to calibrate the model. One way to perform calibration is by model inversion which consists in computing the values of *n* parameters that deterministically correspond to a given set of *n* measurements. Model inversion can be performed using the Modelica feature to express inverse problems. This method gives satisfying results but it can be difficult to implement in practice for complex models. The main drawbacks associated to this method are:

- the necessity to readapt some part of the model in order to express the inverse problem; in some cases it may require to develop new modules facilitating the convergence of the inverse model;
- the necessity to consider exactly the same number of measurements as of parameters to calibrate (which does not happen often);
- the fact that the different measurements are considered homogenously (even if they have not been obtained in the same conditions, or if the model is not intended to be representative of all the available measurements in the same way);
- no consideration of measurement uncertainties (including on the boundary conditions of the model).

Data assimilation framework (Asch M. et al., 2016; Bouttier B. et al., 1999; Kalnay E. et al., 2003) provides a number of alternative methods and techniques that can be used to overcome these difficulties during model calibration.

For illustration purposes and to better understand the main differences between the two approaches, hereafter is presented a calibration problem of a simple TSP model, for more details see *Modeling and simulation of a complex ThermoSysPro model with OpenModelica* (El Hefni B. and Bouskela D., 2017). The model is presented in Figure 1. It corresponds to a singular pressure loss module with given boundary conditions (in this case the inlet and outlet pressures).

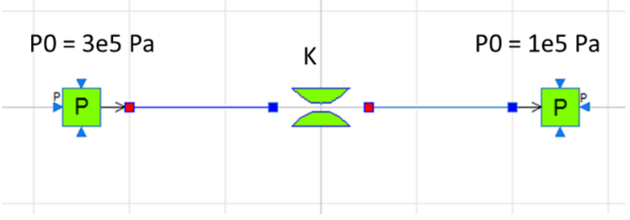


Figure 1. Model of singular pressure loss

The calibration of the model consists in determining the value of the pressure loss coefficient (K) of the pressure loss module. The measurement available to perform this calibration corresponds to the mass flow rate through the pressure loss module (Q).

For the calibration by model inversion, the observed mass flow rate is directly used to compute the exact value of the pressure loss coefficient since both appear in the same physical equation.

The corresponding physical equation is presented below:

$$P_i - P_o = K \cdot \frac{Q \cdot |Q|}{\rho}$$

P_i and P_o are the fluid pressure at the inlet and at the outlet of the singular pressure loss respectively, ρ is the average density of the fluid, Q is the mass flow rate and K is the friction pressure loss coefficient.

In the calibration using data assimilation techniques, the approach is different. From the physical knowledge of the system it is possible to give a guess value to the K coefficient (or use directly the default value of the TSP library), this corresponds to the *a priori* value of the parameter to be calibrated. This *a priori* value is used as a starting point and will be iteratively corrected to find the best value of the calibrated parameter, "best" in the sense that the results given by the model should be in the end as close as possible to the available measurements.

The objective of the article is to show how data assimilation techniques can be used in general to have a more robust approach of the calibration phase.

It illustrates on an industrial-size use-case, which is the model of the secondary loop of a pressurized water reactor, what are the concrete benefits of this approach compared to the traditional one in place using model inversion.

2 Model of the secondary loop of a PWR

2.1 Nuclear power plant performance monitoring

The secondary loop of a 1300 MW PWR nuclear power plant has been modelled with TSP modules in order to determine the best efficiency rate that can be expected from the thermo-hydraulic cycle, given various boundary conditions. This theoretical best efficiency operation setpoint gives an estimation of several physical quantities like pressures and temperatures across the cycle. They are the references against which the on-site measurements will be compared, allowing to identify any deviation causing energy losses. These symptoms will then be processed in order to identify their potential causes.

The more accurate the model is, the better the diagnosis will be.

2.2 Model description

Secondary loops of PWRs are classical Rankine cycles that convert thermal energy into electrical power.

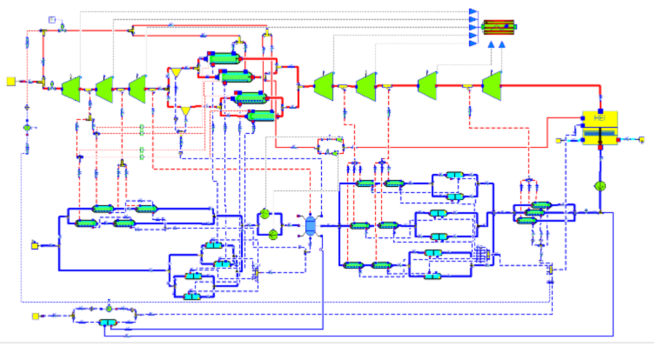


Figure 2. Model of 1300 MW PWR secondary loop with TSP

The TSP model developed to represent such PWR's secondary loop is static and composed of the following key systems (Figure 2):

- a turbogenerator set made of high-pressure (HP) and low-pressure (LP) 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 (LP) and high pressure (HP) feedwater heaters.

Once properly calibrated, the model calculates the nominal operation setpoint from thirteen boundary conditions. Among which the more important are: plant's cooling water temperature and pressure, Steam Generator's (SG) thermal power, SG's moisture carryover level, SG's pressure at the outlet, SG's feedwater flow.

3 Calibration methodology

3.1 Data assimilation framework

Data assimilation is a general well established framework (Asch M. et al., 2016) for computing the optimal estimate of the true state of a system, over time if necessary. It combines knowledge between observations and *a priori* models, including information about their errors. The goal is to obtain the best possible estimate of the system real state and of its stochastic properties. Moreover, data assimilation provides deterministic techniques in order to perform very efficiently the estimation job. Because data assimilation looks for the best possible estimate, its underlying procedure always integrates optimization in order to find this estimate.

The calibration of a model consists in looking for the value (of part) of the parameters of a model, in such a way that the simulation obtained with these parameters is better adapted to real measurements on the same simulated system, in the sense that the distance between model predictions and measurements is smaller. The use of data assimilation for calibration requires the acquisition of measured information in the same conditions under which the simulated system is to be calibrated. The collection and prior analysis of these measurements also establishes elements of confidence and compared quality of the measurements, which will be interesting in the use of algorithms. In addition, the numerical model used must be functional over a validity domain that includes the range of variation of the parameters to be calibrated.

All quantities representing the description of physics in a model are likely to be calibrated in a data assimilation process, whether they are model parameters, initial conditions or boundary conditions. Their simultaneous consideration is greatly facilitated by the data assimilation framework, which makes it possible to objectively process a heterogeneous set of available information.

3.2 Data assimilation applied to 0D/1D models with ADAO

To perform data assimilation, a specialized LGPL free distributed tool ADAO (Salome, 2018) is used to simplify the application of data assimilation for the simulation of complex systems. Available in the Python environment that allows the simulation of Modelica models and hence of TSP models, it allows to easily automatize the calibration of 0D/1D models and the development of complex calibration scenarios according to the states of the analyzed physical system. ADAO was initially developed to perform data assimilation with 2D/3D models. Its adaptation to 0D/1D models has been coded during this work and now simplifies the specification of model parameters to be calibrated and simulated quantities to be compared to measurements, which are known in the Modelica description of the system. In addition, an advanced and simultaneous management of the various possible operating conditions enhances the physical representativeness of the overall calibration of the simulated system.

The use of ADAO in a Modelica/Python environment allows to simply describe the data assimilation problem, through a Modelica representation of the simulation and of the named data for measurements as well as for the *a priori* values of the parameters to be calibrated. Since the different measurements are not obtained with the same sensors, the confidence accorded to the different available measurements can be easily modified as well. Moreover, when calibrating a large number of parameters, a sensitivity analysis can be performed to reduce the set of parameters that should be calibrated to the only ones that have a real impact on the quantities observed through the measurements. The entire data assimilation process is then automated and depends only on the ability of the model to simulate the system for the required parameter values through optimization. The stability and convergence of the simulated system over its entire domain of validity are therefore essential to allow an efficient search for a set of calibration parameters. The availability of complete or aggregated outputs provided by ADAO for a simulation ensemble is also crucial to ensure that the optimal simulation can be analyzed in detail and that the calibrated parameters are relevant.

3.3 Calibration procedure

The specialized tool ADAO allows to easily define the different elements necessary to perform model calibration using data assimilation techniques. These elements are:

- Parameters (including initial conditions or boundary conditions if required) to be calibrated (with given *a priori* values);
- Available measurements (taking into account whether they have been obtained under the same conditions, i.e. with the same boundary conditions, or not);
- Modelica model (describing the physical connection existing between the parameters to be calibrated and the observations/measurements).

It is important to note that a variable confidence error can individually be associated to the different measurements available, under the form of a covariance matrix. This information is then used by ADAO to compute the optimal values of the parameters. This process is illustrated in Figure 3 (in blue the necessary information to be provided to ADAO).

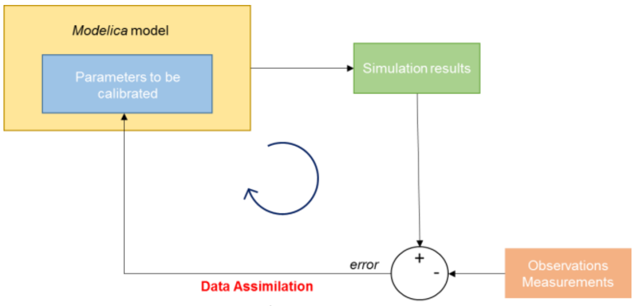


Figure 3. Illustration of the calibration procedure using data assimilation techniques

3.4 Analysis criteria

In order to evaluate how good a calibration of the model is, it is necessary to establish a certain number of criteria. In a model calibration procedure as here, the objective is that the variables computed by the model are as close as possible to the available measurements. Therefore, these indicators should be based on the differences between the available on-site measurements and the corresponding variables computed by the model.

In this paper, two different criteria are considered. Firstly, a global indicator that it is equal to the sum of the quadratic difference between the measurement and the corresponding variable in the model, for all the available measurements. This indicator is not very different from the cost function value minimized by ADAO in the calibration procedure. It provides a general overview of a given calibration. Secondly, an indicator for each measurement, considered individually, is necessary in order to detect a calibration that is unacceptable (i.e. outside of its *a priori* confidence interval) for a given measurement while being correct globally. For each measurement, the relative difference between this measurement and the corresponding model output is computed. The relative difference is defined as the absolute value of the ratio between the difference between the measurement and the model output and the value of the measurement. It provides therefore a homogenous indicator for all the available measurements. This indicator can as well be used to check precisely if the calibrated model is more representative for certain measurements, judged to be more important than the others.

4 Calibration of the secondary loop of a PWR

4.1 Scenario description

The calibration of the TSP model of PWR's secondary loop (Figure 2) is performed over 116 parameters. For comparison purposes, the same observations as for the calibration by model inversion are considered. They correspond to pressure, mass flowrate and temperature measurements. These observations correspond to 116 measures obtained in ten campaigns of measurements. Therefore, in total 10x116=1160 observations are used to perform the model calibration. However, contrary to the calibration by inversion, the observations are considered simultaneously for the calibration presented in this study. The use of ADAO and preprocessing facilities allows to adapt the boundary conditions for each set of 116 observations.

The calibration of the model is performed for two different configurations, which mainly differ from the *a priori* values given to the parameters as an initial guess. In the first configuration the *a priori* values for the parameters to be calibrated correspond to the values obtained by model inversion. In the second one, typical *a priori* values are considered, corresponding to what can be found in technical data sheets of the modeled components. This second case would correspond to a typical calibration procedure while the first one shows how data assimilation methods can improve the calibration obtained by the classical model inversion method which is now in current use in our engineering divisions.

For the first configuration, the following sub-scenarios are studied:

- High confidence on observations (scenario 1);
- High confidence on observations but according more confidence to some of them that are considered as more meaningful (scenario 2).

For the second configuration, the following sub-scenarios are considered:

- High confidence on observations (scenario 3);
- High confidence on observations but considering a reduced number of parameters to calibrate (the selection of these parameters is performed through a sensitivity analysis, 62 parameters are kept) (scenario 4).

For each scenario, the domain in which the optimal value of the parameters is searched is adjusted in order to ensure the convergence of the simulated model (see

3.2 for more details). These research domains are indicated in Table 1.

Table 1. Research domain for the optimal value of calibrated parameters.

Scenario	*Research domain*
Scenario 1	5% around *a priori* values
Scenario 2	5% around *a priori* values
Scenario 3	10% around *a priori* values
Scenario 4	60% around *a priori* values

It appears clearly that reducing the number of parameters to calibrate enables to enlarge the research domain for the optimal value of the parameters: the convergence of the model is facilitated compared to the situation in which all the parameters have to be calibrated and may vary.

4.2 Results and discussion

First of all, it is important to examine the optimal value of the parameters given by the data assimilation procedure. A key point is to check if the optimal value of the calibrated parameter reaches the bounds of the research domain. In such case, it is probable that right optimal value of the parameters is not reached. If there were no limitations, or if the non-convergence situations could be avoided, the calibration of the model would be more trustful. Table 2 summarizes this aspect for the scenarios described in the previous section: it indicates the number of times that the bounds of the research domain for a given parameter are reached.

Table 2. Number of times the bounds of the parameters research domain are reached.

Scenario	*Number of times the bounds of the research domain are reached*
Scenario 1	3
Scenario 2	1
Scenario 3	73
Scenario 4	9

For the scenarios in which the starting values of the parameters is the one obtained by model inversion (Scenarios 1 and 2), it can be checked that, even with a small research domain, the bounds are rarely reached. This seems logical as the value of the parameters obtained by model inversion is supposed to be close to an optimal value. For the scenarios in which typical *a priori* values are considered, the bounds of the domain research are often reached when all the parameters are kept. If the number of parameters is reduced and the research domain is enlarged (as in scenario 4 with respect to scenario 3), reaching the bounds is largely reduced: 63% of calibrated parameters reach the bounds in scenario 3 compared to only 15% in scenario 4.

In order to evaluate how good the calibration is, an overall indicator is the quadratic difference between the observations and the model output (for the 1160 observations), see paragraph 3.4. The smaller this quadratic difference is, the better the calibration is. Table 3 summarizes this result for the four scenarios studied in the present work and for the calibration performed by model inversion, the so-called *Inverse calibration*. The results are presented based on the result obtained for the *Inverse calibration* method (a value lower than 1 indicates that the result is better than the result obtained by model inversion and a value higher than 1 indicates that it is worse).

Table 3. Quadratic difference between the observations and the model output – Inverse calibration as a reference

Scenario	*Quadratic difference*
Inverse calibration	1 – Reference result
Scenario 1	0.166
Scenario 2	0.245
Scenario 3	4.483
Scenario 4	0.167

In Table 3, the most important point is the value of the quadratic difference compared to the one obtained by model inversion that is set to 1 for comparison purposes. For scenarios 1 and 2, results show that this difference is largely reduced (almost by a factor from 5 to 10, especially for scenario 1). This show how data assimilation can improve an existing calibration.

For scenarios starting from typical *a priori* values, the results are very encouraging as well. When all the parameters are considered (scenario 3), the quadratic difference is only a few times higher than the one obtained by model inversion. However, when a fewer number of parameters are kept but with a larger variation range as indicated in Table 1 (scenario 4), Table 3 shows that the quadratic difference is much smaller than the one obtained by model inversion: similar results as for scenario 1 are obtained. This shows how important it is to ensure the model convergence in a domain as large as possible (scenario 3 should give better results than scenario 4, however as indicated in Table 2 for scenario 3 a large number of parameters reach the bounds of their research domain).

In addition of the overall overview of the calibration, it is important to ensure that the calibration provides good results for each observation separately, avoiding for example to reduce the error obtained for one single observation and increasing it for a large amount of them. As presented in section 3.4, a good indicator can be for example the relative difference between a given observation and the corresponding model output. Table 4 shows how many times this relative difference (averaged over the ten campaigns of measurements) is minimal, with a certain tolerance, for the 116 observations.

These results show that this indicator is improved (or at least not worsened) when starting from the values of the parameters obtained by model inversion, especially for scenario 2. This shows that even observation by observation, considered separately, data assimilation techniques can improve the model calibration. For scenarios 3 and 4 it is shown that good results are obtained for a significant number of observations as well.

Table 4. Results with respect to the relative difference between a given observation and the corresponding model output.

Scenario	*Number of times that the relative difference between a given observation and the corresponding model output is minimal (with a tolerance of 10%)*
Inverse calibration	48
Scenario 1	48
Scenario 2	67
Scenario 3	27
Scenario 4	25

Finally, in order to illustrate the effect of modifying the confidence on certain observations, a focus is performed on the observations for which a higher confidence has been considered (in scenario 2, compared to scenario 1 in which all the observations were considered in the same manner). These results are not provided for scenarios 3 and 4 since no specific focus on these observations was performed. Table 5 summarizes these results. The observations for which a higher confidence has been given, i.e. considered as more meaningful, are numbered from 1 to 16. For these observations, the relative difference between the observations and the corresponding model output (averaged over the ten campaigns of measurements) is indicated for the calibration by model inversion and for scenarios 1 and 2. For each observation, the minimal relative difference is put in bold. Moreover, the last line of Table 5 indicates the quadratic difference obtained over this subset of observations (as in Table 3, the results are given with respect to the results obtained with the calibration method by model inversion).

Table 5. Comparison between observations and model output for the observations on which a higher confidence is given.

Observation number	*Inverse calibration*	*Scenario 1*	*Scenario 2*
1	1.66%	1.54%	**1.33%**
2	**1.51%**	**1.51%**	1.54%
3	1.50%	1.39%	**1.31%**
4	0.74%	1.09%	**0.44%**
5	**4.28%**	4.30%	4.33%
6	**0.85%**	0.99%	0.89%
7	0.36%	0.40%	**0.34%**
8	0.41%	0.32%	**0.29%**
9	0.33%	0.23%	**0.21**%
10	0.37%	0.25%	**0.23**%
11	**0.11%**	0.20%	0.18%
12	0.14%	**0.10%**	**0.10%**
13	0.67%	0.72%	**0.59%**
14	1.05%	0.87%	**0.83%**
15	1.07%	0.88%	**0.82%**
16	1.42%	**1.16%**	1.31%
Overall quadratic difference	1 - Reference	0.822	0.676

Table 5 shows that scenario 2 provides better results for a large part of these observations considered individually (and when this is not the case, the relative difference is still very close to the one obtained by inverse calibration or in scenario1). Moreover, the overall indicator, giving the quadratic difference for this subset of observations, shows clearly that in both cases (scenario 1 and 2) the overall results obtained by data assimilation techniques are better than those obtained by model inversion. Therefore, it is possible, using data assimilation techniques, to easily obtain different calibrations of the model according to what the model is intended for or according to the quality of the observations.

These results show how the application of data assimilation techniques for the calibration of complex TSP models can give good calibration results, both in providing or in improving the optimal value of the calibrated parameters. Moreover, these calibration results can be obtained in about one day of calculations, compared to several weeks for the calibration by model inversion currently required (including the development of an inverse model, the pre-treatment of the measurements initially available and the different post-treatment techniques required to determine the optimal value of the parameters).

5 Conclusion and perspectives

A new method for robust and reliable model calibration, based on data assimilation techniques, for complex TSP models is currently under development. It already shows its important benefits compared to the traditional method using model inversion.

The results presented in this paper show how the application of data assimilation techniques to calibrate a complex TSP model of the secondary loop of a PWR is able to improve the calibration obtained by model inversion. In addition, it shows how a usual calibration procedure using these new techniques, coupled with a sensitivity analysis of the model, can as well provide better results than the traditional calibration method.

Therefore, compared to calibration by model inversion, this new method enables to handle conveniently situations that could not be treated before, or that would have required an important number of pre-treatments. For example, when more measurements than parameters to be calibrated are available, with the calibration by model inversion method, it was necessary to make a choice and loose some information, whereas with the new method presented in this paper it is not necessary. On the contrary, if not enough measurements are available it is not possible to calibrate the whole set of parameters using the traditional inversion method, whereas calibration approach based on data assimilation techniques is able to provide an optimal value for the whole set of parameters using efficiently all the available information. In addition, the consideration of measurements obtained in different operating conditions is greatly facilitated by data assimilation since they can all be considered simultaneously (it is therefore not necessary to post-treat independently the results obtained individually by one model inversion per each operating condition or campaign of measurements). Moreover, for some complex models, calibration by model inversion requires to develop new inverse modules when the convergence for inverse calculation is difficult, which may be very time-consuming.

In other word data assimilation method allows to automatize the model calibration procedure and hence to considerably reduce the time necessary to its calibration. Furthermore, it paves the way to improve the calibration accuracy, by enabling the use of additional information (e.g. more measurements than those strictly necessary for calibration by model inversion), or the use of available information in a specific way (e.g. according more confidence to some measurements). However, it is important to keep in mind that a good knowledge of the modelled system and of the model itself is very important in order to ensure that the results obtained applying data assimilation techniques are physically correct.

In the future, the current improvements under development should facilitate the application of this new calibration method. A major aspect is to ensure the convergence of the model over a large domain so that data assimilation techniques can provide even better results. Work on model initialization will in particular be done within the ongoing FUI ModeliScale project in partnership with Dassault Systèmes, INRIA and Phiméca. Other important point is from the methodological point of view to study: (1) how complementary studies such as sensitivity analysis of the model can be used more efficiently to properly formulate the calibration problem (e.g. by considering for calibration only the parameter that have a real impact on the variables of interest considered); (2) how data assimilation could be used for other purposes such as state estimation or prognosis.

References

Asch M., Bocquet M., Nodet M., Data Assimilation - Methods, Algorithms and Applications, SIAM, 2016.

Bouttier B., Courtier P., Data assimilation concepts and methods, Meteorological Training Course Lecture Series, ECMWF, 1999.

El Hefni B., Bouskela D., *Modeling and simulation of a complex ThermoSysPro model with OpenModelica – Dynamic Modeling of a combined power plant*, 12th International Modelica Conference, May 15-17, 2017, Prague, Czech Republic.

Kalnay E., Atmospheric Modeling, Data Assimilation and Predictability, Cambridge University Press, 2003.

Modelica, open-source modelling language, information available at: https://www.modelica.org/

OpenModelica, open-source Modelica-based modeling and simulation environment, information available at: https://openmodelica.org/

SALOME The Open Source Integration Platform for Numerical Simulation, information available at: http://www.salome-platform.org/

Coupling Power System Dynamics and Building Dynamics to Enable Building-to-Grid Integration

Yangyang Fu[1] Sen Huang[2,*] Draguna Vrabie[2] Wangda Zuo[1]

[1]Department of Civil, Environmental and Architectural Engineering, University of Colorado Boulder, Boulder, CO, USA {yangyang.fu, wangda.zuo}@colorado.edu
[2]Pacific Northwest National Laboratory, Richland, WA, USA {sen.huang, draguna.vrabie}@pnnl.gov

Abstract

The interactions between power system dynamics and building dynamics are usually ignored or over-simplified in existing power system and building modeling and simulation tools, which limits how system modeling can support Building-to-Grid integration. This paper discusses a new approach to consider those interactions by modeling motor-driven building devices or systems. The motor-driven model is based on simplified mechanical rotation equations and allows us to study the coupling relationship between frequency/voltage in the power system and motor-driven device operation. This model is validated by performing one proof-of-concept case study with Modelica. The simulation results suggest that the proposed model can yield better representations of these interactions than the existing simplified models, especially the ones with the fast transient dynamics.

Keywords: Building-to-Grid Integration, Motor, Coupling Simulation

1 Introduction

There are significant interactions between power system and the buildings. For example, research suggests that the supply voltage of the power system dramatically affects building energy efficiency (Hood, 2004; Bichik et al., 2015; Lee, 2014). On the other hand, building systems, especially heating, ventilation, and air conditioning (HVAC) systems, attribute to the voltage stability issues in the power system (Wu et al., 2006; He et al., 2012; Li et al., 2017). It is generally necessary to consider those interactions, when designing or operating power systems or buildings, to avoid undesirable side effects.

However, when simulating the power systems and buildings, those interactions are usually ignored. For instance, some power system models assume the power factors of the building system to be fixed (Chassin et al., 2008). In addition, most building modeling tools implicitly ignore the influence of power systems on buildings by assuming the supply voltage to be constant (Crawley et al., 2001). Some models do consider those interactions, but in a simplified manner. One example is the ZIP coefficient model (Bokhari et al., 2014), which represents the variation (with voltage) of a load as a composition of the three types of constant loads: impedance “Z”, current “I”, and power “P” loads. It calculates the active/reactive power of a device under varying voltage conditions, by

$$P = P_o(Z_p(\frac{V}{V_o})^2 + I_p\frac{V}{V_o} + P_p) \tag{1}$$

$$Q = Q_o(Z_q(\frac{V}{V_o})^2 + I_q\frac{V}{V_o} + P_q) \tag{2}$$

where P ,Q, and V are the active power, reactive power, and voltage, respectively. Subscript o denotes the rated condition while p and q denote the active and reactive power, respectively. Essentially, the ZIP coefficient model approximates the influence of the voltage on the device with a polynomial function.

It is noted that the above assumptions or simplifications regarding the interaction between power systems and buildings may be justified for certain applications. For example, for buildings where the resistive devices (such as the electric heater) dominate, the assumption that the power factor is constant may be valid. In addition, when considering a regulated power system, it might be acceptable to assume that the supply voltage is constant in evaluating building design or operation (Gilbert, 1965). Furthermore, the ZIP coefficient model was widely used in the static or semi-dynamic analysis on the power system (Hatipoglu et al., 2012).

Nevertheless, it is our view that the aforementioned assumptions or simplifications may be inappropriate for some applications, especially the Building-to-Grid integration activities. For instance, (Arriffin et al., 2017) proposed an approach to increase the energy efficiency of the buildings by optimizing the supply voltage. When evaluating this approach, it is necessary to consider how the buildings respond to the varying voltage in terms of active/reactive power and the quality of the service they provided to the occupants. It is also necessary to consider how the proposed approach affects the stability of the power supply. Apparently, none of the aforementioned assumptions or simplifications help users to do so.

In this paper, we provide a more realistic representation of the interactions between power grid and buildings by presenting a new motor-driven model. The model is based on simplified mechanical rotation equations. It allows us to consider not only the effect of thermal dynamics in the building side on motor operations, but also the influence from the power supply side on motor behaviors. Thus, it is more suitable for the Building-to-Grid integration activities as we discussed above.

The rest of this paper is organized as follows: we first describe the motor system and simplify it for modeling purpose; then we elaborate the process for creating a Modelica model for the studied system; Last, we discuss the two proof-of-concept cases and future work.

2 Studied System

Figure 1 is a schematic of the studied system. The studied system is defined to represent how a typical HVAC system interacts with the power grid, and consists of three subsystems:

1) *Electrical Subsystem.* This subsystem represents the power system and provides power to the rest of the studied system. It has one variable frequency drive (VFD), which adjusts the frequency of power based on the request from the building system.

2) *Mechanical Subsystem.* This subsystem represents the process of converting to the service that building systems provide to occupants. In our case, the service we considered is cooling/heating demand. The mechanical system contains two components: an induction motor that creates torque given receiving power flow and a transitional device that converts the torque to the mechanical work.

3) *Thermal Subsystem.* This subsystem represents the HVAC system that addresses the cooling/heating demand in buildings. It has motor-driven devices such as a fan/pump that delivers fluid flows such as air or water flow to actually remove/add heat from/to the indoor environment. There are also feedback-loop controls which guarantee that the motor-driven device delivers desired flow rates by adjusting the frequency of the VFD.

It is noted that both the electrical subsystem and the thermal subsystem are treated as abstract interfaces to be connected to more detailed models for power systems or building systems. Details of the power system or the building systems such as the power distribution flow are beyond the scope of this study.

3 System Model

As discussed in the previous section, the electrical subsystem and the thermal subsystem are treated as interfaces to be connected to the power system and buildings, respectively. Thus, when elaborating on the system model, we mainly focus on the mechanical subsystem. For the details regarding the modeling of the power system or buildings, readers can find more information in (Chassin et al., 2008; Crawley et al., 2001).

In the mechanical subsystem, one major equipment is the induction motor. Although Modelica Standard Library has existing models for the induction machine, they are built on an electrical interface that are hardly compatible with Modelica Buildings library (Wetter et al., 2014) that is widely used to perform dynamic simulation on the building side. Whatmore, those induction machine model are too complicated for this preliminary study. Therefore, in this paper, we presented a new induction motor model which can be sufficient to capture the dynamics we need for Building-to-Grid integration, and utilized the electrical interface in Modelica Buildings Library to easily couple with building side.

In the presented induction motor model, the inputs include voltage, V, frequency, f, and load torque, τ_L, while the output is the electromagnetic torque,τ_e. The major parameters include the number of the pole pair, n_p, the number of the phase, n, the moment of the inertia, J_m, the electric resistance of the stator, R_s, the electric resistance of the rotor, R_r, the complex component of the impedance of the stator, the complex component of the impedance of rotor, and the complex component of the magnetizing reactance, X_s, X_r, and X_m.

τ_e is calculated by solving the following equations

$$\tau_e = \frac{n(V\frac{X_m}{X_m+X_s})^2\frac{R_r}{s}}{\omega_s((R_s(\frac{X_m}{X_m+X_s})^2+(\frac{R_r}{s})^2))^2+(X_r+X_s)^2} \tag{3}$$

$$\omega_s = \frac{4\pi f}{n_p} \tag{4}$$

$$s = \frac{\omega_s-\omega_r}{\omega_s} \tag{5}$$

$$\frac{d\omega_r}{dt} = \frac{\tau_e-\tau_L}{J_m} \tag{6}$$

In addition, the active and the reactive power of the motor, P and Q, are calculated by

$$P = \frac{nV^2R_{eq}}{{R_{eq}}^2+{X_{eq}}^2} \tag{7}$$

$$Q = \frac{nV^2X_{eq}}{{R_{eq}}^2+{X_{eq}}^2} \tag{8}$$

$$R_{eq} = R_s + \frac{R_r s X_m^2}{R_r^2+(s^2)(X_r+X_m)^2} \tag{9}$$

$$X_{eq} = X_s + \frac{X_m(R_r^2+(sX_r)^2+(s^2)X_rX_m)}{R_r^2+(s^2)(X_r+X_m)^2} \tag{10}$$

Regarding the transitional device, its inputs include τ_e and the load shaft power P_{shaft} while the outputs include the load speed, ω_r and τ_L. One major parameter for the transitional device is the load moment inertia J_L. τ_L is calculated by solving the following equations:

$$\tau_L = \frac{P_{shaft}}{\omega_r} \tag{11}$$

$$\frac{d\omega_r}{dt} = \frac{\tau_e-\tau_L}{J_L} \tag{12}$$

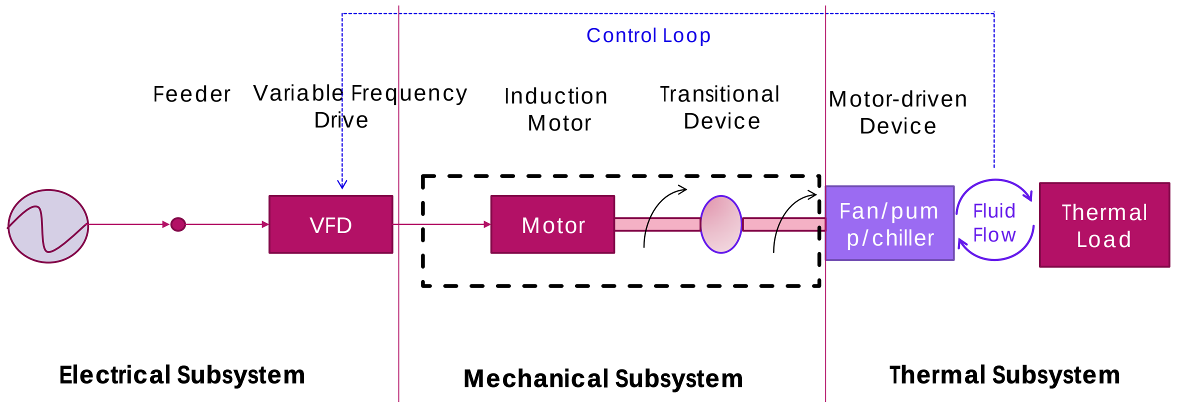


Figure 1. The studied system

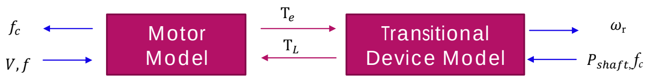


Figure 2. The input-output interface for the generated Modelica model

The load shaft power P_{shaft} can be calculated from a conventional pump model commonly used in building simulation tools.

$$P_{shaft} = \frac{\Delta p Q}{\eta_{shaft}} \tag{13}$$

Δp and η_{shaft} are the head and shaft efficiency of the pump. They can be expressed as a quadratic equation in terms of volume flowrate Q and normalized rotation speed r.

$$\Delta p = (a_0 + a_1(\frac{Q}{r}) + a_2(\frac{Q}{r})^2)r^2 \tag{14}$$

$$\eta_{shaft} = (b_0 + b_1(\frac{Q}{r}) + b_2(\frac{Q}{r})^2)r^2 \tag{15}$$

where the normalized speed r can be calculated by receiving rotation speed ω_r from the transitional device, and the known nominal rotation speed $\omega_{r,0}$.

$$r = \frac{\omega_r}{\omega_{r,0}} \tag{16}$$

We then implemented the above-mentioned models of the induction motor and the transitional device in Modelica (Fritzson and Engelson, 1998). Modelica is an equation-based modeling language that allows the systems to be described with implicit equations. Therefore, equations (3) to (12) can be used directly to create the corresponding Modelica codes. The generated Modelica models have the input-output interface shown as Figure 2. The inputs include the load shaft power and the frequency control signal, f_c, from buildings, as well as the voltage V and the actual frequency f from the power system; the outputs

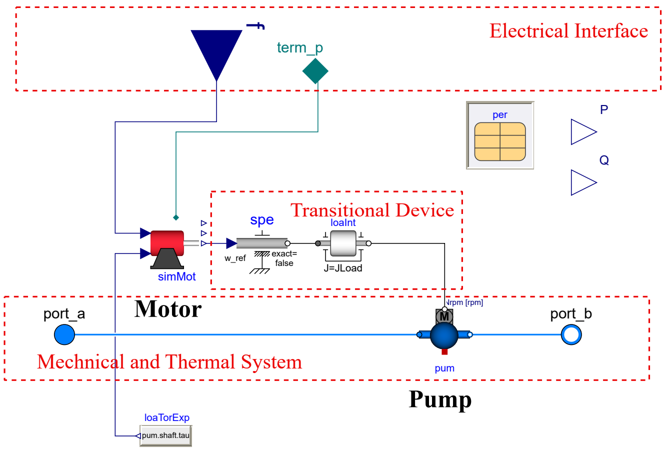


Figure 3. Modelica representation of the motor-driven pump model

are the actual speed of the motor-driven devices ω_r and the actual frequency control signal to the power system.

The final motor-driven pump model is built by connecting the motor model, transitional device, and a mechanical pump model in Modelica Buildings library together. The Modelica representation is shown in Figure 3. The electrical interface is used to connect with the grid model to receive electrical information. The received electrical information is then delivered to the induction motor model, which generates electromagnetic torque that is transmitted to the mechanical pump by the transitional device.

4 Proof-of-concept Study

We conducted one case study to demonstrate how the generated models can capture the interactions between the power system and the buildings. In this case, we considered a simplified hydraulic cooling system. As shown in 4, a pump on the supply side delivers cold water between an ideal cooling source and an air handler. The cooling source maintains the temperature of the leaving water to be 7 °C while the water flow rate is modulated to maintain the air leaving the air handler to be around 16 °C. The change in water flow rate is realized by adjusting the valve

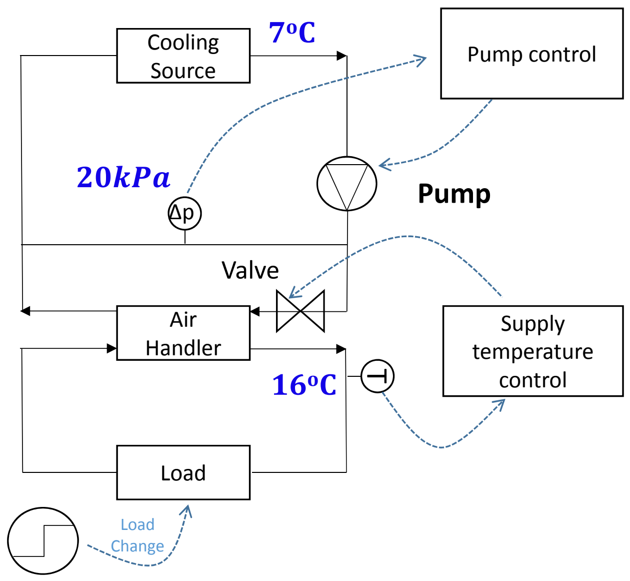


Figure 4. The schematic drawing of the proof-of-concept case

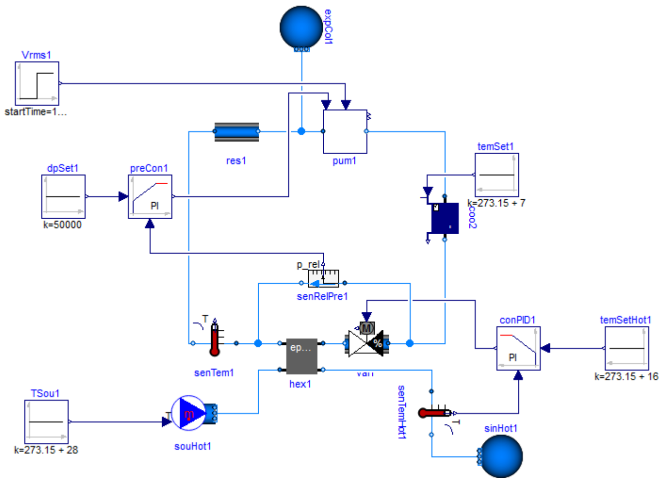


Figure 5. The diagram of the Modelica model in the proof-of-concept case

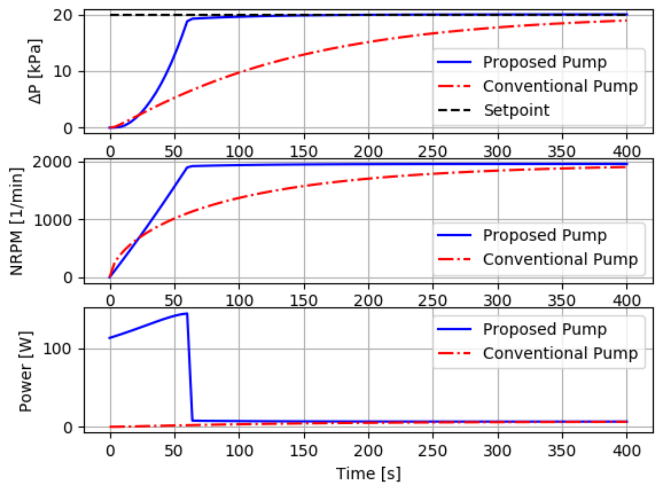


Figure 6. Simulation results for the first scenario

portion of a two-way valve. The frequency of the pump is adjusted to maintain a constant pressure difference in the pipe across the air handler.

We implemented the above case based on the Modelica Buildings library, which has been demonstrated to be able to perform building level energy modeling and simulation (Huang et al., 2016, 2017; Fu et al., 2018, 2019). For example, the *Buildings.Fluid.Sources.BoundarypT* is used to model the ideal cooling source. In addition, to model the pump, we consider three modules: One (named "conventional pump") is the pump module from the Modelica Buildings Library (*Buildings.Fluid.Movers.BaseClasses.PartialFlowMachine*). This model doesn't take voltage into consideration. The second one (named "proposed pump") is modified based on the first one by adding the developed induction motor and transitional device. Figure 5 shows a diagram of the generated Modelica model with the proposed pump in Dymola environment (Brück et al., 2002). The performance characteristics curve of the pump is from the Modelica Buildings Library.

We simulated the Modelica model for two different scenarios. In the first scenario, we studied how the system behaves when we change the status of the pump from off to on. In this case, as shown in Figure 6, for both the "conventional pump" and the "proposed pump", the pressure difference starts with 0 kPa and approaches the set point of 20 kPa. However, it takes around 60s for the "proposed pump" to be quite close to the set point while more than 400s for the "conventional pump" to reach to the similar value. On the same time, we also observe much higher pump power from 0 to 60s in the simulation of the "proposed pump". But as soon as the pressure is close to the set point, the simulated power from the "proposed pump" quickly drops to a value that is closer to that from the "conventional pump". It is our view that the simulated result from the 'proposed pump" is closer to what we observed in the real world, in terms of the power change.

In the second scenario, we studied how the system behaves when we change the supply voltage. To better evaluate the performance of the "proposed pump", we considered another option for modelling the pump as a reference. In this option, the induction motor and transitional device in the "proposed pump" are replaced with the ZIP coefficient model as described in equation 1 and 2. The resulting new model is named "ZIP pump". Figure 7 shows the simulation results for the second scenario. The supply voltage is kept constant before 100s. In that case, all three models generate identical results in terms of active power. At $t = 100$s, the voltage changes from 120V to around 108V; the active pump power in the "conventional pump" is unchanged. For the "ZIP pump", the active pump power immediately changes to a smaller value while that in "proposed pump" decreases at first and then takes around 11s to increase to the same value as predicted in the "ZIP

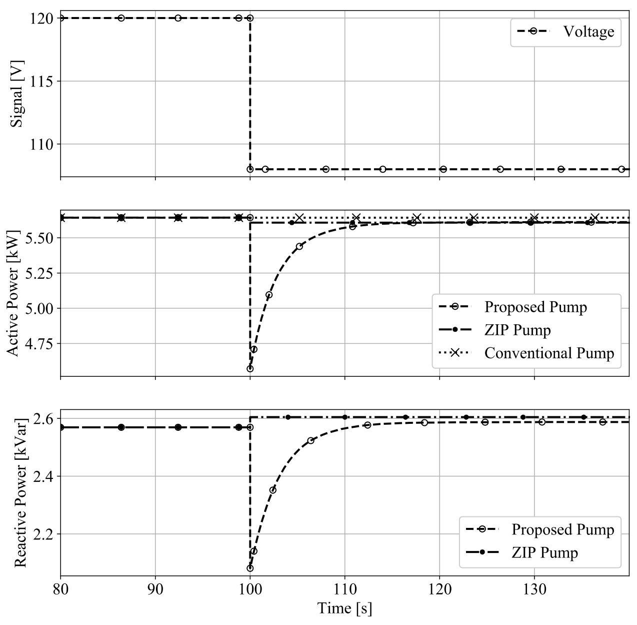


Figure 7. Simulation results for the second scenario

pump". Regarding the reactive power simulation result, we see a pattern similar to that of the active power. Based on the results, it is clear that the "proposed pump" provides a more realistic representation of the response of the building systems to the changing supply voltage.

5 Conclusion

In this paper, we developed Modelica models enable consideration the interactions between buildings and the power system. Based on the result from the proof-of-concept study, we can see that the proposed models can provide much better representation of the response of the building systems to changing operation status and changing supply voltage. Therefore, we believe those models can support research where more complicated interaction between buildings and the power system need to be considered.

In a future study, we will perform the validation of the proposed models with real-world data to quantitatively evaluate their performance. In addition, the motor model will also be connected with power flow analysis tools instead of receiving ideal grid information as in the case study. This will enable a simultaneous simulation of building and grid system.

6 Acknowledgement

This work has been supported by the the Buildings Technologies Office of the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy.

The authors would like to thank Dr. Marina Sofos, Sensors and Controls Technology Manager in the Building Technologies Office at the the U.S. Department of Energy, for her technical guidance and support.

The authors thank Dr. Thomas McDermott for his helpful technical feedback and Matthew Wilburn for editing the final version of the manuscript.

References

Aainaa Mohd Arriffin, Muhammad Murtadha Othman, Amirul Asyraf Mohd Kamaruzaman, Ismail Musirin, Ainor Yahya, and Mohd Fuad Abdul Latip. Stochastic approach of voltage optimization to maximize power saving in a building. *Indonesian Journal of Electrical Engineering and Computer Science*, 8(1):268–272, 2017.

Andrii Bichik et al. Impact of voltage variation on domestic and commercial loads. 2015.

A. Bokhari, A. Alkan, R. Dogan, M. Diaz-AguilÃş, F. de LeÃşn, D. Czarkowski, Z. Zabar, L. Birenbaum, A. Noel, and R. E. Uosef. Experimental determination of the zip coefficients for modern residential, commercial, and industrial loads. *IEEE Transactions on Power Delivery*, 29(3):1372–1381, June 2014.

Dag Brück, Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Dymola for multi-engineering modeling and simulation. In *Proceedings of modelica*, volume 2002. Citeseer, 2002.

David P Chassin, K Schneider, and C Gerkensmeyer. Gridlab-d: An open-source power systems modeling and simulation environment. In *Transmission and distribution conference and exposition, 2008. t&d. IEEE/PES*, pages 1–5. IEEE, 2008.

Drury B Crawley, Linda K Lawrie, Frederick C Winkelmann, Walter F Buhl, Y Joe Huang, Curtis O Pedersen, Richard K Strand, Richard J Liesen, Daniel E Fisher, Michael J Witte, et al. Energyplus: creating a new-generation building energy simulation program. *Energy and buildings*, 33(4):319–331, 2001.

Peter Fritzson and Vadim Engelson. Modelica – a unified object-oriented language for system modeling and simulation. In *European Conference on Object-Oriented Programming*, pages 67–90. Springer, 1998.

Yangyang Fu, Michael Wetter, and Wangda Zuo. Modelica models for data center cooling systems. In *2018 Building Performance Analysis Conference and SimBuild, Chicago, Illinois, United States of America*, 2018.

Yangyang Fu, Wangda Zuo, Michael Wetter, James VanGilder, Xu Han, and David Plamondon. Equation-based object-oriented modeling and simulation for data center cooling: A case study. *accepted by Energy and Buildings*, 2019.

Elliott M Gilbert. Regulated power supply, July 27 1965. US Patent 3,197,691.

K. Hatipoglu, I. Fidan, and G. Radman. Investigating effect of voltage changes on static zip load model in a microgrid environment. In *2012 North American Power Symposium (NAPS)*, pages 1–5, Sept 2012.

X. He, R. Zhao, C. Zhu, and H. Yang. Improving short-term voltage stability problems by variable-speed air-conditioners. In *2012 15th International Conference on Electrical Machines and Systems (ICEMS)*, pages 1–6, Oct 2012.

GK Hood. The effects of voltage variation on the power consumption and running cost of domestic appliances. In *Australasian Universities Power Engineering Conference (AUPEC)*, 2004.

Sen Huang, Wangda Zuo, and Michael D Sohn. Amelioration of the cooling load based chiller sequencing control. *Applied Energy*, 168:204–215, 2016.

Sen Huang, Wangda Zuo, and Michael D Sohn. Improved cooling tower control of legacy chiller plants by optimizing the condenser water set point. *Building and Environment*, 111: 33–46, 2017.

Keun H Lee. *Optimization of a hybrid electric power system design for large commercial buildings: an application design guide*. PhD thesis, Colorado School of Mines. Arthur Lakes Library, 2014.

Dezhi Li, Gaoying Cui, Lingling Sun, Jiru Yang, Ciwei Gao, and Xiao Chen. A control strategy for static voltage stability based on air conditioner load regulation. In *Systems and Informatics (ICSAI), 2017 4th International Conference on*, pages 288–293. IEEE, 2017.

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

Bei Wu, Yan Zhang, and Minjiang Chen. The effects of air conditioner load on voltage stability of urban power system. In *6 th WSEAS International Conference on Power Systems(PE'06)*, volume 6. Citeseer, 2006.

Modelling of the Central Heating Station within a District Heating System with Variable Temperatures

Tobias Ramm[1] Mathias Ehrenwirth[1] Tobias Schrag[1]

[1] Institut für Neue Energie-Systeme, Technische Hochschule Ingolstadt, Germany
Tobias.Ramm@thi.de

Abstract

Within this paper, the concept of developing a detailed model for an existing district heating system (DHS) is described. The research focusses on the central heating station with multiple different supply units. In the present case, the model is implemented with a close-to-reality-control and will be used for testing new control strategies for the DHS. Therefore, a model with both realistic behavior as well as control interfaces similar to the real control is necessary. Within the NATAR research project (Local heating grids with lowered temperature as provider of balancing power), different targets for the improvement of the control will be investigated. One major target is an intelligent linking between the heat and electrical sector to demonstrate the opportunities of heating grids, as the investigated one, to balance the power grid.

Keywords: district heating system, model, Modelica, variable temperature, validation

1 Introduction

The German government agreed on the reduction of human-made greenhouse gas emissions to slow down the climate change (Bundesministerium für Umwelt, Naturschutz, Bau und Reaktorsicherheit, 2018) by focusing on the carbon dioxide emissions, which have by far the highest share based on carbon dioxide equivalents (CO_2 equivalents) (BMWi, 2018). Considering the application sectors power, transportation and heat, the share of the CO_2 emissions by heat production accounts for approximate one third of the total CO_2 emissions. District heating systems (DHSs) are considered cheap and easy way to integrate multiple renewable energy sources as well as to provide the linkage to the electricity grid.

Currently, DHSs are following different approaches to reduce the greenhouse gas emissions and therefore the environmental impact due to heating systems. On the one hand, the improvement of the system's efficiency and the integration of renewables or waste heat are subject to several research activities. To improve the system's efficiency, the temperature should be lowered, since the distribution losses highly depend on this temperature. Lowering the temperature is usually limited, because the majority of houses are already built. These houses usually need higher supply temperatures compared to new ones. On the other hand, current research on DHSs focusses on the integration into the overall energy system in order to reduce the environmental impact. One example of an innovative DHS which addresses both measurers, is described by Ramm et al. (2017). The system is located in Dollnstein (Germany) and operating since 2015. The scheme of this system is shown in Figure 1. Special emphasis is put on the house transfer stations, which include a heat exchanger, a heat pump and a buffer storage. The heating grid may supply the consumers with high temperatures by the heat exchanger or supply low-temperature heat with 30 °C while the decentralized heat pumps are used to provide heat at a desired level up to 60 °C. Within the central heating station, multiple heat sources can be found. The system includes a solar thermal system, a peak boiler, a combined heat and power plant (CHP) as well as a CO_2 heat pump (HP) utilizing close to surface geothermal heat by a well.

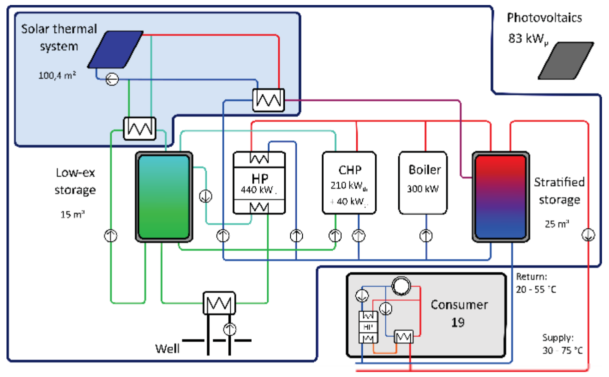


Figure 1. Schematic of DHS with variable temperatures operating in Dollnstein, Germany

The free and open-source programming language *Modelica* and the commercial software tool *Dymola* are widely used for the modelling of complex energy systems. The simulation of DHS in Modelica was done with different focus by numerous researchers.

A conventional district heating system, e.g. a DHS fed by a CHP, was simulated by Sangi et al. (2017). That work focusses on the development and application of exergy sensors for an automated exergy analysis tool based on *Modelica*. Bünning et al. (2018) investigated the novel approach of bidirectional low-temperature district energy systems. They investigated the performance of optimized operation for two test cases with other state of the art DHSs. They proved the concept by reduced energy costs, primary energy consumption and emissions. Del Hoyo Arce et al. (2018) created component models for the fast modelling of district heating and cooling networks to perform real-time simulations for the purpose of model predictive control. Their models were validated against other software tools.

2 Methodology

Within this paper, *Modelica/Dymola* is used to build a detailed simulation model of the heating station of the DHS in Dollnstein. These detailed models may also be referred to as digital twins, like done for the application in the field of building and equipment simulation by (Nytsch-Geusen et al., 2018), but mainly used in the field of production technology.

In the *Modelica* language, many free and open-source libraries are available. This work uses the *Modelica Standard Library* (MSL) and libraries based on the common core library IBPSA *(International Building Performance Simulation Association*). The IBPSA library was created within the project Annex 60 by the IEA EBC (*International Energy Agency Energy in Buildings and Communities Programme*) (Wetter, Fuchs et al., 2015) and is under further development within the IBPSA project. A new, more specialized library is created for the simulation of the DHS in Dollnstein. Main purpose is the use for this application, but the library will be created in an object-orientated way to ensure that the library can be easily used and adjusted for further simulations of low-temperature district heating systems (LTDHS). In this way, the library may constitute the basis for a district heating library at the Institute of New Energy Systems (InES). The scheme for the full model of the DHS is shown in Figure 2. The system is divided into four parts: Central heating station, heating grid, electricity grid and the superior control strategy. An additional block for evaluation purposes completes the model. The arrows indicate the direction of information flow. While the solid arrows represent the flow of physical quantities, which directly influence the receiving subsystem as well as control signals, the dotted lines illustrate the delivering of data like measurement data or set points. The thermal and control models will be developed with high detail and the electricity grid is represented by an ideal source and sink. Additionally, the model of the electricity grid should provide data about the electrical grid, e.g. the residual load. Within this paper, the implementation of the model of the central heating station is described. The model is implemented in an object-orientated manner, using a similar structure as shown in Figure 2. The pumps and valves are integrated within the thermal producer models. In addition, a single sub model represents the full solar thermal system marked with blue border and background (top left). All models contain low-order control algorithms, e.g. for pumps and valves. This kind of implementation together

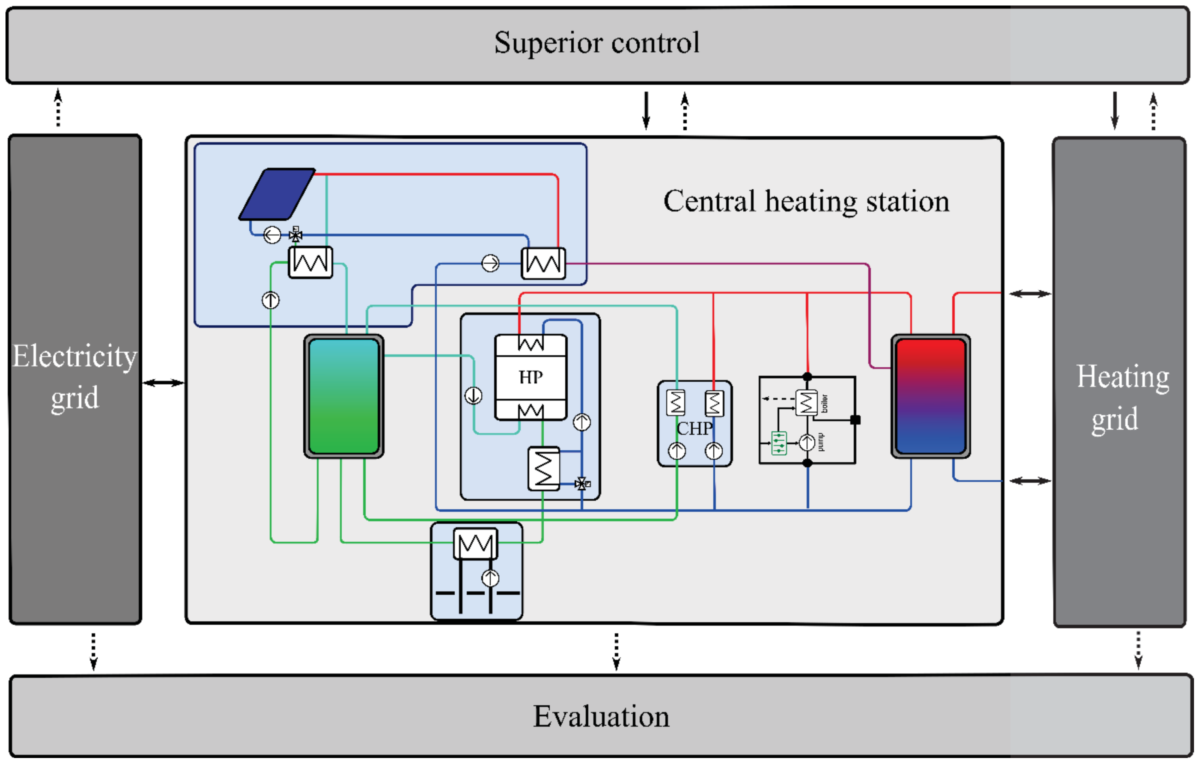


Figure 2. Schematic of system model. Solid arrows represent the flow of physical quantities which directly influence the receiving subsystem as well as control signals, the dotted lines constitute the delivering of data like measurement data or set points.

with the use of base class models as shown e.g. in Figure 3 and Figure 4 allows for a quick exchange of parts of the system.

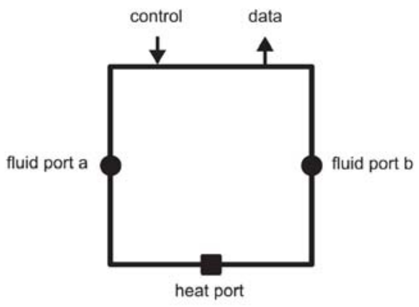


Figure 3. Basis model for a component (e.g. the boiler model)

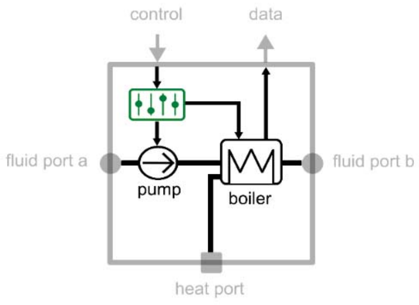


Figure 4. Full model for a component including secondary components and low-level control (e.g. boiler model)

The base class models include all mandatory interfaces to integrate the models to the system model. The boiler model shown in Figure 3 is an example and includes three ports with physical properties and several ports with further data. Due to the non-causal modelling approach of *Modelica*, the physical ports do not need an explicit direction of transferring information. The model has two fluid ports for entering and leaving heat transfer medium as well as one heat port to account for the heat transfer to the ambient. More information about the concept of physical connectors within *Modelica* can be found in Modelica Association (2017). The non-physical connectors have an explicit direction of flow of information. A control signal is entering the boiler component and information for statistical and control purposes are leaving the component. The base class model is extended and additional information and sub models were added as shown in Figure 4. Components within the model can be exchanged for example by more or less detailed ones.

The component models of the heating station will be validated with measurement data from the actual operation mode. Temperatures, heat flow rates as well as electrical power and energy sums are used for the validation.

3 Modelling

The model is implemented in the programming language *Modelica* with the software *Dymola* (version 2019). The implementation is realized as shown at the scheme within Figure 2. On the one hand the model should be capable of simulating periods up to one year, within a reasonable time, to evaluate the systems performance for new operation approaches. One the other hand temperature changes as well as shut-on and shut-off behavior in minute resolution needs to be provided to balance the electrical grid on this scale. In this paper, the distinction between superior and low-level control is made. Low-level control means the control of e.g. pumps and valves by e.g. PID controllers. The superior control on the other hand decides based on the status of the system, e.g. the thermal storage tanks, whether heat production units are shut-on or shut-off.

The entire central heating station and the superior control in the simulation similar to the superior control of the real system are implemented. Both the district heating network as well as the electricity grid are represented as boundary conditions. The heat generators are modelled including the low-level control and additional aggregates like pumps and valves. Subsequently, the implementation of the main components is described briefly, including the boiler, the CHP, the heat pump, the solar thermal system and the thermal storage tanks. The efficiency depending on the operation temperatures is of major importance for the investigation of the system. This is especially true for the solar thermal system and the heat pump, since these components are highly dependent on the operation temperatures. Furthermore, the operation temperatures within the overall system are import to investigate and may change heavily during the year. Input data for the model are the out- and indoor ambient temperature as well as the solar radiation. Hitherto, the electricity grid as well as the heating grid are also input data.

3.1 Combined heat and power plant

The CHP from the manufacturer *Riemag* is powered by liquid gas and extended with an additional hydraulic cycle, which uses the condensing energy as well as the heat from the exhaust air coming from the case of the CHP. The first cycle feeds the stratified storage while the second one provides the heat to the low-temperature storage. The model also has one input to control the start-up/shut-down of the model as well as its behavior under partial load. The CHP unit is a model from the *BuildingSystems* library (Nytsch-Geusen, et al., 2016). The model is based on two characteristic curves, one for the electrical power and one for the heating. These curves are used for the calculation of the fuel demand. While the internal pump of the CHP is controlled to maintain a certain supply temperature, the pump for the condensing cycle just switches on and off with respect to the operation of the CHP and the temperature inside the low-temperature storage. The heat transfer to the additional hydraulic circuit is considered by a constant heat flow rate, as long as the CHP is in operation. The heat flow rate was determined by measurement data. The heat transfer is only possible until a certain temperature of the fluid. The pump of the second hydraulic cycle is only operational when the low-temperature tank has a maximum temperature below 21 °C.

3.2 Heat pump

At the Dollnstein system, a heat pump of the type *thermeco2 HHR 520-3(345)* with additional internal heat exchanger and 135 kW_{el} nominal input, distributed equally on three compressors, is installed. The set supply temperature varies over time.

The HP model is based on a HP model out of the IBPSA library. The HP in the real system has three compressors. It is assumed that the efficiency does not depend on the number of compressors that are in operation as long as they work at full load. The efficiency at partial load can be described by an arbitrary polynomial function. Therefore, a new model including three of the aforementioned HP models in parallel was set up. The main ports of the model are four fluid ports, two at the source and two at the sink as well as a control signal, which can vary between 0 and 1. In this context one third means one compressor at nominal load. The effectiveness at nominal condition can be defined by the Carnot efficiency and a reducing factor or by giving a coefficient of performance (COP) at nominal conditions.

Moreover, the model is used with a given COP at nominal conditions, which was taken from the manufacturer data. For these nominal conditions, the manufacturer claims an electrical demand of 124 kW. The electrical demand of real HP is changing by the variation of the operation point. The model instead has the same electrical demand in every operation point. Therefore, the electricity demand of the model was adjusted to the measurement data from 124 kW to 132 kW.

The main inputs for the energetic calculations are the temperatures and temperature differences at the heat exchangers as well as the nominal COP and the already mentioned electrical power consumption. The dynamic behavior is described by the capacity of fluid within the heat exchangers, which are bounded to the nominal mass flow rates.

3.3 Boiler

The boiler model has four important interfaces: two fluid ports, one heat port and one control input. The model consist of the control, a pump and the actual boiler model, which is part of the *Buildings* library (Wetter, Bonvini et al., 2015). The actual control of the boiler is a two-point control depending on the storage temperature. During run-time, the boiler runs with fixed power. The pump is also operated on a fixed power and switches on and off together with the boiler, but delayed. A short description of the actual boiler model can be found within the model description of the *Buildings* library. Hitherto, a constant efficiency of the boiler is chosen, because the gas consumption from the boiler on its own cannot be determined yet.

The losses to the surrounding can be included by the efficiency curve of the boiler model or additionally added by connecting the heat port to the ambient. The dynamic behavior is determined by the heat capacity of the boiler itself as well as the fluid volume inside the boiler and the heat losses.

In this modelling step, the parameters were taken from the manufacturer data for the boiler type *Buderus SB 625-310 (289.9 kW)*.

3.4 Solar thermal system

The solar thermal system consists of the hydraulic collector cycle including the collectors, a pump, a three-way-valve and two heat exchangers. The solar thermal collectors are flat plate collectors from the manufacturer *ratiotherm* of type *RA 251-4*. The two heat exchangers supply the two different thermal storages on different temperature levels with heat.

The collector model was chosen from the *BuildingSystems* library. This model is a general multi-node model. Due to lower computational effort, the solar collectors are aggregated in one model. The main parameters are the coefficients to calibrate the efficiency curve. Additional parameters are the absorber volume and heat capacity. All parameters are available from the manufacturer data. During summer operation, the solar collectors only feed the stratified storage. The feeding during winter operation depends on the temperature within the low-exergy storage. Until a certain threshold temperature inside the low-temperature storage, the solar heat is fed to that storage. The set supply temperature of the solar thermal system is also dependent on the temperature of the storage to be fed.

3.5 Thermal storage tanks

The central heating station includes two thermal storage tanks with volumes of 15 m^3 and 25 m^3. The larger one is the stratified storage that feeds the grid while the buffer storage at lower temperature feeds the source of the heat pump.

For both storages, the storage model from the *BuildingSystems* library is used. The model is a one-dimensional model that takes the buoyancy by a simple thermal model into account. Mixing due to in- and outflowing streams is neglected. The necessary data to parametrize both heat storages is taken from manufacturer information. The stratified storage includes temperature measurements at five different levels of height. The discretization within the model is done in a way that the temperature sensors are located in different segments. The buffer storage is represented only by five elements and includes three temperature sensors.

3.6 Superior control

The actual superior control is primarily based on the temperatures within the storages. The aggregates switch on and off due to the temperatures within the stratified storage. Minimum operation times and down times are not taken into account yet. The set temperature for the grid is given as an input table and relates both to the outdoor ambient temperature and the heat demand. The superior control is also able to control the decentralized heat pumps and may take different temperatures at the consumers into consideration. However, this is not used yet, as the heating grid only serves as boundary condition.

4 Model validation

The validation of the model is described subsequently. The model was validated quantitatively in terms of energy in- and output, e.g. thermal, electrical and fuel consumption on a monthly scale. Additionally, the dynamic behavior of the component models was evaluated in a qualitative way. Mass flow rate, the supply temperature, the provided heat and consumed electrical power of the HP were studied for one cycle of operation (shut-on and shut-off). In addition, the full period of time is taken into account by analyzing the root mean square error (RMSE) between the simulation and measurement. The RMSE is discussed and described by Chai and Draxler (2014). The discrete form is shown in equation (1). The formula describes the square root of the average of the squared differences at certain evaluation points. The quantity is the same as the quantity of the data.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{\text{sim}} - x_{\text{meas}})^2} \tag{1}$$

For evaluation purposes, the non-discrete notation was implemented as displayed in equation (2).

$$RMSE = \sqrt{\frac{1}{T}\int_{t=1}^{T}(x_{\text{sim}} - x_{\text{meas}})^2} \tag{2}$$

An evaluation for the different components can be found in the following sub-sections.

If available, recorded control data from superior control were used as inputs for testing the components. Otherwise, the control signal was derived from the measurement data. The evaluation is summarized in the end of the section and energy wise displayed in Table 1.

4.1 Combined heat and power plant

The operation of the CHP is investigated for the period from 1st March to 31st May 2018. For the observed months, the CHP was about 1000 hours in operation and fed 164.5 MWh heat to the stratified storage as well as 12.6 MWh to the low-temperature storage. The simulation model fed 163.0 MWh to the stratified storage, which means an error of only 0.9 % (see Table 1), but 32 MWh to the low-temperature storage. The resulting difference occurs because the model also supplies the low-temperature storage within summer-mode, while the real CHP does not. This functionality needs to be added to the model.

Figure 5 shows different values for one operation cycle in March 2018. The control signal is always plotted in the sub-plots below.

The left sub-figure shows the electrical power of the system. Negative values indicate that the CHP is running and electrical power is fed to the public grid. Positive values indicate a supply of the system from the public electricity grid.

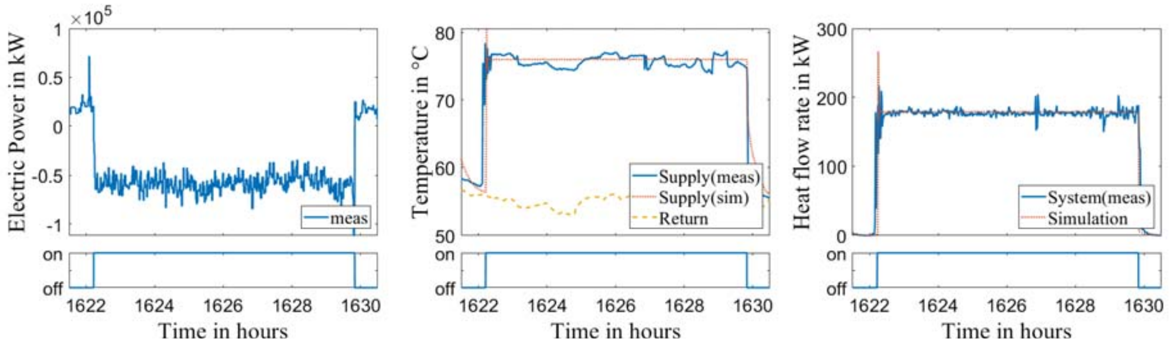


Figure 3. Validation for the CHP operation - one operation cycle. Left sub-figure shows the electrical power of the system. Sub-figure in the middle displays supply and return temperatures for the conventional and the condensing cycle. The right sub-figure shows the heat flow rates for the conventional as well as the condensing cycle. Below the sub-figures, the control signal is plotted.

The real CHP does not have a very stable control. Due to that reason, both the mass flow rate and the supply temperature (middle sub-plot) are strongly varying during operation. As a result, the heat flow rates (right sub-figure) strongly vary during operation too. This is especially true during times without the utilization the condensing cycle. The control of the model is much more stable compared the real control. The supply temperature (middle sub-figure) and provided heat flow rate (right sub-figure) are almost constant during operation. Only after switching on the unit, the supply temperature shows a peak (Figure 5- middle sub- plot) that also leads to a peak in the supplied heat flow rate (Figure 5 - right sub-plot). The RMSE for the heat flow rate is 20.4 kW and 4.2 °C for the temperature. This means noticeable differences over the course of time, but a big share of these differences is due to the unstable control of the real CHP. Since the aim is not to reproduce this behavior, this value itself has only little significance. Therefore, the allover heat energy (Table 1) and the graphs shown in Figure 5 are more meaningful in this case. Although there are some differences between model and reality, the allover validation results show that the CHP model meets the requirements of the simulation analysis formulated in section 3.

4.2 Heat pump

The observed operation period for the heat pump lasts from 1st January to 30th April 2018. During the summer mode, the heat pump is not operating. Within the four observed months, the heat pump is approximately 2000 hours in operation, mostly with one compressor, sometimes utilizing a second compressor, but never using all three compressors. The inlet temperature at the sink needs to be maintained below 50 °C, because higher temperatures lead to problems with the heat pump operation. An additional heat exchanger provides reliability of operation also during the winter period, when higher return temperatures in the grid occur. During summer, this is not an issue and the return temperature may drop down to 20 °C. The supply temperature of the grid during winter usually varies between 60 °C and 80 °C. The source temperature ranges between 10 °C and 23 °C while the temperatures usually range around 11 °C, which is in the range of the supply temperature of the well. The source temperature is in the higher range when the low-temperature storage was charged by the solar thermal system or the condensing cycle of the CHP. The temperatures of the source are not displayed in Figure 6.

Figure 6 shows different values for one operation cycle in January. The control signal is always plotted below the other graphs. While the real HP consumes an electricity amount of 128.7 MWh, a minor deviation of less than 1% between the real HP and the model can be observed as shown in Table 1. The left sub-plot displays the measured and simulated electricity consumption, which match each other very well. In the middle figure, the supply and return temperatures are shown. After switching on the HP, the model needs a certain time to reach stable control as the control of the pump is not optimal parametrized. Adjustments within the low-order control need to be done to achieve a more stable start-up behaviour. While the supply temperature fits during operation, there is a significant deviation during the non-operation time. This is acceptable, since the thermal behaviour during non-operation time is of minor importance as long as the influence on the operation time is limited.

The graphs in the right figure show the effective heat flow rate for the heat, which is transferred to the fluid, i.e. the heat flow rate provided by the heat pump system. The heat pump system includes the additional heat exchanger, shown in Figure 1. Therefore, the heat flow rate provided by the heat pump is reduced by the heat flow transferred between the heat exchanger and the source of the heat pump. The systems efficiency is highly dependent on the set temperature of the valve, which ensures that the return temperature directly at the heat pump stays below the threshold. The simulated heat for the displayed flow rate is slightly lower than the measurement data. This holds true for the most of the

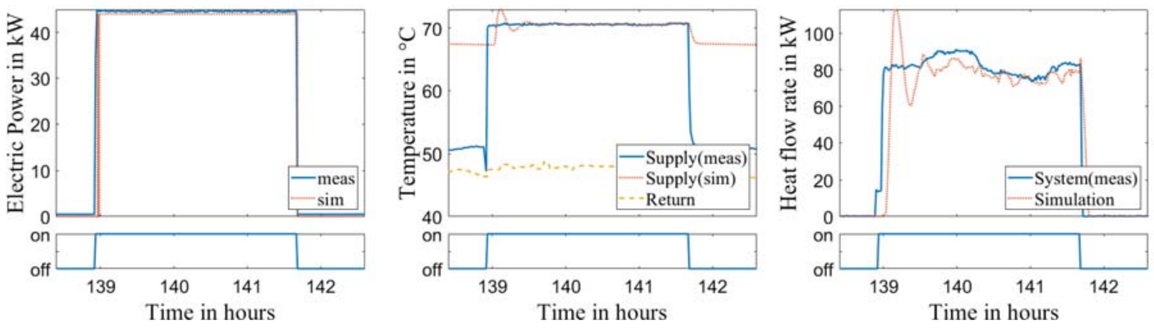


Figure 4. Validation for the HP operation - one operation cycle. Left sub-figure displays the electrical power input. Middle figure displays the supply and return temperature for the sink of the HP. Right sub-figure shows the heat production of the heat pump. Below the sub-figures the control signal is plotted.

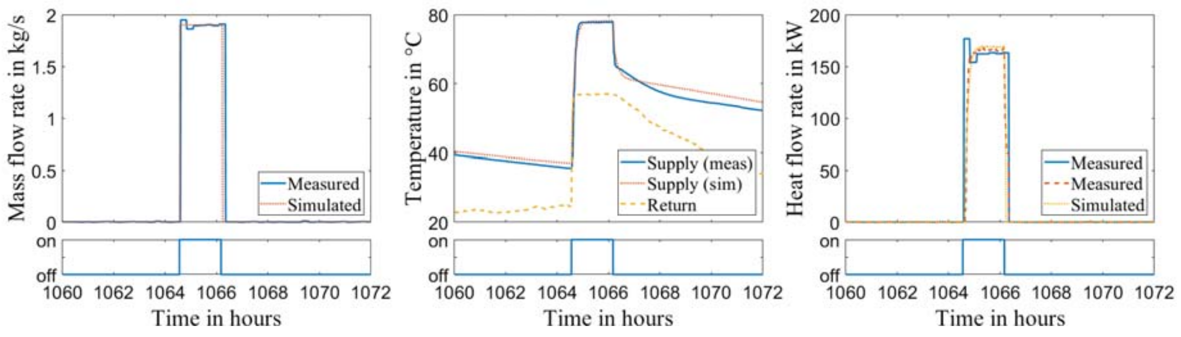


Figure 5. Validation for the boiler operation - one operation cycle. Left sub-figure shows the mass flow rate. The middle sub-figure displays supply and return temperatures. The right sub-figure show the provided heat flow rate. The control signal is plotted below the sub-figures

operation cycles, but the deviation changes, resulting in a deviation of produced heat of 10.7 %. In the future work, the calibration of the HP model will be further pursued.

4.3 Boiler

For the testing of the boiler model, the recorded operation signal from the superior control was used as an input for the simulation model. The actual measured temperature constitutes the boiler's fluid inlet temperature. The validation period is the time from 1st February to 30th of March 2018. The boiler's operating periods lasted between 0.5 and more than 4 hours. Within 34 hours of operation, the boiler generated a heat amount of 5.43 MWh that means an average heat output of 160 kW. During summer, the boiler is not in operation. The heat output in the model is 5.5 MWh, which amounts to an error of 0.9 % (Table 1).

The dynamics of the model compared with the measured data are shown in Figure 7. In the model as well as in reality, sensors measure the temperature directly after the boiler. Below the main figures, the recorded operation time of the boiler is shown. The left figure shows the measured and simulated mass flow rate. The assumption of 1.9 kg/s within stable operation meets the measurement data (1.85 kg/s to 1.95 kg/s) quite well. A comparison of the supply temperature from simulation with the measurement data shows, that these match very good during operation, but shows different results when the boiler is turned off. After turning off, the pump has a shutdown delay of three minutes. During this time, there is a strong drop in the supply temperature in the measurement data and the simulation. In the following time, the temperature drop in the simulation is slower compared to the measurement. Since both the mass flow rate as well as the temperatures fit during simulation, a high degree of correlation can be observed for the heat flow rate, too (Figure 7- sub-plot right). The RMSE of the heat flow rate for the full operation time is 11.6 kW, 2.2 °C for the supply temperature and 0.15 kg/s for the mass flow rate. Taking all these values in consideration, the boiler model meets the measurements and requirements very well.

The efficiency e.g. the gas consumption could not be validated yet, because there is no data from that sensor yet.

4.4 Solar Thermal System

The collector model was parametrized by the data sheet of the real collector. The simulation results meet the steady-state performance according to the data sheet. The full solar thermal system was not validated yet, but will be processed in following studies.

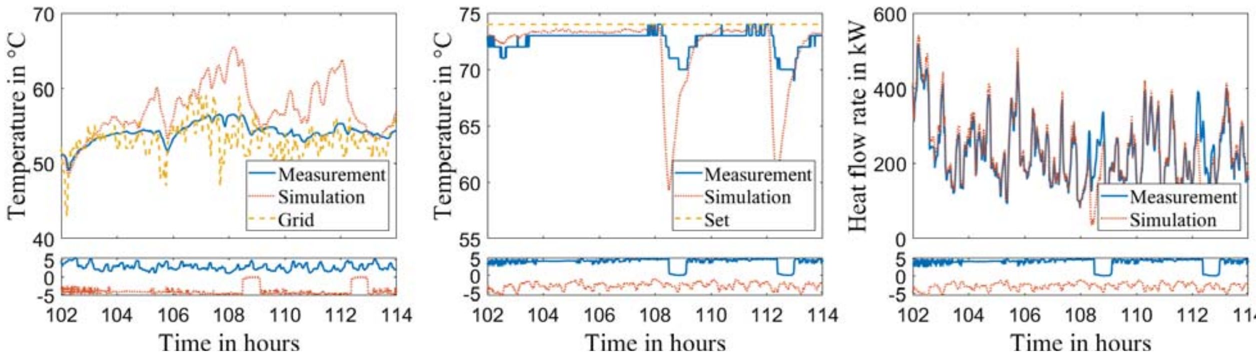


Figure 6. Validation for the utilization of the stratified storage – Period of 12 hours in January. Left sub-figure shows the temperatures of the fluid leaving the bottom of the storage to enter the return of the producers and from the return inflowing from grid. The middle sub-figure displays the supply temperature for the heating grid. The right sub-figure shows the heat flow rate leaving the storage to the district heating grid. The entering (positive) and leaving (negative) mass flow rates are plotted below the sub-figures.

4.5 Thermal storage tanks

The validation period for the storage tank is from 1st January to 1st February 2018. Figure 8 shows 12 hours of the validation period. To perform the validation the inflowing mass flow rates with temperatures as well as the indoor ambient temperature are described. For the outflowing fluid, only the mass flow rate is given and the temperature is part of the evaluation. The temperatures of the outgoing mass flows for the supply of the heating grid as well as the return to the heat generators are import for the system's behavior. In addition, the temperatures at the temperature sensors inside the storage are important for the superior control, because the heat producers are shut on and off due to these temperatures.

Over the course of the month, the storage supplied to the grid amounts to 176.8 MWh of heat. The model supplied 175.3 MWh and meets the measurement data with an error of 0.8 % (see Table 1).

In the left sub-figure the measured and simulated return temperatures to the heat producers, outflowing from the stratified storage at the bottom, are plotted together with the temperature of flow entering from the grid. In the sub-figure below, the mass flow rates, leaving and entering at the bottom of the heat storage are shown. The outflowing mass flow rate is negative (flow to the producers) and the entering one (flow from the grid) is positive. For most of the time, the simulated temperature is higher than the measured one. During periods without operation of heat producers, the temperatures approach each other. The temperature deviation is significant, thus the behavior needs to be investigated more in detail. The temperature difference at the bottom of the storage needs to be reduced, since this temperature is relevant for the entire system. The sub-figure in the middle displays the measured and simulated temperature at the top of the storage, supplying the grid as well as the set-temperature of the supply. To maintain the supply temperature the colder return from the grid can be mixed with the supply from the storage. This ensures that the supply has no higher temperatures than necessary.

The measured and simulated grid supply temperatures match well. This holds true for the most of the observed period. Nevertheless, it can also be realized that the temperature in the model changes faster and that there are periods with big differences. These periods typically occur when the temperature at the top of the heat storage is dropping. This happens in case that the producers do not supply the heat storage anymore while the storage still feeds the grid. The drop in the model is more drastic. In the right sub-plot, the influence on the supplied heat flow rate is displayed. It can be seen that measurement and simulation match well for most of the time, but the differences occur at similar times as for the supply temperature. The simulated heat flow rate is lower than the measured one during phases of no heat production. As already mentioned in the beginning of this section, the overall heat supply of the model from the storage to the grid meets the measurement data very well, so that the influence of the differences in the heat flow rates at certain times is not significant for the evaluation of the energy balance.

4.6 Superior control

The superior control for the operation of the central heat production units was implemented. For the most cases, the real controller and the model controller show the same behaviour, but slight time shifts occur.

4.7 Heating Central

The model of the entire heating central contains the models validated in the sub-sections before. For the central heating station, the efficiency of the whole system is most important and will be evaluated. Therefore the incoming and outgoing energy flows will be used.

Table 1. Evaluation of produced heat and energy demand for observation periods (CHP: 1st January to 30th April 2018, Heat 1 feeding to stratified storage, Heat 2 feeding to low-temperature storage; HP: 1st January to 30th April 2018; Boiler: 1st February to 30th March 2018; Storage: 1st January to 1st February 2018)

Component	Heat/ Fuel/Electricity	Energy (Measurement)	Energy (Simulation)	Rel. error [%]
CHP	Heat 1	164.5 MWh	163.0 MWh	- 0.9
	Heat 2	12.6 MWh	32.0 MWh	
HP	Heat (effective)	236.8 MWh	211.4 MWh	- 10.7
	Electricity	128.7 MWh	127.6 MWh	- 0.9
Boiler	Heat	5.4 MWh	5.5 MWh	+ 0.9
Storage	Heat (grid supply)	176.8 MWh	175. 3 MWh	- 0.8

4.8 Discussion

The component models show a good correspondence between simulation and reality. The superior control sends operation signals to the production units. These are mainly dependent on the storage temperatures of the stratified storage. The part of the superior control that is related to the grid operation will be implement in the next step.

The heat producers and the superior control are implemented, but for some operation conditions, significant differences occurred. One of these is the CHP within during the summer period. At this time, the real CHP does not provide any heat to the low-temperature storage. This needs to be taken into account and is the reason, why there is a significant difference in the heat amount supplied to the low-temperature storage between measurement and simulation. The delivery of heat by the HP is very sensitive with respect to the mixing temperature at the sink inlet. The mixing of the inlet temperature as well as the implementation of the of the heat pump characteristics need adjustment to reduce difference between the simulation and measurement results for the heat pump.

To observe the system's efficiency with respect to fuel some, adjustments for the consumption of fuel have to be made. Furthermore, the power demand of the pumps within the system must be taken into account. Additionally, minimum operation and shut-down times should be considered, since these are limiting future control strategies.

5 Conclusion

The central heating station of the DHS was implemented within *Modelica* with all major units and additional components like pumps and valves. The model is appropriate for the detailed simulation of the central heating station and investigation of new superior control strategies.

The models of heat generators were implemented and validated by the measurement data from the real system. The heat supply shows a high degree of correlation. Nevertheless, there is potential for improvement. The efficiencies with respect to the fuel needs to be evaluated. In addition, improvements for the low-level control could be done to improve the stability and computational performance. Additionally the model of the heat pump should be further improved. On the one hand, the model itself could be improved, one the other hand the parametrisation of model could be improved. Since the temperatures within the heat storages are of major importance for the control of the system, occurring temperature differences should also be investigated more in detail.

Completing the validation of the system, will be the next step. Especially the solar thermal system and the gas consumption of the CHP and boiler need to be adjusted to measurement data. Additionally, the power consumption for the pumps needs to be added to evaluate the overall performance of the heating station and the subsystems.

The energy amounts for the whole period are used for the evaluation, since these are of major importance for evaluating the efficiency for the whole year. However, it needs to be secured that the operation for different conditions meet the real systems behaviour.

To model the entire system, a model of the heating grid including the consumers' needs to be added. This model extension is necessary to improve the entire control system including the control of the circulator pumps and the use of the distributed HPs.

6 Acknowledgements

The authors are grateful to *Bundesministerium für Wirtschaft und Energie* (BMWi), which finances the project NATAR (Local heating grids with lowered temperature as provider of balancing power; grant number: 03ET1425A), the founding agency *Project Management Jülich (*PtJ*)* as well as all project partners.

7 References

BMWi (2018). Höhe der Treibhausgasemissionen in Deutschland nach Gas im Jahresvergleich 2000 und 2016 (in Millionen Tonnen CO2-Äquivalent). [Online]. Available at https://de.statista.com/statistik/daten/studie/311861/umfrage/menge-der-treibhausgasemissionen-nach-gasen-in-deutschland (Accessed 2 October 2018).

Bundesministerium für Umwelt, Naturschutz, Bau und Reaktorsicherheit (2018). Klimaschutz in Zahlen: Sektorenziele 2030. [Online]. Available at), www.bmub.bund.de (Accessed 18 October 2018).

Bünning, F., Wetter, M., Fuchs, M. and Müller, D. (2018). Bidirectional low temperature district energy systems with agent-based control: Performance comparison and operation optimization. *Applied Energy*, vol. 209, pp. 502–515. doi:10.1016/j.apenergy.2017.10.072

Chai, T. and Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. *Geoscientific Model Development*, vol. 2, 1247–1250. doi: 10.5194/gmd-7-1247-2014

del Hoyo Arce, I., Herrero López, S., López Perez, S., Rämä, M., Klobut, K. and Febres, J. A. (2018). Models for fast modelling of district heating and cooling networks. *Renewable and Sustainable Energy Reviews*, vol. 82, pp. 1863–1873. doi: 10.1016/j.rser.2017.06.109

Modelica Association (2017) Modelica - A Unified Object-Oriented Language for Systems Modeling: Language Specification.

Nytsch-Geusen, C., Kaul, W. and Kharraz, S. (2018). Der digitale Zwilling in der energetischen Gebäude- und Anlagensiulation. *BauSIM2018 - 7. Deutsch-österreichische IBPSA-Konferenz*. Karlsruhe, pp. 311–318.

Nytsch-Geusen., C., Banhardt, C., Inderfurth, A. and Mucha, K. (2016). Buildingsystems - Eine modular hierarchische

Modell-Bibliothek zur energetischen Gebäude und Anlagensimulation. in Grunewald, J. (ed) *CESBP Central European Symposium on Building Physics/BauSIM 2016: Dresden, Germany, September 14 - 16, 2016. E-Book of Proceedings,* Stuttgart, Fraunhofer IRB Verlag.

Ramm, T., Hammel, C., Klärner, M., Kruck, A. and Schrag, T. (2017). Energy storage and integrated energy approach for district heating systems. *Energy Procedia*, vol. 135, pp. 391–397. doi: 10.1016/j.egypro.2017.09.515

Sangi, R., Jahangiri, P., Thamm, A. and Müller, D. (2017). Dynamic exergy analysis – Modelica®-based tool development: A case study of CHP district heating in Bottrop, Germany. *Thermal Science and Engineering Progress*, vol. 4, pp. 231–240. doi: 10.1016/j.tsep.2017.10.008

Wetter, M., Bonvini, M., Nouidui, T. S., Tian, W. and Zuo, W. (2015). Modelica Buildings Library 2.0. *Proceedings of BS*. Hyderabad, India, Dec. 7-9, 2015.

Wetter, M., Fuchs, M., Grozman, P., Helsen, L., Jorissen, F., Lauster, M., Müller, D., Nytsch-Geusen, C., Picard, D., Sahlin, P. and Thorade, M. (2015). IEA EBC Annex 60 Modelica Library – An International Collaboration to Develop a Free Open-Source Model Library for Buildings and Community Energy Systems. *Proceedings of BS.* Hyderabad, India, Dec. 7-9, 2015, pp. 395–402.

SESSION 5C: THERMODYNAMIC 1

Towards Hard Real-Time Simulation of Complex Fluid Networks

Dirk Zimmer[1]

[1] DLR German Aerospace Center, Insitute of System Dynamics and Control,
Münchener Strasse 20, 82234 Oberpfaffenhofen, Germany,
dirk.zimmer@dlr.de

Abstract

Complex fluid systems created with Modelica have often been difficult to simulate under hard real-time constraints since they typically involve non-linear equation systems that are difficult to solve especially within a predictable finite time frame. This paper explores the usage of a new approach that avoids non-linear equation systems and its suitability for real-time simulation. A model of an aircraft environmental control system is taken as a use case for this study.

Keywords:Thermal fluids, Thermal process systems, real-time simulation, environmental control systems

1 Introduction

1.1 Real-time simulation

Real-time simulation is a very useful tool, especially since it is a key enabler for virtual testing: control units can be tested against a virtual plant model or single components can be tested with software-in-the-loop in order to emulate a yet unavailable environment.

Although aviation, as our main application domain invests heavily in virtual testing efforts, real-time simulation has been difficult to achieve for the complex fluid systems that are contained within an aircraft, especially for the environmental control system (ECS). Figure 1 shows a Modelica example system that (simplistically) describes the flow of fresh air and recirculation air on board of a conventional aircraft.

Although this is not a realistic model, it is a suitable example for our purposes since it contains a variety of components and describes an active thermodynamic process as well as natural air distribution.

For public demonstration purposes, this model already runs in real-time (once the initialization phase has been passed) using DASSL as ODE solver. This is a soft real-time simulation since we cannot give any definitive statement on its computing speed or responsiveness other than a purely empirical: “it seems to be fast enough.”

Hard real-time requires definitive statements. The system needs to respond with a given frequency.

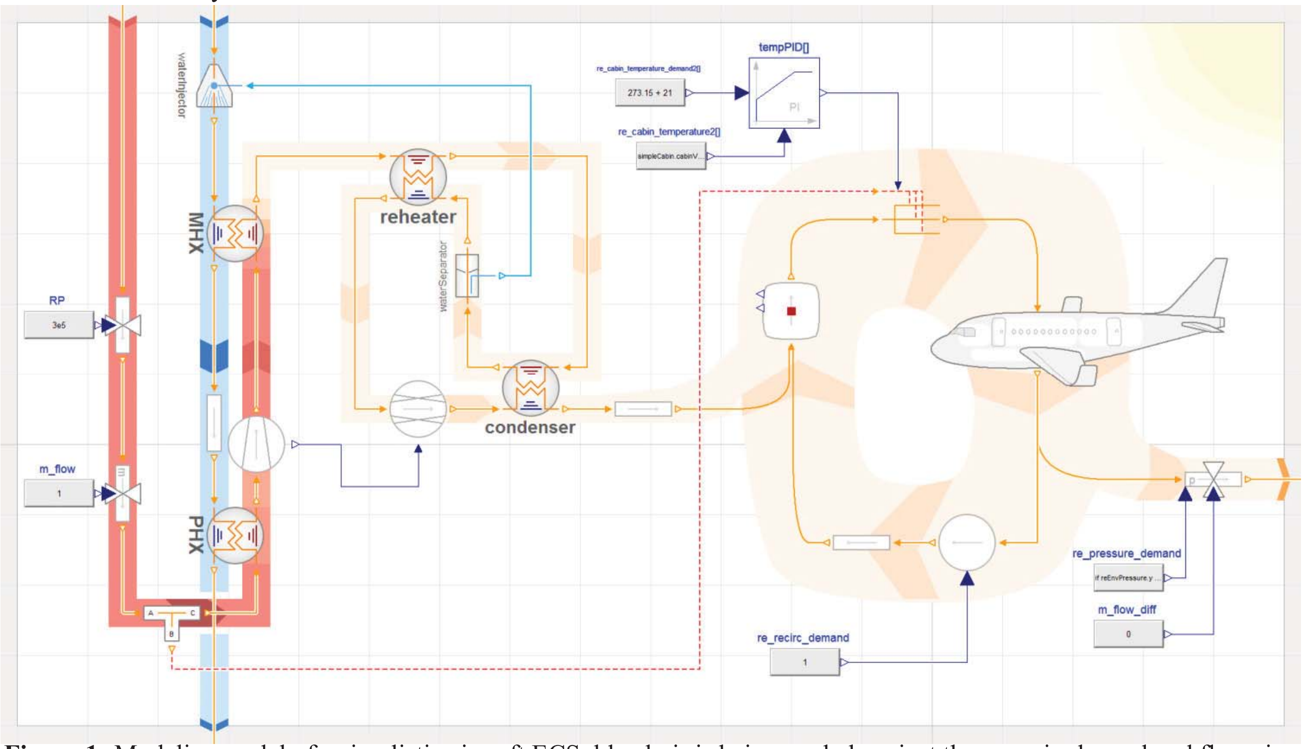


Figure 1: Modelica model of a simplistic aircraft ECS: bleed-air is being cooled against the ram-air channel and flows in a unit where it mixes with recirculated air. This mixture is then being warmed for 3 zones by further bleed-air and let into the cabin. From the 3 upper floor zones, the air flows to the underflow zone where a valve controls the cabin pressure. This model is derived from the work of Alexander Pollok (Pollok, 2017)

Within a period corresponding to this frequency, the system has to be completely updated and the error must be within certain bounds. These requirements heavily affect the suitability of the simulation code and its IT-infrastructure. Figure 2 shows a typical cascade of a real-time simulation.

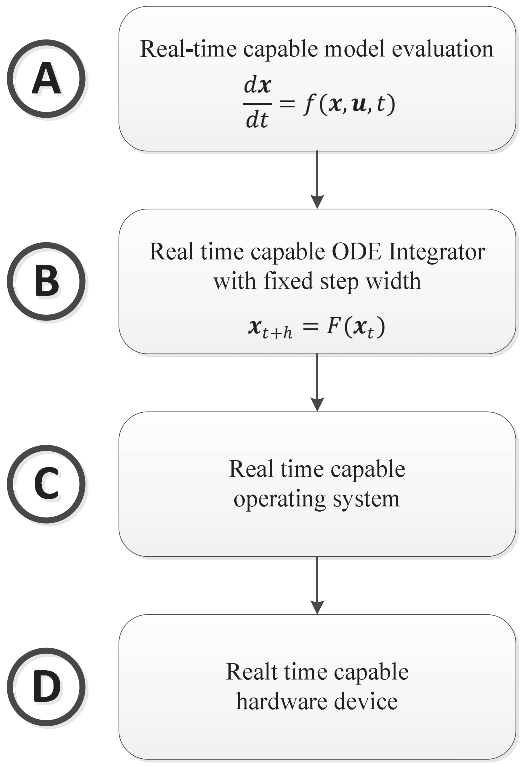

A
Real-time capable model evaluation
$\frac{d\boldsymbol{x}}{dt} = f(\boldsymbol{x}, \boldsymbol{u}, t)$
B
Real time capable ODE Integrator with fixed step width
$\boldsymbol{x}_{t+h} = F(\boldsymbol{x}_t)$
C
Real time capable operating system
D
Realt time capable hardware device

Figure 2: real-time cascade

In this paper, we focus on steps (A) and (B) since they are the most challenging for fluid systems. Once we have solved these issues, it is reasonable to assume that standard solutions for (C) and (D) will help us to go all the way.

1.2 Challenges specific to fluid systems

Why are (A) and (B) challenging for fluid systems? Of course, fluid systems have been simulated in real-time before (Schulze, 2011, Wetter, 2014) but rather in soft real-time. The systems either had to be simple or the simulation speed could only be determined by extensive testing.

Hard real-time is difficult since the model evaluation (A) often contain iterative numerical algorithms. This is because many systems are typically formulated using fluid streams (Casella 2006) and often form large systems involving non-linear algebraic equations (Zimmer 2013) that are then solved using iterative non-linear solvers. Once these solvers are required to evaluate the model function $dx/dt = f(x, u, t)$, hard real-time capability is lost. It is in general practically impossible to make definitive statements on how many iterations are needed for convergence and whether convergence can be guaranteed in the first place. To reach the area of convergence, homothopy methods are used (Casella 2011) primarily at initialization. During simulation the step-size control of the ODE-solver shall guarantee convergence. Again for hard real-time, variable step-size control is not feasible.

Since for (B) explicit ODE solvers (such as explicit Runge-Kutta methods) with fixed step size are preferred, the stiffness of the system becomes an issue. Flows between small volumes may cause such stiffness and small volumes are often introduced by modelers in order to fight the problem of (A): truly a viscous circle.

1.3 A new approach

Fortunately, we made recent progress regarding the DAE-based modeling of fluid streams (Zimmer, 2018). This progress was triggered by the needs to achieve a high level of robustness. Robustness in this context means that the simulation shall not return any errors due to solvability of non-linear equation systems and that the modeler shall not have to care overly about initialization of the system.

To this end, a computational scheme was developed that ensures that once the components robustly compute, also the complete system will robustly compute, meaning no error due to unsuccessful application of iterative numerical solvers.

We can use the same scheme to avoid non-linear equation systems altogether. If every component has a form that enables an explicit computation of the thermodynamic state in the downstream direction, then the thermodynamic states of the whole system can be computed explicitly in the downstream direction. Evidently this will address task (A) in our real-time cascade and Section 2 will briefly discuss its concept. Yet, this is only a very short repetition and the reader is invited to study (Zimmer, 2018).[1]

As part of this concept, the mass-flows $\dot{\boldsymbol{m}}$ become a state vector of the system that is (for natural parameters) typically associated with eigenvalues that are strongly negative or strongly imaginary. This prevents the application of explicit methods with sufficiently large step-width.

However, there are methods how these eigenvalues can be manipulated in such a way that they can be placed within the stability region of a stable method without interfering with the dynamics of the relevant thermodynamic processes (e.g. the heat capacity of a volume, etc.). This will address task (B) and is discussed in Section 3

Finally, we apply the lessons of Section 2 and Section 3 to our demonstration example. This shows the principal feasibility of the proposed methodology and lets us speculate about its limitation and potentials.

[1] Unfortunately the notation w.r.t p and $\hat{p}$ differs in the former paper. A proper journal paper is being written.

2 Avoiding non-linear equation systems

Non-linear equation systems are not only a problem for real-time systems, they are a problem for robustness in general. However, the pressure gradient of a flow along a straight pipe (see Figure 4) contains a component that is independent of the thermodynamic state and only linearly dependent on the time derivative of the mass flow rate. This gradient Δr corresponds to the inertial force f for accelerating an arbitrary line flow of length Δs:

$$\Delta r \cdot A = \Delta f = \frac{d\dot{m}}{dt}\,\Delta s$$

We can decompose the pressure p into the terms r (denoted here as inertial pressure) and $\hat{p}$ (denoted here as steady mass-flow pressure):

$$p = \hat{p} + r$$

This decomposition is useful since for gases, r is often very small. For liquids r can become sizeable but many liquids are in their properties (such as density or viscosity) relatively insensitive to r.

Hence, it makes sense to apply a different spatial resolution for $\hat{p}$ and r. We propose that the spatial discretization of $\hat{p}$ may be chosen as fine as the modeler wants but r is only discretized between boundaries and volume elements. The blue lines in Figure 3 show this discretization for our example.

The paper (Zimmer, 2018) demonstrates that if you apply this discretization scheme, $\hat{p}$ can be computed downstream and r results from the occurring differences in $\hat{p}$ by a linear equation system. If we also choose to use $\hat{p}$ and not p for the calculation of the thermodynamic properties, all non-linear thermodynamic computations in the total system can be brought into an explicit form.

Precondition is that the component models enable an explicit calculation of their outlet state from their inlet state and do not relate flows directly by algebraic equations. This precondition is not always easy to meet but it is practically possible (albeit with some effort).

In this way, we can achieve a fully explicit form by accepting an error that originates from neglecting the influence of r on the thermodynamic state. The error is often very small and is increasing in downstream direction until a volume or boundary is met. For steady mass-flow rates when $d\dot{m}/dt = 0$, the error is zero; see also (Otter, 2019) for a discussion of the error in a segmented pipe.

Another price for this explicit form is that all mass-flows now become state variables of the system. However, in many technical systems many components share the same mass flow, so this price is often acceptable. The implementation of this methodology in Modelica requires a new fluid library with its own connectors. Again, the paper (Zimmer, 2018) illustrates such a library called HEXHEX and the demonstration example of this paper has been built up solely with components from this library. Note that the modeler needs not concern itself with the discretization scheme. It becomes an inherent feature of the HEXHEX library.

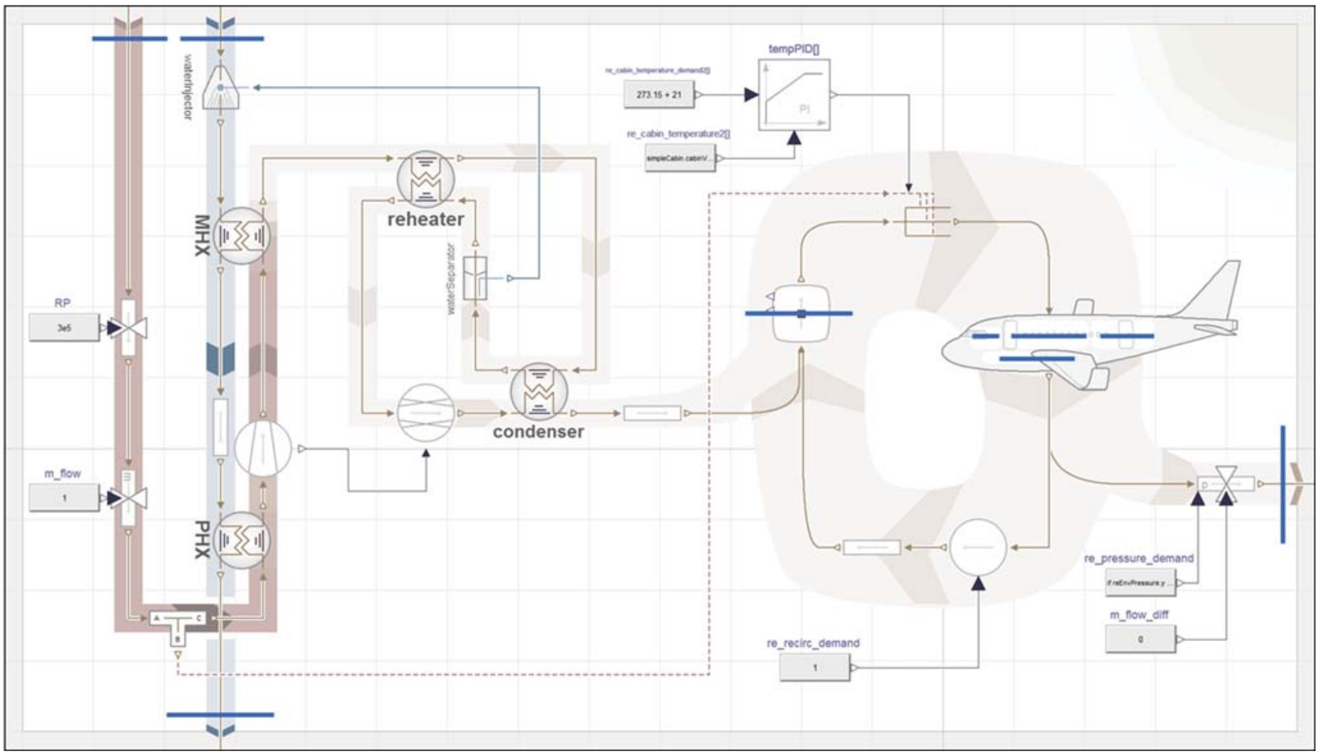


Figure 3: The blue bars indicate the discretization chosen for $\boldsymbol{r}$. In between these bars the influence of $\boldsymbol{r}$ on $\boldsymbol{p}$ is neglected.

3 Coping with Stiffness

Although the mass-flow dynamics help to avoid non-linear equation systems, they also create a problem on another level: typically they generate very fast transient towards equilibrium and hence corresponds to strongly negative eigenvalues. The system thus becomes stiff and explicit solvers can only be applied with very small step-size; often too small in order to meet the performance requirements.

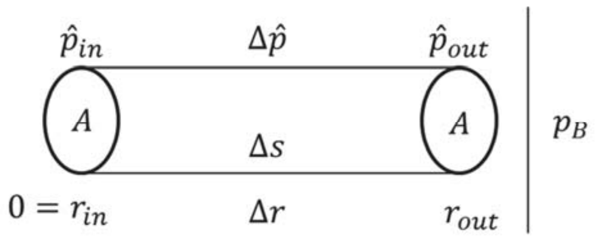


Figure 4: A simple pipe representing a generic 1-D system. The law for the pressure drop $\Delta\hat{p}$ can be arbitrary non-linear or time-dependent.

For better understanding we may look at a 1-dimensional, linearized system. Again: for a straight pipe as in Figure 4:

$$\frac{d\dot{m}}{dt} = \frac{A}{\Delta s}\Delta r$$

Δr now accounts for the pressure difference at the outlet boundary:

$$\Delta r = \hat{p}_{out}(\dot{m}) - p_B$$

As can be seen the outlet pressure is itself a function of the mass-flow. We can linearize this function for $\dot{m} = \dot{m}_0$ and retrieve:

$$\frac{d\dot{m}}{dt} = \frac{A}{\Delta s}\left(\hat{p}_{out}(\dot{m}_0) + (\dot{m} - \dot{m}_0)\frac{\partial\hat{p}_{out}(\dot{m}_0)}{\partial\dot{m}} - p_B\right)$$

Evidently, there are two sources of stiffness: the "inertia" of the mass flow, defined by its geometry $A/\Delta s$ and the sensitivity w.r.t. the steady mass-flow pressure $\hat{p}$: $\partial\hat{p}(\dot{m}_0)/\partial\dot{m}$ the latter shall also be negative in order to form a stable system (a condition that is typically fulfilled but can be violated for instance in the unstable area of a compressor).

Which of these two factors is dominant depends on the concrete (sub-)system at hand. However, distinguishing between two categories is helpful: there are

- uncontrolled mass-flows and
- controlled mass-flows.

For both categories, we will suggest means to manipulate the natural eigenvalues of the system into a range suitable for explicit solvers. Figure 5 shows the stability region of Runge-Kutta 3 and illustrates the idea of manipulation the set eigenvalues from $\boldsymbol{\lambda}$ to $\boldsymbol{\lambda}'$

The stability region is in grey and the area of interest for the actual dynamics is in orange. The eigendynamics of interest are correspondingly marked by green eigenvalues. The blue eigenvalues represent dynamics needed to compute the system but not of actual interest. These may prevent the application of an explicit solver and hence shall be manipulated to be within the stability region. The challenge is to only move specific eigenvalues without changing the complete dynamics.

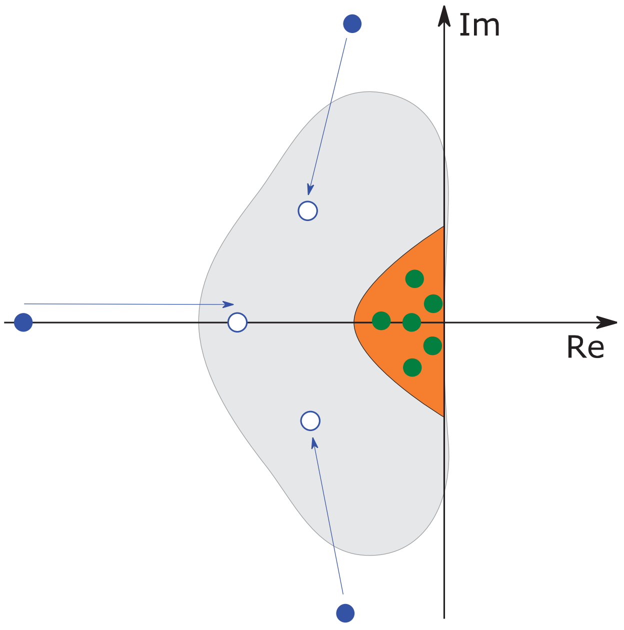


Figure 5: Manipulation of eigenvalues. Goal is to move the blue eigenvalues into the stability region of an explicit solver without influencing the green eigenvalues.

Without doubt such a manipulation of eigenvalues is an ugly thing to do but there is a price for achieving hard real-time capability. Loss of precision at fast transients is often an affordable price and hence can justify the manipulation of eigenvalues that are strongly negative or have a strong imaginary part.

Because it is an ugly practice, it is also often a little reported practice (although maybe quite commonly applied). But non-reporting does not make a practice irrelevant. It is better to report also on those things that sometimes just need to be done.

3.1 Coping with controlled mass-flows

Controlled mass-flows are typically part of actively driven systems. In our example in Figure 1, the bleed air mass-flows are controlled by valves and the recirculation mass-flow is controlled by a fan.

A typical valve model is an excellent example about the problems that are involved with controlled mass-flows. A classic implementation will regulate the valve opening (modeled here by linear resistance k) such that the measured mass-flow reaches the set-point:

$$\frac{d\dot{m}}{dt}\frac{\Delta s}{A} = \Delta r$$

$$\Delta\hat{p} = k \cdot \dot{m}$$

$$k \leftarrow \text{control input}$$

The inertial pressure then follows from the system composition. For a simple system we can assume that:

$$\Delta r = p_0 - \Delta\hat{p}$$

The eigenvalue of the system is then $-kA/\Delta s$ (at least if k is not directly dependent on $\dot{m}$ by the control input). Since k can become very large for a nearly closed valve, it is the dominating factor of the eigenvalue. Furthermore, this cannot be compensated by artificial increase of Δs, since k can also be very small if the valve is open.

To overcome this problem we propose to stipulate the mass-flow and in consequence make the valve a potential source of inertial pressure. The steady mass flow pressure drop $\Delta\hat{p}$ is then adopted so that Δr matches the fluid acceleration.

$$\frac{d\dot{m}}{dt}\,T = \dot{m}_{set} - \dot{m}$$

$$d\frac{\Delta\hat{p}}{dt}T = \frac{d\dot{m}}{dt}A/\Delta s + \Delta r$$

In this way, the eigenvalue corresponding to the mass flowrate can be arbitrarily stipulated to be $1/T$ with T being a parameter for this time constant.

Since this valve model is a source of inertial pressure, it can suppress all excitations of higher frequency. In consequence it violates the first law of thermodynamics for such high frequencies. The resulting error weakens for low frequencies and becomes zero for steady-state. When using such a model for a mass-flow control valve, one must thus ensure that excitations stay below a certain frequency. This, however, is almost a given for most real-time applications.

The method that has been shown here can also be adapted to other components used to control mass-flows such as compressors or fans. In a real implementation, one also has to deal with further constraints, for instance that valves must have a positive pressure drop $\Delta\hat{p}$.

3.2 Uncontrolled mass flows

Uncontrolled mass-flows are natural mass-flows that distribute the medium between volumes. Examples can be found again in Figure 1. The flow from the mixing unit to the cabin is assumed (in this model) as uncontrolled mass-flow. The flow between the different cabin zones and the flow from the cabin area to the cargo area also represent uncontrolled mass flows.

Again, it is a good idea to study a simple example: the flow between a volume through an orifice to a boundary element of fixed pressure p_0 . Using the bulk modulus K we can state the following equations:

$$\frac{d\hat{p}}{dt} = \frac{K}{V}\frac{\dot{m}}{\rho}$$

$$\frac{d\dot{m}}{dt} = \frac{A}{\Delta s}(\hat{p} - p_0 - k\dot{m})$$

where V is the volume and k the linear flow resistance of the orifice. The density is here only approximated by a constant ρ. The eigenvalues for this system are:

$$\lambda_{1,2} = \frac{k \pm \sqrt{k^2 - 4\,\frac{KA}{V\rho\Delta s}}}{2}$$

Here the difficulty for a real-time simulation is that the eigenvalues may be too strong in their imaginary part hence resulting in a high frequency behavior. This is for instance the case when small volumes are (almost) frictionless connected as small zonal elements in a cabin model. When the frequencies of such a model become too high for real-time simulation, we can only replace the actual dynamic behavior with a behavior representing a potential steady-state solution of the original model. To this end, we have to apply artificial damping.

The problem with just increasing the value of k to k' is that we may affect the steady-state solution. Fortunately, two neat little tricks help us to avoid this:

1. The artificial damping k' shall only be applied to the mass-flow attributed to the volume expansion and not applied to the mass-flow through the volume element.
2. The pressure drop resulting from the artificial damping k' is not attributed to the steady mass-flow pressure $\hat{p}$ but to r, the inertial pressure.

In this way, the steady-state solution remains unchanged.

For clarification, let us examine a volume model with two ports as illustrated by Figure 6. The mass of the volume follows from the mass balance

$$\frac{dM}{dt} = \dot{m}_1 + \dot{m}_2$$

The equation for mass-fraction follows from conservation of mass and the equation for the specific enthalpy follows from the conservation of energy. Mass, mass fraction and enthalpy then also stipulate the pressure $\hat{p}$. For $p = \hat{p} + r$ a term r can be added. To apply the proposed artificial damping we propose:

$$r = k' \frac{dM}{dt}$$

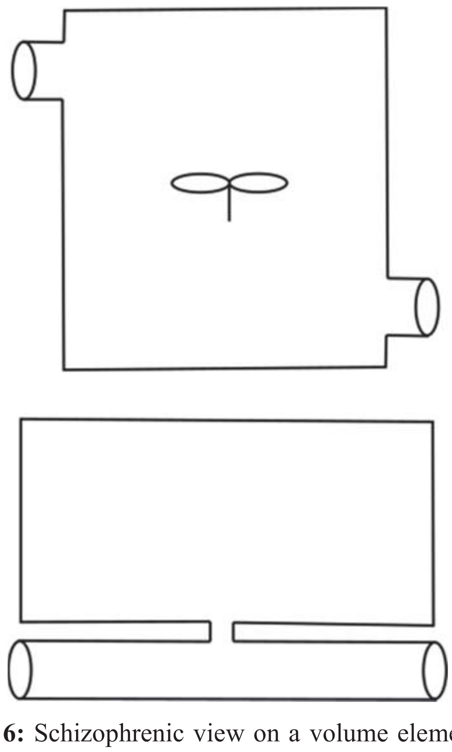


Figure 6: Schizophrenic view on a volume element with two ports. Top figure: in terms of enthalpy and mass fraction, the volume is idealized as an ideal mixing unit. Bottom figure: in terms of pressure and mass-flow rate, the volume is regarded as a frictionless pipe with an orifice to a larger expansion volume. The pressure drop of the orifice is only attributed to the inertial pressure $\boldsymbol{r}$ in order to dampen the dynamics of the mass flow.

There remains one last issue, the value for k' that is needed to achieve critical damping has the lower limit of $k' > 2\sqrt{KA/V\rho\Delta s}$. For a high frequent system, we thus simply replace strong imaginary eigenvalues by strong negative eigenvalues. This might be of no help for the real-time simulation. Hence, before implementing k', we may want to artificially lower the frequency. Manipulation Δs to a larger $\Delta s'$ will achieve exactly this without affecting the steady-state solution or the thermal properties of the system.

3.3 Additional remarks

As stated previously, the manipulation of eigenvalues is an ugly business and one is forced to sacrifice physical correctness to some degree.

Nevertheless, the methods shown above enable such a manipulation while upholding two important principles:

1. The solution for steady mass flows, meaning $d\dot{m}/dt = 0$ does not change.

2. Changes to the natural parameters do not involve any parameters for thermal capacities or the like.

For instance, another variant to fight stiffness would be to artificially increase volume sizes. But this would also influence the thermal capacity and the time constant for mixing processes. So this is not a good idea.

Altogether, the methods have their appeal but should be used with care.

4 Demonstration

Let us now simulate the aircraft ECS model of Figure 1 under hard real-time constraints. Figure 3 already illustrates that the concept of HEXHEX naturally avoids any non-linear equation systems as long as no component contains a non-linear equation system for the classic downstream computation.

Figure 7 highlights the components that have been manipulated from the original model in order to reduce the stiffness. The red elements have been manipulated for mass-flow control and the green volumes elements have been manipulated with an artificial damping on the mass-flow dynamics.

In total the system describes 52 continuous time states:

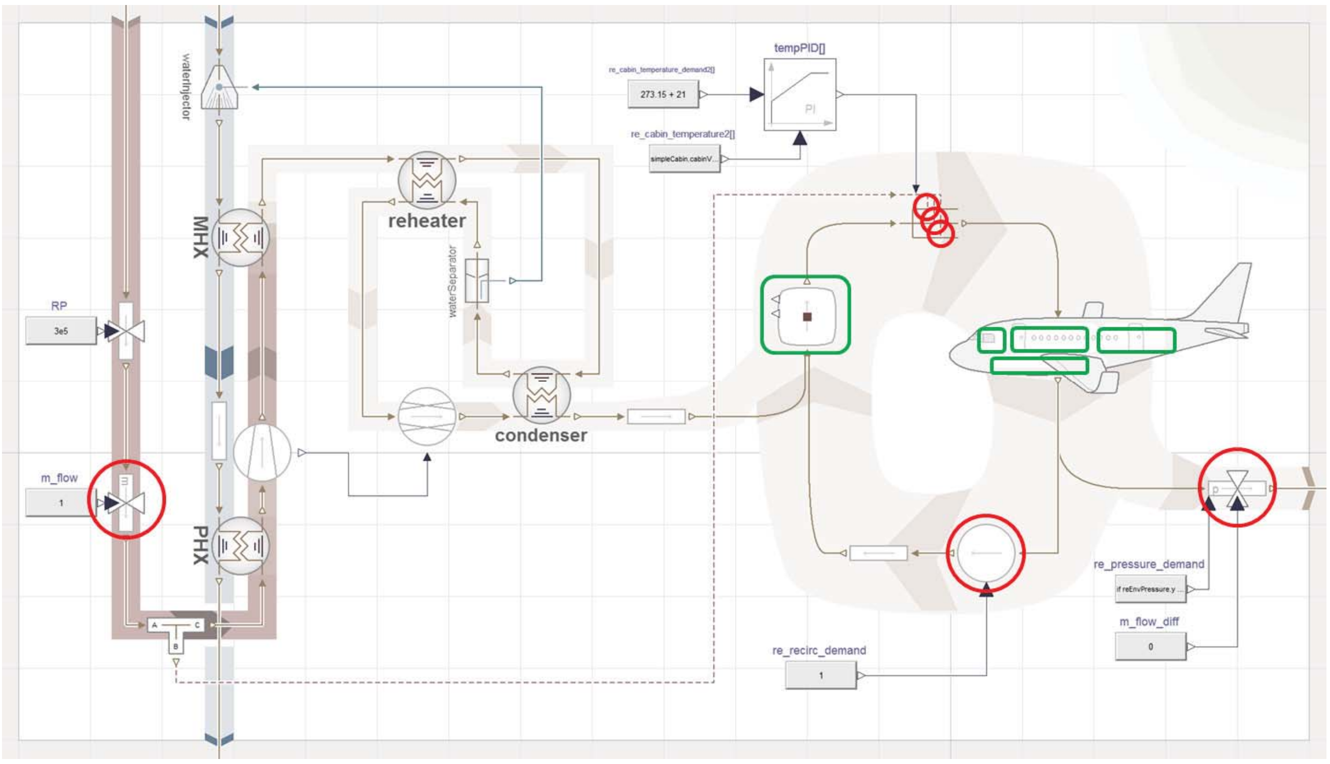


Figure 7: The red circles indicate components that have been manipulated for controlled mass flows. The green rectangles depict the volume elements where artificial damping has been applied.

- 6 states describe the aircraft altitude, the controllers for ram-air opening and zonal temperature.
- 15 states attribute to the pressure, enthalpy and water content of the 5 volume elements.
- 11 states describe the thermal dynamics of the heat exchangers and the cabin lining.
- 8 states describe the natural mass flows in the ram air channel, water separation and between the volume zones in the aircraft cabin as well as in the mixing unit.
- 12 states describe the 6 controlled mass flows and the corresponding pressure differences for the control component.

Since all components have been developed so that they enable an explicit computation of their outlet state from their inlet states, no non-linear equation system occurs in the whole translated model code.

The equations for the mass-flow dynamics result in 8 linear equation systems of size {5, 20, 15, 2, 8, 8, 8, 8, 8} that Dymola can reduce to the size to {0, 2, 3, 2, 3, 2, 2, 2} by the method of tearing.

Explicit Runge-Kutta 3 with a stepsize of 40ms (25Hz) is chosen as ODE solver. Figure 8 shows the eigenvalues of the system at t=10s. Evidently all eigenvalues are well within the boundaries of RK3 with the chosen step width. Even larger step widths could be chosen. Furthermore, no special effort is needed for initialization in this particular example.

In order to synchronize with real time, the corresponding block from the Modelica Device Drivers Library (Thiele 2017) has been used.

In terms of performance the simulation easily runs in real-time. Idle time (with deactivated outputs) is roughly 90% on an Intel Xeon 2,66GHz which indicates that the simulation on its own could run 10 times faster.

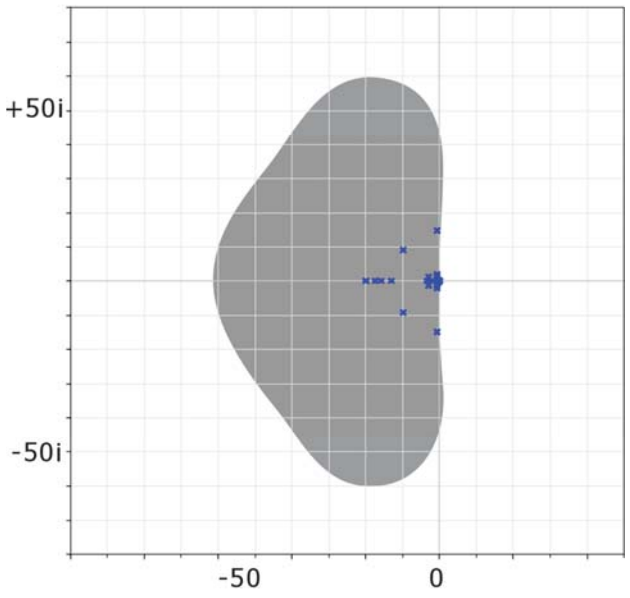


Figure 8: The eigenvalues of the system as blue crosses against the stability region of RK3 with h=0.04s

5 Outlook and Final Remarks

Although this new concept is a very promising path for hard real-time capability of thermofluid models, there are still a few missing links.

5.1 Missing parts

First of all, we shall not forget that we have only discussed the steps (A) and (B) of Figure 2. A more realistic test needs to be performed on real-time hardware (D) with a real-time operating system (C). However, it is a reasonable expectation that standard solutions will do the job.

Yet, there remain issues with the application of explicit solvers (B). So far, we have only shown for a few operating points that the eigenvalues are within the stability range of the chosen integration method. This is a too weak statement. Either this analysis must be performed for an extensive set of operating conditions or more favorably an algebraic analysis can be performed.

To this end, the simplistic 1-dimensional computation of eigenvalues as in section 3 must be replaced by a full multi-dimensional analysis for the system at hand. The model function:

$$d\boldsymbol{x}/dt = f(\boldsymbol{x}, \boldsymbol{u}, t)$$

must be bounded in its partial derivatives with respect to the mass-flow $\partial f / \partial \dot{m}$ for the complete state-space domain. Furthermore, strict statements are needed for all other eigenvalues.

Although such an analysis is by no means simple, it should be principally feasible under certain preconditions.

In terms of precision, the behavior at faster transient needs to be properly analyzed. So far, only a qualitative judgement of the simulation result was performed.

5.2 Future potential

The presented use case is still a relative simple example and hence it is yet unclear for how large or how complex system the proposed methodology can be successfully applied. For the future, more testing is required but for now, here are a few principal considerations.

First of all, the provided example already performs 10 times faster than real-time on a conventional desktop PC of 2012.

The model evaluation is certainly slowed down by the evaluation of the media models. Papers such as (Quoilin, 2014) suggest that there is substantial room for improvement.

The application of explicit Runge-Kutta makes parallelization an attractive choice. Especially those parts of the system that are separated by a volume element are attractive to compute in parallel.

Last but not least, also multi-rate integration (Ranade, 2014) becomes an attractive option since some thermal time constant can be simulated at slower rate.

Altogether, we estimate that the proposed methodology can be applied quite easily to systems 10-100 times larger (or more complex) than what has been presented here. A demonstration of this is, however, still pending.

5.3 Concluding Remarks

The inertial pressure (or its counterpart in terms of pressure-gradient force) is a very useful term to describe thermofluid systems without having to numerically solve non-linear equation systems. Only linear equation systems need to be solved during runtime. This can be done in predictable time.

The inertial pressure is also a very useful term to manipulate the eigenvalues of the system. In this way, the resulting model becomes suitable for the application of explicit ODE solvers such as Runge-Kutta.

In this way, the proposed methodology offers a promising way to model and simulate complex fluid systems under hard real-time constraints within a Modelica framework.

Acknowledgments

The author wants to thank Alexander Pollok for setting up the original demonstration example and Daniel Bender for his contributions to the HEXHEX library.

References

Casella, Francesco, et al., (2006) The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks, *Proc. 5th International Modelica Conference*, Vienna, Austria, Vol.2, pp.559-568.

Casella, Francesco and Sielemann, Michael und Savoldelli, Luca (2011) Steady-state initialization of object-oriented thermo-fluid models for homotopy methods. In: *Proceedings of 8th International Modelica Conference. 8th International Modelica Conference*, 20.-22. Mar 2011, Dresden.

Otter, Martin et al., (2019) Thermodynamic Property and Fluid Modeling with Modern Programming Language Constructs. In: *Proceedings of 8th International Modelica Conference. 13th International Modelica Conference*, Mar 04-06, 2019, Regensburg, Germany.

Pollok, Alexander und Schröffer, Andreas (2017) Simulation of Helmholtz Resonance Effects in Aircraft ECS. In: *6th International Workshop on Aircraft System Technologies. 6th International Workshop on Aircraft System Technologies*, 21.-22. Feb. 2017, Hamburg, Deutschland.

Quoilin, Sylvain et. al. (2014) ThermoCycle: A Modelica library for the simulation of thermodynamic systems In: *Proceedings of the 10th International Modelica Conference* 2014, Lund, Sweden.

Ranade, Akschay, Francesco Casella (2012), Multi-rate integration algorithms: a path towards efficient simulation of object-oriented models of very large systems. *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)* , Berlin, Germany

Schulze, Christian (2011) Real-Time Simulation of Vapour Compression Cycles. In: *Proceedings of 8th International Modelica Conference*., 20.-22. Mar 2011, Dresden.

Thiele, Bernhard, et. al. (2017) Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library. In: *Proceedings of 8th International Modelica Conference*. 12th International Modelica Conference, Prague, Czech Republic

Wetter, Michael, et al. (2014), The Modelica Buildings Library. In: *Journal of Building Performance Simulation*, 7(4), 253–270, 2014

Zimmer, Dirk, (2013), Using Artificial States in Modeling Dynamic Systems: Turning Malpractice into Good Practice. In: *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)* , Nottingham, United Kingdom

Zimmer, Dirk and D. Bender, A. Pollok (2018) Robust Modeling of Directed Thermofluid Flows in Complex Networks. In: *Proceedings of the 2nd Japanese Modelica Conference*, Tokyo, Japan

Thermodynamic Property and Fluid Modeling with Modern Programming Language Constructs

Martin Otter[1] Hilding Elmqvist[2] Dirk Zimmer[1] Christopher Laughman[3]

[1]DLR - Institute of System Dynamics and Control, Germany
{martin.otter, dirk.zimmer}@dlr.de
[2]Mogram AB, Magle Lilla Kyrkogata 24, 223 51 Lund, Sweden, Hilding.Elmqvist@Mogram.net
[3]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA, laughman@merl.com

Abstract

Modelica is used extensively to model thermo-fluid pipe networks. Experience shows that Modelica models in this domain have limitations due to missing functional expressiveness of the Modelica language. In this paper, a prototype is described that demonstrates how thermodynamic property and thermo-fluid pipe component modeling could be considerably enhanced via modern language constructs. This prototype is based on the Modia modelling and simulation prototype and relies on features of the Julia programming language. It utilizes some key ideas of Modelica.Media, and part of Modelica.Media was semi-automatically translated to Julia.

Keywords: Modelica, Modia, Julia, Modelica.Media, Modelica.Fluid, ModiaMedia, thermodynamic property models, thermo-fluid models

1 Introduction

Thermodynamic property models (abbreviated as Media models in the rest of this article) require a great deal of flexibility with regards to the choice of thermodynamic and dynamic states to achieve robust and fast simulations. These medium models need functions to describe thermodynamic relationships with different inputs and differential equations to describe dynamic behavior. When such medium models using the Modelica language were first introduced, the only mechanism available that satisfied these requirements was that of a replaceable Modelica package (Elmqvist, et al. 2003). Special constructs for functions were also added to enable media modeling. This use of packages was not part of the initial Modelica language design, however, as they were primarily intended for the organization of model components. As a result, compilers typically handle packages completely at compile time. This fact has several significant implications, such as the restriction from changing the medium or the level of detail of the medium model during simulation.

This paper investigates alternative media and fluid modelling architectures available in the modern programming language Julia (Bezanson, et al. 2017). Mechanisms of interest instead of replaceable packages include member functions, function references, and multiple dispatch[1]. The resulting architecture provides more dynamic flexibility and uses common language constructs so that it is easier to understand and maintain.

The design of the fluid library for Modia is based on a new approach by (Zimmer et al. 2018). This approach is currently used in aircraft industry and enables the robust modeling of fluid streams and avoids the creation of large non-linear equation systems that can be a major source of problems for conventional fluid libraries in Modelica.

2 Thermodynamic Property Models

2.1 Users view

A medium model consists of a data structure that holds the data of the medium and a set of functions operating on this data. The fluid properties are computed from a set of variables called the *thermodynamic states* of the medium. For example, the *thermodynamic states* of the ideal-gas moist air model Modelica.Media.Air.MoistAir are temperature T, pressure p, and the mass fraction of water X, because all other quantities can be computed from them.

A fluid component model, such as a volume model, defines independent variables called *model states* that describe the differential equations of a component model as functions of these states. For example, if a medium is used only in a single phase region, often pressure p and temperature T are used as states of the model, whereas pressure p and specific enthalpy h might be used if the medium enters the two-phase region. Other choices, such as pressure p and density d, may also be necessary to address application-specific requirements (Laughman, Qiao 2016). All media models in Modelica.Media therefore have various possibilities for *model states*, including (p,T), (p,h), (p,s), and (d,T), as well as mass fractions X.

[1]Multiple dispatch in Julia means that method selection is based on the types of *all* non-optional function arguments (if possible at compile-time, otherwise at run-time).

In general, fluid properties are computed by (a) transforming *model states* into medium-specific *thermodynamic states*, and (b) executing medium-specific functions having the medium-specific *thermodynamic states* as input arguments. A programming language that supports member functions can usually implement such a scheme in a reasonable way.

Modelica does not support member functions and therefore the definition of media is awkward and limited. Julia does not have member functions, but instead supports multiple dispatch to select the desired function based on the types of all (non-optional) function arguments (rather than base the selection on a single input argument, as in object-oriented programming languages).

The implementation of this media modeling approach, called ModiaMedia, is available on Github[2]. From a user's point of view, a medium is an object (an instance of a Julia immutable struct) that is returned by function getMedium(..), given the medium name as a string. Example:

```
using ModiaMedia
Medium = getMedium("Air")
```

All media models are stored in a dictionary and getMedium(..) inquires the medium from this dictionary and returns the reference to it. In a second step, the *thermodynamic state* of the medium is computed from the desired independent variables of the application, for example,

```
state = setState_pT(Medium, 1e5, 300)
```

Here the thermodynamic state of the Air medium is computed from p = 10^5 Pa and T = 300 K. Given the thermodynamic state, functions are provided to compute other desired medium properties. For example, the density and specific enthalpy of air can be computed by

```
d = density(Medium, state)
h = specificEnthalpy(Medium, state)
```

An alternative implementation of setState_pT(..) could store a reference to Medium in state. This would then simplify the further access calls, for example: d = density(state). This has not been implemented, as there are open questions about this approach.

In an object-oriented programming language, the syntax would be:

```
d = Medium.density(state)
h = Medium.specificEnthalpy(state)
```

Functions are also provided to plot properties of the medium. In the current version of this project, the most important characteristics of a medium are plotted with the call standardPlot(Medium). This interface is identical for all media. Figure 1 illustrates the (current) standard plot for Air.

In addition to constants and functions, a medium package in Modelica.Media also defines a Modelica *model* called BaseProperties that computes the properties of a medium needed for mass and energy balances. Since ModiaMedia is a standalone package that does not depend on Modia and can be used in other standalone modeling environments, no equivalent Modia model to BaseProperties is defined in ModiaMedia.

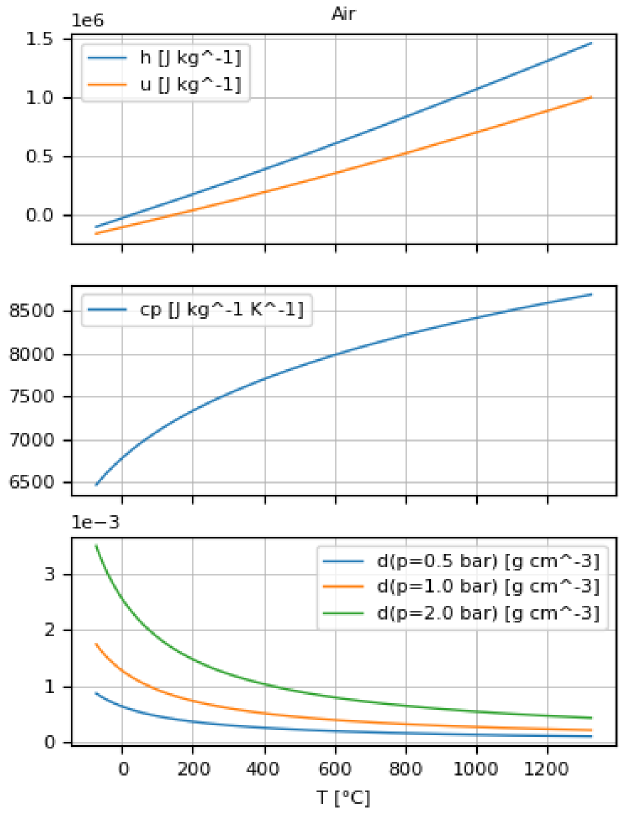


Figure 1. Result of: standardPlot(getMedium("Air")).

2.2 Structure of the ModiaMedia package

The Julia package ModiaMedia has many features in common with Modelica.Media, but is based on a hierarchical type system that allows for greater simplicity and flexibility. The abstract type system of ModiaMedia is a direct mapping of the Modelica.Media class hierarchy:

```
abstract type AbstractMedium end
abstract type PureSubstance <: AbstractMedium end
abstract type MixtureMedium <: AbstractMedium end
```

The above definitions state that PureSubstance and MixtureMedium are subtypes of AbstractMedium. A Medium model is defined as a medium-specific Julia struct that is either a direct or indirect subtype of AbstractMedium and has the following structure:

[2] https://github.com/ModiaSim/ModiaMedia.jl

```
struct MediumTypeName <: AbstractMedium
  infos::FluidInfos
  fluidConstants::SVector{1,AbstractFluidConstants}
  fluidLimits::FluidLimits
  data::Any        # fluid spec. data
end
```

where

- FluidInfos contains all constants that are similarly defined for a Modelica.Media package (such as mediumName and singleState),
- fluidConstants contains all the data of the fluidConstants vector of records of the equivalent Modelica.Media package,
- fluidLimits defines the validity range of the medium model and
- data contains additional fluid specific data.

Example:

```
struct SingleGasNasa <: PureSubstance
  infos::FluidInfos
  fluidConstants::SVector{1, IdealGasFluidConstants}
  fluidLimits::FluidLimits
  data::SingleGasNasaData

 function SingleGasNasa(...)
   # Constructor function
   ...
  end
end
```

The functions that are available for an AbstractMedium are defined in the following form:

```
density(m::AbstractMedium,
        state::ThermodynamicState) = error("undefined")

specificEnthalpy(m::AbstractMedium,
        state::ThermodynamicState) = error("undefined")
```

where ThermodynamicState is the abstract super type of all thermodynamic states.

A medium model must provide concrete implementations for these functions, e.g.,

```
density(m::SingleGasNasa, state:: State_pT) =
                                    state.p/(m.data.R*state.T)
```

In summary, while ModiaMedia models store analogous data to that which is contained in a Modelica medium package, it is stored in a hierarchical data structure. In comparison, all data is stored in form of constants inside a Modelica package. The benefit of the hierarchical data structure is that this data can be passed as argument to a function allowing a user to easily add functionality for pre- and post-processing. Since a Modelica medium model is actually a package, and Modelica does not support functions that can operate on packages, Modelica medium models can be used for simulation only and it is not possible to easily implement other functions, such as those that plot data of a Modelica medium.

It should be noted that there are several thermodynamic property packages available where medium models are defined with objects and member functions implemented in a programming language such as C++, e.g., FluidProp[3], CoolProp[4], and TILMedia[5]. In comparison with these packages, ModiaMedia is a very early prototype to experiment with a tight integration of a thermodynamic property package with fluid component modeling to achieve fast simulations of transient thermodynamic processes.

2.3 Conversion of Modelica to Modia/Julia

The Modelica Standard Library has a rich set of media models, containing data, functions for thermodynamic properties calculation, table lookup and interpolations, and basic media model equations. Each medium is represented as a Modelica package. To utilize the extensive knowledge and effort encoded in this library, a translator[6] performing source-to-source transformation from Modelica to Modia/Julia has been written in Julia. It has a recursive descent handwritten LL(2) parser. Each grammar production of Modelica (Modelica Association 2017, Appendix B) is represented by a Julia function. Example:

Modelica grammar production:

```
extends-clause :
   extends type-specifier
   [ class-modification ]
   [ annotation ]
```

Julia function:

```
function extends_clause(env)
  expect("extends")
  type_specifier(env)
  if nextItem == "("
    class_modification(env)
  end
  if nextItem == "annotation"
    annotation(env)
  end
end
```

A scanner updates global variables nextItem and nextType. The function expects checks nextItem and if found, scans the next item. The First and Follow sets used in LL parsers have been determined manually and are used to select productions/functions and to end repetition. The variable env is used to transfer which output file is used, indentation level, etc.

[3] http://www.asimptote.nl/software/fluidprop
[4] http://www.coolprop.org/
[5] https://www.tlk-thermo.com/index.php/en/software-products/tilmedia-suite
[6] https://github.com/ModiaSim/ModiaFromModelica

Top level packages and classes of Modelica are translated to Julia modules, while subpackages cannot be converted to modules because cyclic dependencies between Julia modules, such as exist between Modelica subpackages, are not allowed. These subpackages are therefore removed and flattened by introducing long names such as: `Modelica_Blocks_Interfaces_SISO` for the content.

Models, connectors, blocks are converted to Modia `@model` macros, while records are converted to mutable structs. Julia supports unit handling also in functions using the package Unitful (Keller, et al 2018).

Since expressions in Modelica are quite similar to those in Julia, there is no need to introduce an abstract syntax tree; as a result, the corresponding Julia text can be generated directly by the parsing function. Examples of slightly different syntax:

```
{1,2,3} → [1,2,3]
[1,2,3; 4,5,6] → [1 2 3; 4 5 6]
```

While regular expression substitution could be considered for expressions, if-then-else expressions pose problems due to the need to introduce `end` in Julia:

```
if a then b else c ⟶ if a; b else c end
```

(One could have used the syntax `a ? b : c` instead.)

Variable declarations in models have a different structure in Modelica and Modia:

```
Real x[10] = ones(10) ⟶
                  x = Float(size=(10,), start=ones(10))
```

This means that information needs to be moved from one parsing function to another. This is accomplished by temporarily building small ASTs using tuples.

In comparison, syntax for declarations in functions is quite different, e.g.,:

```
Real x[10] = ones(10) ⟶ x::Array{Float64,1} = ones(10)
```

2.4 Conversion of Modelica.Media to Modia/Julia

Since the Medium definitions in Modelica and in Modia are quite different, it is not yet possible to fully automatically transform a Modelica medium package in an equivalent ModiaMedia model. Instead, the Julia code generated by the translator is currently semi-manually transformed into the desired form with the help of an editor that supports regular expressions.

For example, the `SingleGasNasa` coefficients that have been transformed from Modelica to Julia are defined as:

```
const Ag = IdealGases.Common.DataRecord(
   name="Ag",
   MM=0.1078682,
   ...)
```

With an editor (defining the changes with regular expressions), all 1200 definitions have been automatically transformed to the assignment of dictionary entries:

```
singleGasesData["Ag"] =
     IdealGases_Common_DataRecord(
        name="Ag",
        MM=0.1078682,
        ...)
```

This dictionary is then serialized and stored so that these medium data can be quickly loaded by the user, rather than be regenerated on every use.

A Modelica medium function of the form:

```
redeclare function extends density
algorithm
   d := state.p/(data.R*state.T);
end density;
```

can be changed to the equivalent ModiaMedia model function:

```
density(m::SingleGasNasa, state::state_pT) =
                              state.p/(m.data.R*state.T)
```

via the following rules:

1. Add the medium instance `m` as the first argument to the function.
2. Add the appropriate type information for the input and return variables.
3. Prepend `m` to all variable data, e.g., "`data.Hf`" is changed to "`m.data.Hf`".

We plan to transform the complete Modelica.Media package to ModiaMedia. The base Modelica package has already been transformed to Julia and the somewhat labor-intensive semi-manual final adaptation is currently on the way.

3 Fluid Component Models

In general, our objective is to model and simulate thermo-fluid pipe networks, such as heat exchangers, air conditioning systems, distillation columns, or power plants. Traditional simulation programs in this field tightly couple the equations of the fluid components to the equations of the medium that is flowing in the components. Modelica.Fluid was designed to increase the flexibility of these models by separating the model of the fluid component from the medium model, enabling the use of a pipe model for media that have different thermodynamic states. The Modia fluid prototype continues to pursue the simplification and generalization of the Modelica.Fluid approach.

There are different ways to formulate fluid network models, depending on the application and the properties of the fluid that need to be taken into account. To experiment with simpler and more robust network models, the new method from (Zimmer et al. 2018) is used as basis for the fluid component models and has been implemented in a small experimental library named ModiaFluid for unidirectional, 1D thermo-fluid pipe flow that is suited for incompressible media or for compressible media if Ma ≤ 0.3. The Modelica.Fluid library has a similar application area but supports bidirectional fluid flow.

3.1 Users view

A simple pipe network is shown in Figure 2.

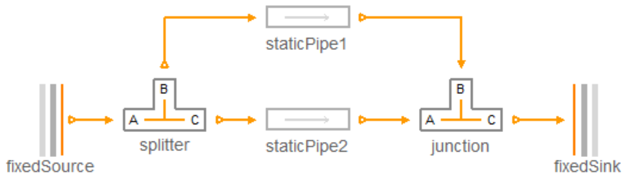


Figure 2. Simple pipe network with splitter and junction.

With the ModiaFluid library, this network is defined as:

```
const air = getMedium("Air")
@model PipeSystem begin
   fixedSource = FixedSource(Medium=air,
                                          p0=1.0e5, h0=10000,)
   fixedSink    = FixedSink(p0=0.8e5)
   staticPipe1 = StaticPipe(k=3e5, l=1.0, A=0.01)
   staticPipe2 = StaticPipe(l=1.0, A=0.01)
   junction      = Junction()
   splitter       = Splitter()
   @equations begin
      connect(staticPipe2.outPlug, junction.inPlugA)
      connect(staticPipe1.outPlug, junction.inPlugB)
      connect(junction.outPlugC, fixedSink.inPlug)
      connect(splitter.outPlugB, staticPipe1.inPlug)
      connect(splitter.outPlugC, staticPipe2.inPlug)
      connect(fixedSource.outPlug, splitter.inPlugA)
   end
end
```

Currently, Modia has no graphical user interface, and the system must be manually defined with the textual definition above. The definition on this level looks close to a corresponding Modelica model. The essential difference is that the medium model is defined only at one component (because the medium is propagated through the connection structure), whereas in Modelica it must be defined for every component.

3.2 Fluid connectors

The ModiaFluid library currently supports only unidirectional fluid flow. This assumption is already built into the connectors that are defined corresponding to (Zimmer et al. 2018):

```
MediumVariable() = Variable(size=())

@model InPlug(:connector) begin
   Medium = MediumVariable()
   m_flow = Float(flow = true)
   r  = Float()
   p = Float(input=true)
   h = Float(input=true)
end

@model OutPlug(:connector) begin
   Medium = MediumVariable()
   m_flow = Float(flow = true)
   r  = Float()
   p = Float(output=true)
   h = Float(output=true)
end
```

The variables have the following meaning:

- Medium is a reference to the medium data structure of section 2 and defines the medium that is flowing through the connector. This reference is propagated through the connection structure by means of alias elimination and is treated as one Modia variable in the symbolic engine. Note, a Modia variable can be any Julia data structure.
- m_flow is the mass flow rate into the connector,
- r is the pressure that is used to accelerate the fluid (see section 3.4),
- p = staticPressure - r, and
- h is the specific enthalpy.

A connector is modelled as a Modia @model macro with the Symbol :connector as macro parameter. Note that p and h are declared as either input or output.

Formally, an InPlug connector can only be connected with an OutPlug connector and not with another InPlug connector, so there are restrictions how components can be connected together.

3.3 Medium propagation

In the connectors of section 3.2, Medium is a Modia variable, where the type of the variable is not yet defined but will be deduced by type inference. At one or at several components, this variable is redefined to an instance that is a subtype of AbstractMedium. Example:

```
@model FixedSource begin
   Medium = MediumVariable()
   outPlug  = OutPlug()
   state      = Variable()
   ...
@equations begin
   outPlug.Medium = Medium
   state = setState_ph(Medium, outPlug.p, outPlug.h)
   d      = density(Medium, state)
  ...
end

const air = getMedium("SimpleAir")

@model PipeSystem begin
```

```
  source=FixedSource(Medium=air)
  ...
end
```

In model FixedSource, the Medium variable must be redefined when the component is used. This is performed by first generating a medium model with

```
air = getMedium("SimpleAir")
```

and then using air as modifier to the FixedSource instance:

```
source=FixedSource(Medium=air)
```

In the FixedSource component, an equation outPlug.Medium = Medium is present. Furthermore, the Medium variable might be used to compute medium properties such as the density d.

Modia treats Julia structs (such as variable Medium) specially: Struct variables can be used as modifier or as variable in an equation "var1 = var2". When propagating a reference in this way, an overdetermined system of equations can occur (when connections form a loop, or when the same medium is defined at several components). This issue is automatically resolved by extended alias elimination.

Since the media in Modelica.Media are packages and Modelica cannot use packages in equations and also does not have special language elements to propagate packages in connections, a medium has to be defined in every component where it is used. Therefore, if a pipe system has 20 components, then the medium needs to be defined 20 times.

In ModiaFluid, a medium can be defined in one component model and is then propagated through all components and connections where the fluid is flowing. The current implementation of medium propagation has however the temporary limitation that a medium must also be defined at all volumes, for details see section 3.5. In principal, this mechanism would allow changes to the medium during simulation as long as the same BaseProperties models (see section 3.5) are used.

3.4 Momentum balance

Modelica.Fluid utilizes the steady-state *or* the dynamic momentum balance depending on the chosen option. In ModiaFluid the approach from (Zimmer et al. 2018) is used to achieve more efficient and more robust simulations. Hereby, the unsteady Bernoulli equation is the starting point (Zimmer et al. 2018, Schade et al. 2013 eq. 4.4-3, Brennen 2006 section Bnda[7])[8].

[7]http://brennen.caltech.edu/FLUIDBOOK/basicfluiddynamics/Unsteadyonedimensionalflow/Unsteadybernoulli/unsteadybernoulli.pdf

[8] The unsteady Bernoulli equation is derived by integrating the Euler equations for incompressible fluid flow along a stream line. The Euler equations in turn are the differential form of the momentum balance that is used in Modelica.Fluid.

The approach is sketched with the simple example shown in Figure 3, where three pressure drop components (for example pipes, orifices, bends) are connected between two volumes and fluid flows from volume1 to volume2.

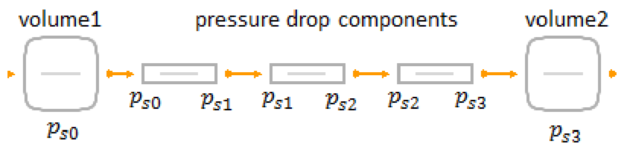


Figure 3. Three pressure drop components connected between two volumes ($\boldsymbol{p_{si}}$ is the static pressure at the indicated location)**.**

For simplicity of the derivation the specific kinetic energy is neglected. Assume that all pressure drop components have the same area A and component i has length Δs_i, that m_{flow} is the mass flow rate, p_{si} is the static pressure at the indicated location and $\Delta p_{i-1,i}$ is the pressure drop correlation of component i. The unsteady Bernoulli equation can then be formulated as:

$$\frac{dm_{flow}}{dt}\cdot\frac{\Delta s_1}{A}+p_{s1}-p_{s0}=\Delta p_{01}(m_{flow},p_{s0},h_0)$$
$$\frac{dm_{flow}}{dt}\cdot\frac{\Delta s_2}{A}+p_{s2}-p_{s1}=\Delta p_{12}(m_{flow},p_{s1},h_1)$$
$$\frac{dm_{flow}}{dt}\cdot\frac{\Delta s_3}{A}+p_{s3}-p_{s2}=\Delta p_{23}(m_{flow},p_{s2},h_2)$$

The specific enthalpies h_i are separately computed, e.g., for isenthalpic pressure drop components the h_i are the upstream specific enthalpies ($h_2 := h_1 := h_0$).

Note, in the Modelica.Fluid library the momentum balance is used in the form[9]

$$\frac{dI_s}{dt}+(p_{s1}-p_{s0})\cdot A=\Delta p_{01}\cdot A$$
$$I_s = m_{flow}\cdot\Delta s_1$$

As can be seen, this is exactly the unsteady Bernoulli equation multiplied with A.[10] So, the starting point of the derivation below are exactly the same equations as used in the Modelica.Fluid library.

The static pressures are now split into two parts:

$$p_{si} := p_i + r_i$$

where r_i is the pressure that is used to accelerate the fluid and p_i is the remaining part of the pressure. In steady state operation, $r_i := 0,\ p_{si} := p_i$. Introducing

[9] Modelica.Fluid.Interfaces.PartialDistributedFlow

[10] The unsteady Bernoulli equation has, however, the advantage that in its general form it holds along a streamline, so also for bends and orifices. The momentum balance along a streamline includes the (unknown) reaction forces on the component and therefore it can only be used in equations for a straight pipe, where the reaction forces in direction of the pipe flow are zero.

these terms in the unsteady Bernoulli equations and utilizing the abbreviation $\Delta r_{i-1,i} = r_i - r_{i-1}$ results in:

$$\frac{dm_{flow}}{dt} \cdot \frac{\Delta s_1}{A} + \Delta r_{01} + p_1 - p_0 = \Delta p_{01}$$
$$\frac{dm_{flow}}{dt} \cdot \frac{\Delta s_2}{A} + \Delta r_{12} + p_2 - p_1 = \Delta p_{12}$$
$$\frac{dm_{flow}}{dt} \cdot \frac{\Delta s_3}{A} + \Delta r_{23} + p_3 - p_2 = \Delta p_{23}$$

Since the r_i are defined to solely accelerate the fluid, the equations can be split into two parts:

$$p_1 - p_0 = \Delta p_{01}(m_{flow}, p_{s0}, h_0)$$
$$p_2 - p_1 = \Delta p_{12}(m_{flow}, p_{s1}, h_1)$$
$$p_3 - p_2 = \Delta p_{23}(m_{flow}, p_{s2}, h_2)$$
$$\frac{dm_{flow}}{dt} \cdot \frac{\Delta s_1}{A} = -\Delta r_{01}$$
$$\frac{dm_{flow}}{dt} \cdot \frac{\Delta s_2}{A} = -\Delta r_{12}$$
$$\frac{dm_{flow}}{dt} \cdot \frac{\Delta s_3}{A} = -\Delta r_{23}$$

Furthermore, we have the boundary conditions at the volumes:

$$p_{s0} = p_0 \quad (r_0 = 0)$$
$$p_{s3} = p_3 + r_3$$

No approximations have been introduced so far (the original equations have been just reformulated). Now, the *approximation* is made, that the dependency of the pressure drop equations on the *inertial pressure r is neglected*:

$$\Delta p_{i-1,i} = \Delta p_{i-1,i}(m_{flow}, \boxed{\boldsymbol{p_{i-1}}}, h_{i-1})$$

Note that the pressure drop equations are typically determined only for steady-state operations, and that the relationships/equations that take the acceleration of a fluid into account are often not known. In particular, all the pressure drop correlations used in Modelica.Fluid hold only for steady-state operations.

The big advantage of this slight approximation is that the equations are now decoupled, as described in the following. First, the pressures p_i can be computed in a forward sequence because the static pressures and the specific enthalpies at the volumes are known, as well as the mass flow rate m_{flow} (since its derivative appears in the unsteady Bernoulli equation, m_{flow} is a state):

$$p_1 := p_0 + \Delta p_{01}(m_{flow}, p_0, h_0)$$
$$p_2 := p_1 + \Delta p_{12}(m_{flow}, p_1, h_1)$$
$$p_3 := p_2 + \Delta p_{23}(m_{flow}, p_2, h_2)$$

The remaining equations then form a *linear* system of equations and the coefficients of the system matrix are constants:

$$\begin{bmatrix} \frac{\Delta s_1}{A} & 1 & 0 & 0 \\ \frac{\Delta s_2}{A} & 1 & -1 & 0 \\ \frac{\Delta s_3}{A} & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{dm_{flow}}{dt} \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} p_{s0} \\ 0 \\ 0 \\ p_{s3} - p_3 \end{bmatrix}$$

The result is that the interconnection of pressure drop components results only in a *small linear equation system* with a constant coefficient matrix. Since this matrix is constant, it is sufficient to perform an LU decomposition once and then reuse it during the simulation. If this approximation is not performed as in the Modelica.Fluid library, *nonlinear algebraic equation systems* often appear that can be either hard to solve (especially, when starting at $m_{flow} = 0$) or in which these equation systems have no unique solutions.

For these reasons, there is a "rule of thumb" in the Pipe-network community to always have a volume between two pressure drop components. In Modelica.Fluid there is, for example, the option modelStructure in the DynamicPipe model to define whether the pipe ends with a volume on either of the ports or not, in order to avoid such nonlinear equation systems. Obviously, such options are only for fluid specialists. When using the new method, such advanced options are no longer needed.

In (Zimmer et al. 2018) more complicated cases with branching and joining piping networks are additionally discussed. Here, the dynamic momentum balance is also taken into account, whereas in Modelica.Fluid only junction models with steady-state momentum balances are provided.

In the ModiaFluid library the new method is used in all components, including junctions. For example, a pipe model is defined in the following way, directly utilizing the equations explained above:

```
@model ShortPipe begin
  inPlug  = InPlug()
  outPlug = OutPlug()
  …
@equations begin
  # Medium propagation
  outPlug.Medium = inPlug.Medium

  # mass flow balance
  m_flow = inPlug.m_flow
  inPlug.m_flow + outPlug.m_flow = 0

  # Propagation of specific quantities
  outPlug.p = inPlug.p + dp
  outPlug.r = inPlug.r + dr
  outPlug.h = inPlug.h + dh
  dp = -m_flow*abs(m_flow)*k  # pressure drop
  dr = -der(m_flow)/A*l        # inertial pressure
```

```
    dh = 0.0                        # isenthalpic process
  end
end
```

In the Modelica.Fluid package there are many options that can be set either at the component level or globally. For example, the user can choose either a steady-state or dynamic momentum balance, the presence of a volume on either port of a pipe, or the definition of pressure drop components as functions of the pressure difference or as functions of the mass flow rate.

In ModiaFluid the complexity of the code and of the options is drastically reduced by providing only the dynamic momentum balance, by describing pressure drop components as function of mass flow rate, and by having only one discretization scheme for a pipe. The simulation is also potentially more robust than when defined with Modelica.Fluid, because no nonlinear algebraic equations occur, even if pressure drop components are connected together without a volume in between.

3.5 Mass and energy balance

The definition of mass and energy balances are essentially analogous to the approach used in Modelica.Fluid. With the specific internal energy u, density d, volume V, mass m, internal energy U, the sum of the mass flow rates into the volume m_{flow}, the sum of the enthalpy flow rates in to the volume H_{flow}, the contribution due to the unsteady movement of the fluid H_r, and the pressure p and specific enthalpy h as the independent variables of the utilized medium model, the balance equations can be formulated as:

$$d = d_{medium}(p,h)$$
$$u = u_{medium}(p,h)$$
$$m = d \cdot V$$
$$U = m \cdot u$$
$$\frac{dm}{dt} = m_{flow}$$
$$\frac{dU}{dt} = H_{flow} + H_r$$
$$H_r = \sum_{i \in inflowing} m_{flow,i}\, r_i / d$$

Since each inlet i of a volume forms a boundary for the pressure, a pressure difference may occur between the volume and the inlet pressure p. This difference is accounted by $r_i = p_s - p_i$. Since the volume work $V_{flow,i} r_i = m_{flow,i}\, r_i / d$ of this pressure gradient is accounting for the acceleration (or deceleration) of the inflowing fluid, the enthalpy of the inflowing fluids needs to be corrected by the term H_r. This is not necessary for the outlets since for the outlets, the term r is zero by definition.

If the @model equations would be defined in this way, then m and U would be selected as states and the independent medium variables, for example (p,h) or (p,T) would be in general determined by solving nonlinear equation systems. The approach is to rewrite the equations. In Modelica.Fluid this is performed by providing the attribute StateSelect.prefer on the desired states. The goal in Modia is to arrive at a simpler language as Modelica and therefore an attribute StateSelect is not supported. Instead, the derivatives are manually expanded until the state derivatives appear. For example, (p,h) shall be used as states. The mass- and energy balance can then be reformulated to (= manual index reduction):

$$d = d_{medium}(p,h)$$
$$u = u_{medium}(p,h)$$
$$\text{der\_d} = \frac{\partial d_{medium}}{\partial p}\dot{p} + \frac{\partial d_{medium}}{\partial h}\dot{h}$$
$$\text{der\_u} = \frac{\partial u_{medium}}{\partial p}\dot{p} + \frac{\partial u_{medium}}{\partial h}\dot{h}$$
$$m = d \cdot V$$
$$U = m \cdot u$$
$$\text{der\_d} \cdot V + d \cdot \dot{V} = m_{flow}$$
$$m_{flow} \cdot u + m \cdot \text{der\_u} = H_{flow} + H_r$$

By this reformulation only derivatives for the independent medium variables (and of the volume V) appear and therefore only these variables can be potential states. Note, the equations above are linear in the derivatives and no nonlinear equation system appears anymore.

The first four equations are marked in blue to indicate that these equations are medium specific. *Depending on the medium type*, these four equations need to be provided. This is accomplished by providing a specific BaseProperties Modia model. Example for the SimpleIdealGas medium that has (p,T) as states, where d=d(p,T) and u=u(T):

```
@model SimpleIdealGas_BaseProperties begin
  p_start = 1e5
  T_start = 300.0
  Medium = MediumVariable()
  p = Float(start=p_start)
  h = Float()
  T = Float(start=T_start)
  d = Float()
  u = Float()
  der_d = Float()
  der_u = Float()
  state = MediumState()
@equations begin
  d = d_pT(Medium,p,T)
  u = u_T(Medium,T)
  h = h_T(Medium,T)
  der_d = d_pT_der_2(Medium,p,T)*der(p) +
          d_pT_der_3(Medium,p,T)*der(T)
  der_u = u_T_der_2(Medium,T)*der(T)
  state = setState_pT(Medium,p,T)
  end
```

```
end

BaseProperties(Medium:: SimpleIdealGasMedium;
    p_start=1e5, T_start=300.0) =
    SimpleIdealGas_BaseProperties(Medium=Medium,
                    p_start=p_start, T_start=T_start)
```

Under the assumption that p,T are states and that all used functions are available in ModiaMedia, all the left hand side variables can be computed from p,T and their time derivatives. In the @equations section the Modia convention is used that fc_der_i(..) is the partial derivative of function fc with respect to its i-th argument. Note, the medium specific functions must reflect the true dependency of the function from the independent variables in order that the symbolic transformation does not introduce singularities in the generated code. For example, although p,T are the thermodynamic states of this medium, the inner energy u is only a function of T and not of p,T.

Above, for every medium type a medium specific function BaseProperties(Medium; ...) is defined that selects the *medium specific* BaseProperties @model, instantiates it and returns this instance. Alternatively, all medium-specific BaseProperties @models could be stored in a dictionary, and function BaseProperties could just return an instance of the corresponding @model using the Medium type as a dictionary key. Note, media types that have the same functional dependency on d,u,h, can use the same BaseProperties model. With these pre-requisites a general volume model can be defined as:

```
@model ClosedVolume begin
  Medium = MediumVariable()
  inPlug    = InPlug()
  outPlug  = OutPlug()
  p0 = Medium.infos.p_default
  T0 = Medium.infos.T_default
  medium = BaseProperties(Medium; p_start=p0,
                                   T_start=T0)
  ...
@equations begin
  outPlug.Medium = Medium
  outPlug.Medium = inPlug.Medium
  m = medium.d*V
  U = m*medium.u
  m_flow = inPlug.m_flow + outPlug.m_flow
  H_flow = inPlug.m_flow*inPlug.h +
          outPlug.m_flow*outPlug.h
  H_r = inPlug.m_flow*inPlug.r / medium.d

  # Mass and energy balance
  medium.der_d*V + medium.d*der(V)  = m_flow
  m_flow*medium.u + m*medium.der_u = H_flow+H_r

  # Propagation of specific quantities
  inPlug.p + inPlug.r = medium.p
  outPlug.p = medium.p
  outPlug.r = 0.0
  outPlug.h = medium.h
  end
end
```

A key part of this @model is the declaration of the BaseProperties @model:

```
medium = BaseProperties(Medium; p_start=p0,
                                  T_start=T0)
```

This declaration provides an instance of a medium-specific BaseProperties @model depending on the type of variable Medium. The problem here is that Medium should be propagated through connections and the instantiation of BaseProperties can only be performed when the type of the propagated Medium is known (so instantiation and extended alias elimination must be performed incrementally). Modia does not yet support such a scheme and therefore the current implementation of ModiaFluid requires to define the Medium at volumes to select the BaseProperties @model based on the Medium type.

3.6 Further Issues

ModiaFluid should optionally support bi-directional fluid flow in the future. Additionally, there are other issues:

Caching for media calculations

More complicated media, such as two-phase media or mixture media, may require the solution of nonlinear equation systems whenever medium variables, such as specific internal energy, have to be computed. In Modelica.Fluid typically the nonlinear solver either starts always from the same start values of the iteration variables, or with some very simplified models first start values for the iteration variables are computed. The current version of ModiaFluid only supports the same approach.

In principal it would be possible to make such medium calculations more efficient and more robust by caching the medium states from the previous model evaluation and use them as start values at the next time instant:

```
setState_pT!(state, Medium, p, T)
```

Julia allows to update input arguments and therefore to keep a memory between function calls. The setState_xxx(..) functions would thus be slightly rewritten to update the current state and hereby utilize a cache in the state. In order that Modia knows which variable is computed by such a call (for size inference and equation sorting), the argument that is updated by the call must be "somehow" marked.

Nonlinear equations at junctions

As described in detail in (Franke et al. 2009, section 4.2), junctions may give rise to nonlinear algebraic equation systems where the iteration variables are discontinuous when the mass flow rate changes sign and therefore the solution is hard. In ModiaFluid this cannot occur, because only unidirectional flow is supported where the upstream direction does not change during simulation.

Unnecessary nonlinear equations at 1:1 connections

As described in detail in (Franke et al. 2009, section 4.3), unnecessary nonlinear equation systems occur at every 1:1 connection of fluid components if the *thermodynamic states* are not (p,h) and h is a *nonlinear* function of the *thermodynamic states*. This effect currently appears in ModiaFluid. In Modelica.Fluid this issue is resolved by an inverse function annotation and an involved symbolic manipulation of the equations. Current efforts in ModiaFluid include the pursuit of a simpler solution to this problem.

4 Automatic Differentiation of Media Functions

Partial derivatives of functions are needed since relationships between thermodynamic variables are modelled using functions and these relations needs to be differentiated due to index reduction in the mass and energy balance or for obtaining the Jacobian for iterative solvers. Modelica.Media has many manually provided derivatives of functions. The described approach in ModiaMedia allows automatic differentiation of functions to be easily utilized.

There are several Julia packages for automatic differentiation, see http://www.juliadiff.org/. The partial derivative of a function specificEnthalpy_water(T) is obtained as follows by using the ForwardDiff package:

```
specificEnthalpy_water_der_1(T) =
  ForwardDiff.derivative(specificEnthalpy_water, T)
```

5 Conclusion

Despite past successes of thermo-fluid modeling using the Modelica language, there have been long-standing discussions on how to improve thermodynamic property modeling for dynamic systems by making it more convenient, easier to comprehend, and more powerful. Many of the modern programming constructs of the Julia language, such as multiple dispatch, lend themselves to new approaches to address these existing challenges. The ModiaMedia and ModiaFluid architecture described in this paper represents one experimental effort to leverage these recent developments: Thermodynamic property modeling becomes an order of magnitude simpler, both for implementation and for usage. Furthermore, it becomes then possible to propagate medium models through connection structures and use them in pre- and post-processing. By adopting the new fluid approach from (Zimmer et al. 2018), recent progress within the Modelica community can be directly transferred to Modia. We expect to continue developing and enlarging this thermo-fluid modeling framework to further explore the opportunities afforded by these new computing paradigms and tools.

Acknowledgements

The authors want to thank Jarrett Revels, MIT for valuable advice on Julia and automatic differentiation using Julia.

References

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. (2017): Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. doi: 10.1137/141000671

Brennen, C.E. (2006): An Internet Book on Fluid Dynamics. http://brennen.caltech.edu/FLUIDBOOK/FLUIDBOOK.htm

Elmqvist, H., Tummescheit, H. and Otter, M. (2003): Object-Oriented Modeling of Thermo-Fluid Systems, *Proceedings of the 3rd International Modelica Conference*, Linköping, Sweden, November 3-4, pp. 269-286. https://www.modelica.org/events/Conference2003/papers/h40_Elmqvist_fluid.pdf

Franke R., Casella F., Otter M., Sielemann M., Elmqvist H., Mattsson S.E., Olsson H. (2009): Stream Connectors – An Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena. *Proceedings of the 7th International Modelica Conference*, Como, Italy, Sept. 20-22, pp. 108-121. http://www.ep.liu.se/ecp/043/012/ecp09430078.pdf

Keller, A., et al: https://github.com/ajkeller34/Unitful.jl, downloaded 19 Nov 2018.

Laughman, C.R., Qiao, H. (2016): On the Influence of State Selection on Mass Conservation in Dynamic Vapor Compression Cycle Models. *Mathematical and Computer Modeling of Dynamical Systems*, Vol. 23, No. 3, pp. 262-283, December 2016, DOI: 10.1080/13873954.2017.1298625

Modelica Association (2017): *Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.4,* April 10, 2017. https://www.modelica.org/documents/ModelicaSpec34.pdf

Schade H., Kunz E., Kameier F. and Paschereit C.O. (2013) *Strömungslehre*. 4. Auflage, de Gruyter.

Zimmer D., Bender B., Pollok A. (2018): Robust Modeling of Directed Thermofluid Flows in Complex Networks. *Proceedings of the 2nd Japanese Modelica Conference*, pp. 39-48, Tokyo, May 17-19. https://elib.dlr.de/120701/

Simulative Potential Analysis of Combined Waste Heat Refrigeration using Ammonia in an Intercity Bus on dynamic route

Maximilian Hebeler[1]*; Christian Schulze[2]; Wilhelm Tegethoff[2]; Jürgen Köhler[1]

[1]Institut für Thermodynamik, TU Braunschweig, Germany,
[2]TLK Thermo GmbH, Germany
** Corresponding author. Phone: +49 531-391-7892; E-Mail: m.hebeler@tu-bs.de*

Abstract

In this work, a simulative potential analysis of a possible topology for combining waste heat recovery and passenger compartment refrigeration using ammonia is carried out. The focus is on the energetic assessment using a detailed simulation model of a long haul intercity bus.

The topology combines a conventional refrigeration cycle with an Organic Rankine Cycle (ORC). Both systems share the working fluid and the condenser. The used refrigerant is Ammonia (R-717). Expansion machine and compressor are both connected to the drive belt of the vehicle. In order to evaluate the fuel consumption reduction potential of that topology the intercity bus simulation model, equipped with a CO2 (R-744)-refrigeration system, is used as a reference.

The results show that using an Organic Rankine Vapor Compression Cycle (ORVC) equipped with ammonia leads to an effective reduction of fuel consumption for a long-haul journey. The ORVC topology reduces fuel consumption by 7.9 %.

Keywords: ORC, ORVC, CO2, Ammonia, R-744, R-717

1 Introduction

The compressor of the Air-Conditioning System (AC) of an intercity bus uses up to 15 kW of additional mechanical power from the engine, thereby reducing the effective power available for vehicle traction. For a long haul journey of several hundred kilometers, this energy input accounts for around 8 % of the overall diesel fuel consumption. On the other hand, approximately one third of the supplied chemical energy is rejected as hot exhaust gas into the environment and another third is dissipated by the cooling system leaving only one third for vehicle traction. In order to use the exergetic potential of the exhaust gas, a Waste Heat Recovery (WHR) system can be applied, such as an Organic Rankine Cycle (ORC). The recovered energy can be used for reducing the engine load mechanically or by driving auxiliary loads like the alternator or the refrigerant compressor. Therefore, combining the waste heat recovery and the air-conditioning system can be a promising method for reducing primary energy usage. Combining these two systems efficiently is a challenge, as many aspects and interactions have to be considered.

The work described in this paper is part of the overall research efforts that aim to develop and compare different topologies and methods for intercity bus climatization using exhaust waste energy. The aim is to completely provide the energy need of the refrigeration system and all its components via the WHR system taking environmental regulations into account. In this work, a preliminary analysis of one possible topology for combined waste heat refrigeration is carried out using a detailed simulation model of a long haul intercity bus equipped with an R-744-refrigeration system. In terms of ORVC, the carried out research is mainly distinguished to other researches (e.g. Yilmaz, 2015; Wang *et al*, 2011; Saleh, 2016) by using ammonia as working fluid/refrigerant.

2 Overview of Organic Rankine Vapor Compression Cycle Systems

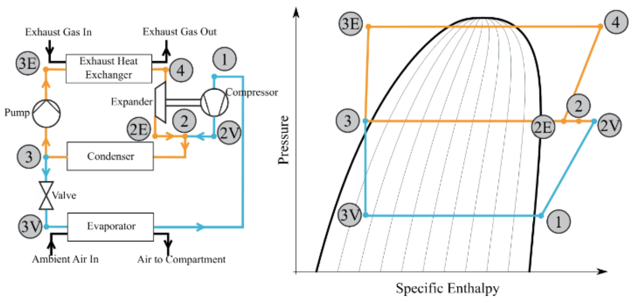


Figure 1. Schematic overview of an Organic Rankine Vapor Compression Cycle. On the left hand side, the diagram shows all essential components, the right hand side shows the corresponding thermodynamic points of state for the working fluid ammonia.

Figure 1 shows an example configuration of an Organic Rankine Vapor Compression Cycle (ORVC), as commonly shown in the literature (e.g. Yilmaz, 2015; Wang *et al*, 2011; Saleh, 2016). All relevant components and thermodynamic points of state are depicted. The conventional refrigeration process, consisting of evaporator, condenser, compressor and expansion valve, is extended by a Rankine Cycle, that receives heat from the engine exhaust stream and converts that thermal energy into mechanical energy by evaporation and expanding the fluid. Expansion machine and refrigereant compressor are mechanically coupled thus the presented process is a combination of a heat engine and refrigeration machine, sharing the same working fluid. The remaining heat of both cycles is rejected via the same condenser on middle pressure level. Depending on the system design, the expansion machine or the compressor can be connected additionally to an auxiliary load or drive. The net power output of the process is defined as follows:

$$P_{Pump} + P_{Expander} + P_{Compressor} + P_{Fan} = P_{Net} \quad (1)$$

3 Simulation Model and Investigated Topology

In the following, the investigated topology and its implementation into the omnibus as well as the simulation statistics of the corresponding models are described. All models have been created using the TIL and TIL Media library (Richter, 2008; Gräber *et al*, 2010; Schulze *et al*, 2011) in Modelica. In terms of the TIL library a highly dynamic modelling approach has been used. All components can handle zero mass flow rate and flow reversal.

3.1 Vehicle simulation model

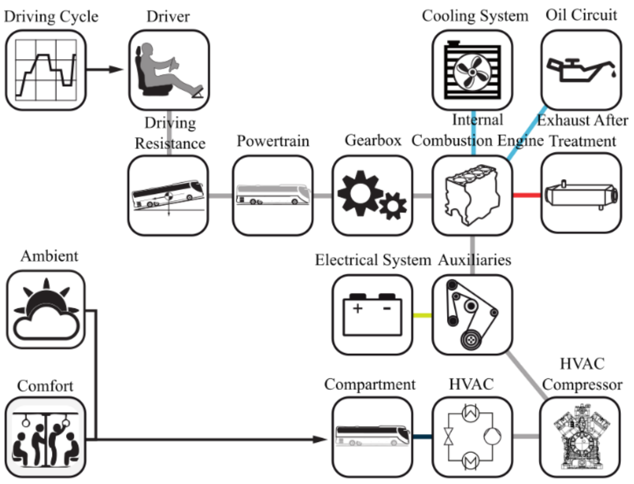


Figure 2. Schematic depiction of the main subsystem interactions of the omnibus as modeled in (Kaiser, 2018). Depiction in dependence on (Ebeling, 2018).

Figure 2 gives an overview of the complete omnibus simulation model and its sub-models. All longitudinal dynamics have been considered. Vertical dynamics have been neglected. The model was created by Kaiser in (Kaiser, 2018) using the object orientated programming language Modelica and has been validated with experimental data.

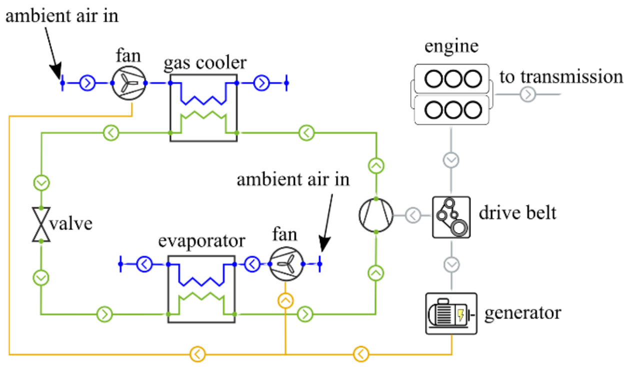


Figure 3. Schematic of the R-744 reference refrigeration system including connections to engine, drive belt and generator. Blue represents air, green represents working fluid and yellow represents electrical current. Mechanical work is depicted in gray.

The actual vehicle represents an omnibus with a passenger capacity of 48 people and an engine peak power of 295 kW. Figure 3 shows the R-744-refrigeration system and its connections into the powertrain of the omnibus. The refrigeration system has three evaporators in parallel, one for the climatization of the driver and two for the passenger compartment. For the sake of simplification, Figure 3 shows a simplified version of the refrigeration cycle with only one evaporator for the passenger compartment. The compressor is driven by the drive belt of the combustion engine. All fans are fed by the vehicle's electrical system, respectively the alternator. Further explanation of the refrigeration system can be found in detail in (Kaiser, 2018).

Table 1 summarizes some of the simulation system's statistics for the vehicle simulation model including HVAC-unit.

Table 1. Some of the simulation statistics of the vehicle simulation model (including HVAC-unit)

Nontrivial Equations	21189
Continuous time states	1024
Time-varying variables	31845
Number of mixed real/discrete systems of equations	35
Highest number of non-linear equations after manipulation	3
Number of systems of equations with size of 3	2
Number of systems of equations with size of 2	0
Number of systems of equations with size of 1	51

3.2 Organic Rankine Vapor Compression simulation model

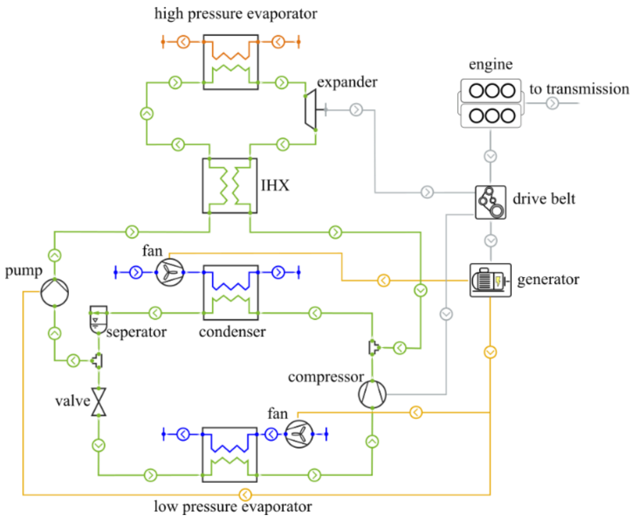


Figure 4. Schematic of the ORVC-topology including connections to engine, drive belt and alternator. Orange represents exhaust gas, blue represents air, green represents working fluid and yellow represents electrical current. Mechanical work is depicted in gray.

Figure 4 shows the ORVC-topology and its connections into the powertrain of the omnibus. This topology is a combination of an R-134a refrigeration system and an ordinary ORC-topology. The condenser of the refrigeration system replaces the ORC-condenser. The expected condensation heat is approximately twice the condensation heat of the original refrigeration system, hence the size of the condenser is doubled. The number of condenser fans is increased as well. The expansion machine and the compressor are both coupled to the drive belt. The expansion machine is assumed to have a constant overall efficiency of 70 %. Finally, the gear ration between compressor and drive belt is changed in order to achieve the same cooling power as in the original R-134a system. In terms of working fluid ammonia (R-717) is chosen as a drop-in, since it is a natural refrigerant (GWP and ODP = 0) with a high volumetric cooling capacity and a good compromise in usability as a working fluid in terms of waste heat recovery. The fundamental equation of state of ammonia was implemented as a multi parameter equation of state as in (Tillner-Roth *et al*,1993; Span *et al*, 1996), therefore there was no interpolation routine necessary. It has to be pointed out, that in this context it has not been taken into account that the use of R-717 in direct evaporation systems can lead to grave safety issues, since it is a highly toxic substance. This study is supposed to be a simulative potential study of ammonia as a working fluid in ORVC systems with respect to highly transient boundary conditions.

For controlling the pump speed, a PI-Controller is used, which sets the outlet state of the expansion machine to 60 K superheated vapor. In that manner, the following internal heat exchanger is provided with enough temperature difference to transfer heat from mid pressure to high pressure level. Table 2 summarizes the control concept of the presented ORVC topology.

Table 2. PI-Controller assignment for the presented ORVC topology

Actuating Variable	**Controlled Variable**
pump speed	overheating at expander outlet
condenser fan speed	mid-pressure level
evaporator fan speed	air inlet and compartment temperature
valve eff. flow area	overheating at evaporator outlet

The described topology is similar to (Yilmaz, 2015), where R-134a and R-245fa are investigated as working fluid. In (Yilmaz, 2015), compressor and expander are directly coupled with no other external connection. However, in this work the expansion machine and the compressor are coupled via the drive belt of the vehicle. Hence, the shaft work of expander and compressor may be unequal, so that power is drawn from or supplied to the engine. Furthermore, as mentioned, in this work ammonia is used as working fluid.

Table 3 summarizes some of the simulation system's statistics for the vehicle simulation model including the ORVC-topology. Despite the system size, the presented model offers an integration time close to real-time on a standard Desktop-PC. In comparison to the reference model (see. Table 1), the number of nontrivial equations has increased. Still, the number of nonlinear systems of equations remains the same.

Table 3. Some of the simulation statistics of the vehicle simulation model (including ORVC-topology)

Nontrivial Equations	22992
Continuous time states	1053
Time-varying variables	36176
Number of mixed real/discrete systems of equations	35
Highest number of non-linear equations after manipulation	3
Number of systems of equations with size of 3	2
Number of systems of equations with size of 2	0
Number of systems of equations with size of 1	51
CPU-Time for integration of 37622 s of simulation time	3.38e4 s

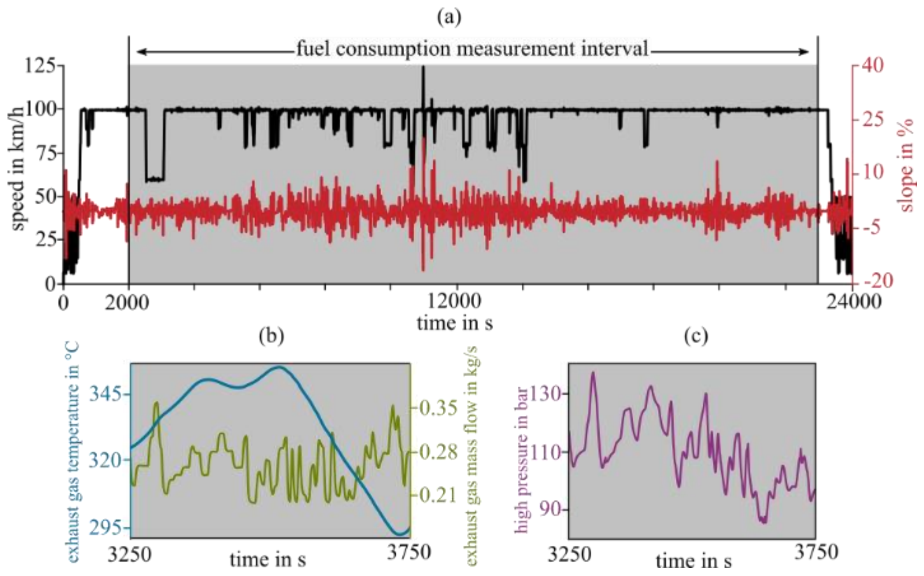


Figure 5. Real life driving scenario from Hanover to Munich (Kaiser, 2018). Vehicle speed and slope of route depicted with respect to time (a). System input boundary conditions (b) and system response of pressure at high pressure evaporator outlet (c) for a representative time interval of 3250 s to 3750 s.

4 Boundary Conditions and Simulation Results

In order to evaluate the integration of the presented topology into the vehicle, a real driving scenario from Hanover to Munich is applied as simulation input. The scenario considers vehicle speed, slope of route and weather conditions. For the applied scenario a typical august summer day has been chosen, further details are explained in (Kaiser, 2018). Figure 5 shows the vehicle speed over time and the corresponding driving slope. In case of the ORVC-topology the system is only activated during highway conditions. As a consequence the fuel consumption measurement interval has been chosen from 2000 until 23000 seconds of the journey. The mentioned interval is depicted in Figure 5, as well. The results are compared to the reference only in that time period. In terms of the ORVC configuration, the exhaust gas evaporator is integrated after the SCR-catalyst of the exhaust after treatment system of the vehicle.

In all simulations the Dassl's integration method has been used with a solver tolerance of 1e-5. The output interval has been set to 1s.

4.1 Evaluation Parameters

The exergetic efficiency is defined as the ratio of the net power of the process and the provided exhaust gas exergy at the exhaust gas heat exchanger inlet:

$$\eta_{ex} = \frac{-P_{Net}}{\dot{Ex}_{inlet,Gas}} \qquad (2)$$

However, for positive values of the net power of the process (the process draws power), the exergetic efficiency is negative. In the shown results this is usually the case, since the drawn power of the refrigeration system exceeds the provided power of the waste heat recovery system. It is therefore more feasible to introduce the so called Work Number (WN), which gives the ratio of the cooling heat to the necessary work input of the whole process:

$$WN = \frac{\int \dot{Q}_{Evaporator}\ dt}{\int P_{Net}\ dt} \qquad (3)$$

4.2 Results

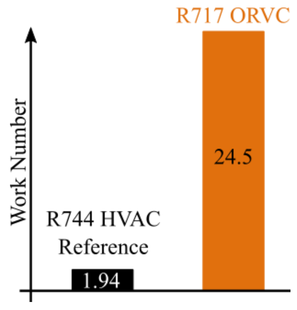


Figure 6. Overview of the Work Number for the evaluated topologies for the period of a real life driving cycle from Hanover to Munich between 2000 and 23000 seconds of the journey.

Figure 7 shows the result for the exergy analysis in the mentioned driving scenario for the reference system and the mentioned topology. Exergy source and exergy demand are compared to each other. The exergy balance is formed around the engine, the drive belt and the gear-box, taking all necessary consumers into account. The exergy demand is hereby divided into driving resistance and gearbox losses, engine auxiliaries (all kinds of consumers like pumps or fans etc.), vehicle auxiliaries (lighting and electronic control units etc.), HVAC auxiliaries (condenser and evaporator fans and control units), compressor shaft power and the resulting drive belt losses. Electrical conversion losses of the electrical components are included as exergy demand of the corresponding component. In case of ORVC applications the exergy demand of the pump is depicted as well. As exergy source the corresponding component is depicted in opposition and is divided into engine crankshaft and, if existing, expansion machine.

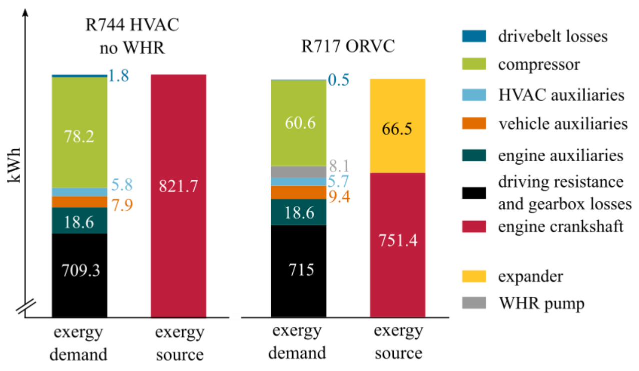


Figure 7. Exergy demand and source for the evaluated topologies for the period of a real life driving cycle from Hanover to Munich between 2000 and 23000 seconds of the journey.

It can be seen that the exergy demand covered by the engine decreases from the reference system without WHR to the ORVC topology. The exergy output of the engine decreases by 8.6 % compared to the reference. This decrease corresponds to a fuel consumption decrease by 7.9 %. The reason for this decrease is mainly due to the WHR system, since the expansion machine reduces the engine load, as mentioned above. In addition to that, the change of the refrigerant from R-744 to R-717 already has a notable influence. The necessary work input for the compressor is reduced by almost 23 %, which already improves fuel efficiency. The mentioned change in condenser size and the increase in the number of condenser fans do not seem to have a noticeable effect on the HVAC auxiliaries' exergy demand. Figure 6 shows the work number of the simulated topology in comparison to the system without WHR taking all work in and outputs into account. It can clearly be stated, that in case of the ORVC topology the necessary work input into the system is twelve times lower than in case of the reference system with no WHR.

Still, the impact of the expansion machine on the drive train of the engine has to be taken into account. As mentioned, in both topologies identical boundary conditions were applied. Due to that, the exergy demand for the driving resistance should be equal in all simulations. As shown in Figure 7, this is not the case. The driving resistance differs by about 6 kWh in case of the ORVC topology compared to the reference. Because of the thermal capacity of the WHR and the exhaust manifold, the time constant of the WHR is several magnitudes bigger than that of the drivetrain's mechanics, hence the expansion machine is still providing energy while the driver is applying the brake pedal. Due to the additional power input of the expansion machine into the drive train, the driver, represented by a PI-controller, tends to accelerate and decelerate more aggressively, which leads to the shown increase in driving resistance, respectively fuel consumption. Consequently, the fuel reduction potential for the shown ORVC configuration is slightly higher than depicted in Figure 6 and Figure 7 respectively. The adaption of the driver model as function of drivetrain design is part of future work of the author.

5 Summary & Discussion

In this work, a simulative preliminary comparison of a possible topology for combined waste heat refrigeration has been carried out using a detailed simulation model of a long haul intercity bus. The shown topology implements an Organic Rankine Vapor Compression Cycle into the vehicle, where the refrigeration system and the waste heat recovery system share the refrigerant and condenser. The refrigeration side of the shown topology is derived from an R-134a refrigeration system. The applied refrigerant is Ammonia (R-717). Expansion machine and compressor are both connected to the drive belt of the vehicle. As study reference, the intercity bus model equipped with an R-744-refrigeration system is used.

The simulation results show that with the applied topology and parameters it is possible to effectively decrease the fuel consumption of an intercity bus on a long haul journey under dynamic boundary conditions. Furthermore it has been shown, that the dynamic modelling approach enables the transient simulation of large processes in a time-efficient manner without sacrificing accuracy in physical behavior.

Still, it has to be stressed, that in this comparison it has not been taken into account that the use of R-717 in direct evaporation systems can lead to grave safety issues, since it is a highly toxic substance. The scope of future work is therefore to implement an ORVC in combination with an intermediate evaporation system. It has to be evaluated, if yet a fuel reduction potential is given. In that case the use of an ORVC with R-717 is highly promising, since the needed cooling load can be maintained with simultaneously meeting environmental regulations, such as GWP and ODP.

Apart from that, it has to be pointed out, that further extended research has to be carried out in order to compare the two systems. Several effects have not been analyzed, which could improve or limit the presented ORVC performance. To highlight all these effects goes beyond the scope of this work and will be presented in future publications.

Acknowledgements

Parts of this publication have been developed with funding from the German Federal Ministry of Education and Research (BMBF) within the research project *VEOS – Verfahren zur energetischen Optimierung dynamischer thermischer Systeme (KMU Innovativ, 01 | LY1502) and VEOTOP – Verfahren zur optimalen Synthese und Topologieoptimierung komplexer thermischer Energiesysteme (KMU Innovativ, 01 | LY1809B)*

References

Yilmaz, A.: Transcritical organic Rankine vapor compression refrigeration system for intercity bus air-conditioning using engine exhaust heat. Energy 82 (2015) 1047-1056.

Saleh, B.: Parametric and working fluid analysis of a combined organic Rankine-vapor compression refrigeration system activated by low-grade thermal energy. Journal of Advanced Research (2016) 7, 651–660

Kaiser, C.: Untersuchungen zur Effizienz- und Leistungsverbesserung von Omnibusklimaanlagen. PhD thesis, TU Braunschweig (approx. published in 2018)

Richter, C.: Proposal of New Object-Oriented Equation-Based Model Libraries for Thermo-dynamic Systems. PhD thesis, TU Braunschweig (2008)

Schulze, C., Gräber, M., Huhn, M., and Grätz, U.: Real-Time Simulation of Vapour Compression Cycles. In: Proceedings of the 8th International Modelica Conference, Dresden (2011)

Wang, H., Peterson, R., Herron, T.: Design study of configurations on system COP for a combined ORC (organic Rankine cycle) and VCC (vapor compression cycle). Energy 36 (2011) 4809-4820.

Gräber, M., Kosowski, K., Richter, C., and Tegethoff, W.: Modelling of heat pumps with an object-oriented model library for thermodynamic systems. Mathematical and Computer Modelling of Dynamical Systems, 16(3):195–209., (2010)

Ebeling, P.: Konzeption eines Rankine-Prozesses für den transienten Betrieb im Omnibus, PhD thesis, TU Braunschweig (approx. published in 2018)

Tillner-Roth, R., Harms-Watzenberg, F., and Baehr, H.D., In: Eine neue Fundamentalgleichung fuer Ammoniak, DKV-Tagungsbericht, 20:167-181, 1993.

Span, R. and Wagner, W.: A New Equation of State for Carbon Dioxide Covering the Fluid Region from the Triple-Point Temperature to 1100 K at Pressures up to 800 MPa, In: J. Phys. Chem. Ref. Data, 25(6):1509-1596, 1996.",

SESSION 5D: ELECTRICAL POWER 2

Modeling of PMU-Based Automatic Re-synchronization Controls for DER Generators in Power Distribution Networks using Modelica and the OpenIPSL

Biswarup Mukherjee[1] Luigi Vanfretti[2]

[1]Indian Institute of Technology Bombay, India, bismuk.ece@gmail.com
[2]Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute (RPI), USA, vanfrl@rpi.edu

Abstract

Re-synchronization is traditionally coordinated between the electric power transmission network operators and power plants in an isolated portion of the grid. As the number of DER continues to increase with the rise of renewable energy sources located at the lower voltage networks, automatic re-synchronization methods that can be applied to a great number of DER are desirable. This paper describes the architecture and modeling of an automatic re-synchronization controller, which can be applied to synchronize an islanded portion of the grid by using remote measurements to drive a Distributed Energy Resource (DER) within the islanded network. The controller's re-synchronization function uses bus frequency measurements, which are derived using bus voltage phasors and a new bus frequency computation technique that can be used during the execution of dynamic simulations. This paper also introduces a new bus-angle difference control function within the re-synchronization control system, which allows monitoring the phase angle difference between two buses so to avoid unwanted re-synchronization. The effect of the angle difference control function is evaluated using a controlled circuit breaker considering different power dispatch levels of the generator in the distribution network model. Both deterministic and stochastic load models are used to analyze the performance of the automatic re-synchronization control system.

Keywords: Automatic re-synchronization controller; phase angle difference controller; power distribution network; synchrophasors; Modelica; OpenIPSL

1 Introduction

1.1 Motivation

Re-synchronization needs to be coordinated between the transmission operators and power plants in an isolated portion of the grid in order to maintain the balance between the power supply and demand. This task can be challenging when one portion of the distribution grid contains small generators having low inertia which is the case of Distributed Energy Resources (DER), such as small hydro, wind and solar power plants. As the number of DER continues to increase with the rise of renewable energy sources located at the lower voltage networks, automatic re-synchronization method that can be applied to a great number of DER are desirable. One of the main challenges with conventional synchronization techniques is to maintain a stable system operation when a disturbance occurs, during the re-synchronization process (Belyaev et al., 2015).

In addition, as reported in (Assis and Taranto, 2013), that perfect re-synchronization can only be achieved when there is no power flow through the coupling circuit breaker at the time of operation. Improper re-synchronization leads to poor power quality and reliability, diminishing energy economics (Mazloomzadeh et al., 2012).

To address these challenges, this paper presents the modeling of an automatic re-synchronization control system in a centralized control architecture that allows to reconnect two isolated power networks by exploiting synchrophasors and frequency estimates from Phasor Measurement Units (PMUs). PMUs are placed in both transmission and distribution networks to assist the automatic re-synchronization controller. All modeling has been carried out using Modelica language (Fritzson, 2004) and the OpenIPSL (Vanfretti et al., 2016) library.

1.2 Literature Review

Several aspects for grid synchronization have been discussed in (Blaabjerg et al., 2006) (Timorabadi and Dawson, 2006). It is described in (Belyaev et al., 2015) that improper re-synchronization may lead to damaging circuit breaker contacts, mechanical stresses on a generator and prime mover, and other unwanted wear/tear.

The previously cited searches also highlight that, there will be a sudden active power flow at the time of re-synchronization, until the initial frequency difference of the two sides of the circuit breaker reach a stable common frequency. The other criterion is the magnitude of the voltage differences across the circuit breaker. If this magnitude is very high, it will cause sudden transient current to flow through the circuit breaker, resulting in high reactive power flow. Meanwhile it is reported in (Belyaev et al., 2015) that phase angle difference is another important criterion for re-synchronization. Therefore control of the phase angle difference across the circuit breaker can be beneficial for re-synchronization.

To automate the re-synchronization process, it is proposed in (Assis and Taranto, 2013) to use a voltage and speed control strategy for automatic re-connection using remote sensing of voltage and frequency signals after an intentional islanding occurs. However, this approach still depends on conventional synchronism check relays. Auto-synchronizers are also available to perform automatic re-synchronization when the synchronization function is active. They will automatically adjust the speed of the generator through the governor in order to match the bus frequency. This arrangement is ideal for synchronizing an islanded system of multiple generators and load to the grid or another islanded network (Thompson, 2012). Alternatively, the use of frequency estimates from PMUs are proposed in (Kirkham et al., 2014). Reference (Almas and Vanfretti, 2016) proposes to use PMUs to obtain frequency measurements and GOOSE messaging for control, however, the control actions are discretized and applied using a look-up table, which can introduce undesirable perturbation to the generator.

Alternatively, to perform re-synchronization operation, it would be equally interesting to develop a controller capable of using the frequency estimates from PMUs while applying smoother control actions. Therefore this paper proposes to use PMU measurements from both transmission or distribution networks for the re-synchronization control system, while applying smooth control actions on error signals from voltage, frequency and angle differences.

1.3 Paper Contributions

The main contributions of this paper are as follows:

- A bus-angle difference-based controller that helps reducing the bus voltage angle difference and enhances operation of the re-synchronization process. The controller uses the unwrapped angle calculated from the bus angles, which is required due to the angle switching.
- The proposed control system is interfaced in cascade with traditional turbine governor and excitation controllers (i.e. governor and voltage regulators) supplementing generator control systems model instead of replacing existing ones.
- The performance of the proposed control system is evaluated for both stochastic and deterministic load models to illustrate the impact of load uncertainties in the short time-scale of the re-synchronization process.

The remainder of this paper is organized as follows. In Sections 2 the centralized control system architecture and modeling of the automatic re-synchronization controller is explained. Section 3 describes the power system and simulation execution models. Finally, case studies are analyzed in Section 4 and conclusions are drawn in Section 5.

2 The Re-synchronization Controller

2.1 Control Architecture

A schematic of the proposed control architecture for automatic re-synchronization is shown in Figure 1. It contains three major functions or units: computation unit, activation unit and control unit. The computation unit computes the bus voltage, frequency and angle differences. The control system is comprised of a voltage controller, a frequency controller and an angle-difference controller. In addition the 'activation unit' is used to trigger the individual regulators following different control modes.

The decisions taken by this unit require checking thresholds for three different synchronization variables: ΔV, ΔF, $\Delta\theta$; these are the voltage, frequency and angle differences, respectively, obtained from the computation unit. The synchronizing variables are calculated using PMU measurements from two locations of the power network; one is the main grid and one in the islanded grid under control.

One of the control modes of the activation unit will be analyzed herein the traditional sequential mode. This mode activates each of the individual controllers in sequence, after each of the individual thresholds for the synchronization variables have been reached. The order is ΔV, ΔF and finally $\Delta\theta$. Once the thresholds set for the synchronization variables have been reached, the re-synchronization process is completed by sending a trip signal to the receiving end of the controlled circuit breaker. Throughout this paper it is assumed that the sending end circuit breaker is closed and that the power line is energized. Switching transients (i.e. those involved in the electro-magnetic behaviour of the system) are ignored, as it is assumed that because the control system minimizes errors in the synchronizing variables, the switching transients will be negligible.

2.2 The ΔV, ΔF and $\Delta\theta$ controller

Similarly to (Assis and Taranto, 2013), both the voltage controller and the frequency controller use a PI block. In addition an angle difference controller, proposed here, also requires a PID function. The schematic of voltage controller and the frequency controller are shown in Figures 2 and 4 respectively while their Modelica implementations are shown in Figures 3 and 5 respectively.

Table 1. Truth table of the voltage controller

Boolean Input signal	Output (y)
True	$y = \Delta V \left(K_P + \frac{K_I}{S} \right)$
False	0

In the Modelica implementation of the voltage controller (see Figure 3) `u1` and `u2` represent the voltage magnitudes of the transmission and distribution network buses, respectively. The output of the voltage controller is

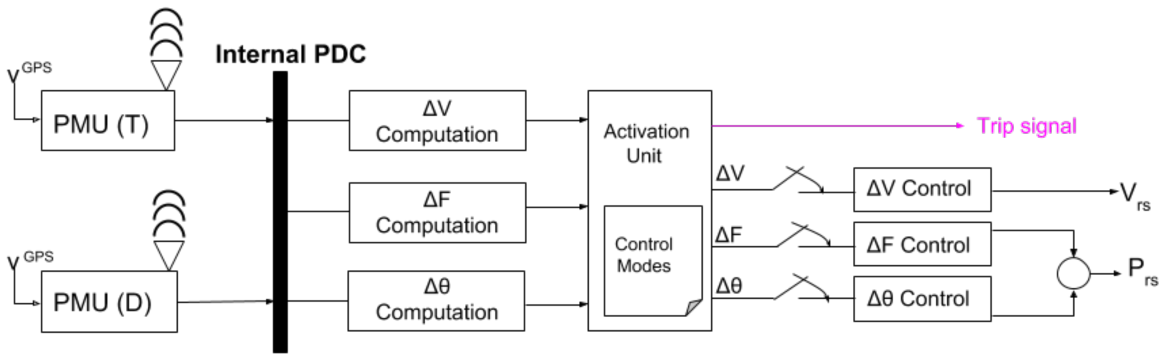


Figure 1. Architecture of the automatic re-synchronization controller.

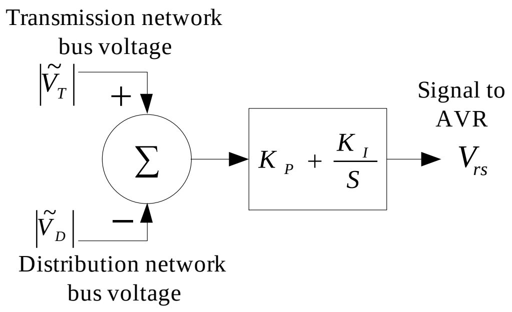


Figure 2. Schematic of voltage controller.

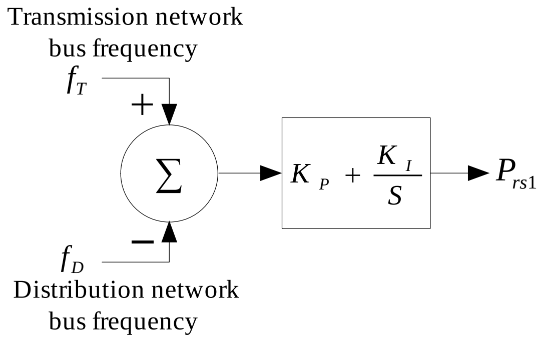

Transmission network bus frequency
f_T
+
Σ
$K_P + \frac{K_I}{S}$
P_{rs1}
−
f_D
Distribution network bus frequency

Figure 4. Schematic of frequency controller.

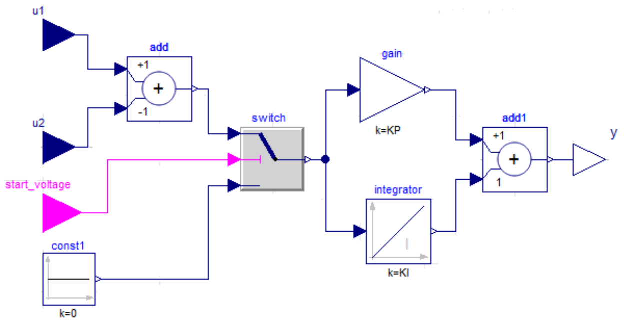


Figure 3. Implementation of the voltage controller in Modelica.

Table 2. Truth table of the frequency controller

Boolean Input signal	Output (y)
True	$y = \Delta f \left(K_P + \frac{K_I}{S} \right)$
False	0

applied to the AVR error signal, which controls the field voltage of the generator.
If the Boolean input signal, `start_voltage`, is applied to the switch is true then the computed voltage difference is fed to the PI block. If this Boolean signal is false, the output of the controller becomes zero. In sum, the voltage controller output is defined by the truth table shown in Table 1.

In the Modelica implementation of the frequency controller (see Figure 5) `f_IB` and `f_DN` represent the frequencies of the transmission and distribution networks, respectively. The output of the frequency controller is `Prs1` as shown in Figure 5.

If the Boolean output signal from the XOR gate is true, the output of the switch is the frequency difference, which is applied to the input of the PI controller. If this Boolean output of the XOR gate is false then the output of the switch is zero, and as a result, the output of the PI controller becomes zero. The output of the XOR gate becomes true when either of the Boolean inputs is true, if booth the inputs are true then output is false. The Boolean signal `Trigger` turns true when the voltage limit is satisfied. The Boolean signal `Block` is true when all three (ΔV, ΔF and $\Delta\theta$) limits are checked successfully. Therefore the output of the frequency controller can be represented as shown in Table 2.

The angle difference controller is a PID controller whose input is the angle difference between the transmission and distribution side bus voltage angles. A simple

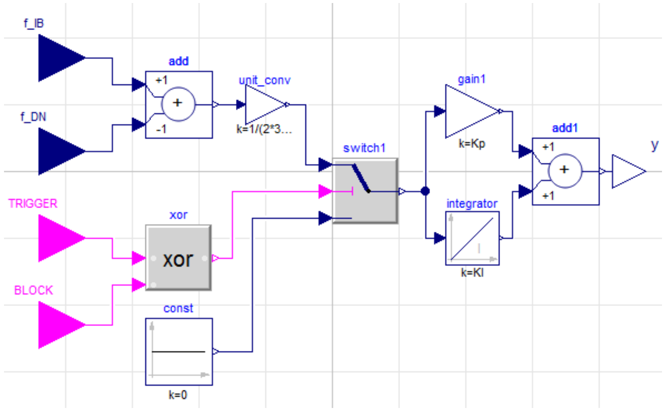


Figure 5. Implementation of the frequency controller in Modelica.

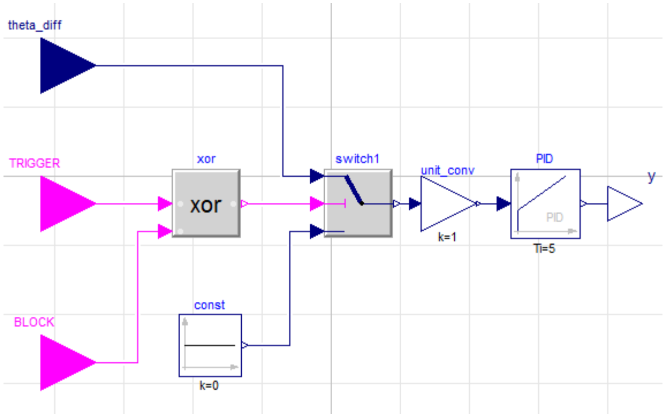


Figure 7. Implementation of the angle difference controller in Modelica.

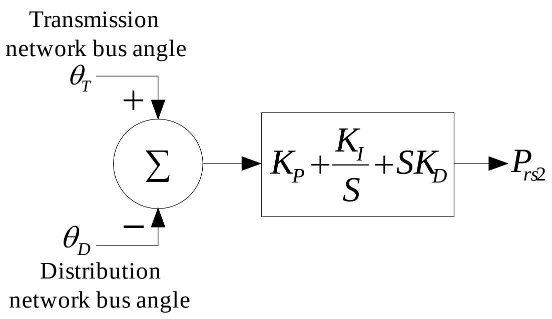

Transmission network bus angle
θ_T
+
Σ
−
θ_D
Distribution network bus angle
$K_P + \frac{K_I}{S} + SK_D$
P_{rs2}

Figure 6. Schematic of $\Delta\theta$ controller.

block diagram representation of this controller is shown in Figure 6, while its Modelica implementation is presented in Figure 7. The output of the angle difference controller is described in Table 3.

Table 3. Truth table of the $\Delta\theta$ controller

Boolean Input signal	Output (y)
True	$y = \Delta\theta \left(K_P + \frac{K_I}{S} + K_D S \right)$
False	0

PMUs perform angle wrapping when reporting phasor data, if used in such representation during simulation, computations will face numerical issues (Milano and Ortega, 2017). Hence to avoid this difficulty the following Modelica code was implemented to unwrap the wrapped bus voltage angle at each bus, from where the angle difference is calculated. The angle controller uses the new angle input from `theta_diff_new`, so that the angle difference does not corrupt the performance of the proposed controller. The calculation of `theta_diff_new` uses the Modelica operator `Homotopy`[1] that operates on the actual bus angle difference due to its non-linearity.

[1]Online at: http://modelica.readthedocs.io/en/latest/operators.html

```
equation
theta_diff = (-B6.angle) + B4.angle;
theta_diff_new = homotopy(actual =
    smooth(0, noEvent(if theta_diff >
     180 then theta_diff - 360 else
    if theta_diff < (-180) then
    theta_diff + 360 else theta_diff)
    ), simplified = theta_diff);
connect(theta_diff_new,
    G22.theta_diff);
end;
```

2.3 Centralized control system architecture and Modelica implementation

The Modelica implementation of Figure 1 is presented in Figure 8. The controller is modeled using a centralized control architecture deployed in the generator at the distribution network. The `TriggeredSampler` blocks are used inside the re-synchronization unit. These blocks are used from the Modelica Standard Library [2] and latch the input when the Boolean trigger input signal is true. The Boolean output signal from the re-synchronization unit is applied to the circuit breaker when all three limits are satisfied inside the activation unit. The Modelica implementation of the generator at the distribution network is presented in the Figure 11.

2.4 Modeling of Frequency computation Block

In previous work (Mukherjee and Vanfretti, 2018), the authors developed a technique to compute frequency estimates during the execution of dynamic simulation. Simulation results comparing this model with the conventional frequency computation approach used in power systems. This previous results are used in this work.

To briefly summarize this previous work, let ω be frequency of the bus voltage, the first order derivative of the bus angle represents the frequency deviation at the bus.

[2]Online at: https://github.com/modelica/ModelicaStandardLibrary

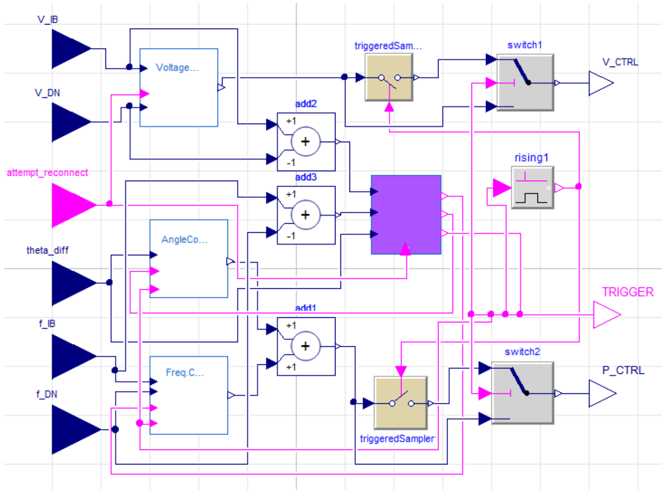


Figure 8. Implementation of the automatic re-synchronization controller in Modelica.

The bus frequency can be determined from the following equation, where V_r and V_i represent the real and imaginary parts of complex bus voltage respectively:

$$\omega = \frac{V_r \dot{V}_i + V_i \dot{V}_r}{V_i^2 + V_r^2} \tag{1}$$

The controller uses the equation above to compute the bus frequency from the bus voltage data, which is implemented as a Modelica code in the frequency computation blocks inside the simulation set-up that will be described in the next sections.

3 Power System Model and Simulation Set-up Implementation

3.1 Power System Model Implementation

Figure 9 shows the power system model mapped on the component layer of Smart Grid Architecture Model (Hooshyar and Vanfretti, 2017), while it's Modelica implementation presented in Figure 10. Figure 9 is useful to understand re-synchronization would require the coordination between three domains - transmission, distribution, and DER owners, and thus, the proposed re-synchronization controller would be beneficial. The figure shows how PMU data is measured at both a transmission and distribution substation, while the re-synchronization controller is located at the DER substation.

In the Modelica model shown in Figure 10, the re-synchronization controller is within the generator (G22). This is expanded in Figure 11, that shows how G22 is modeled using the GENSAL block in OpenIPSL that corresponds to the synchronous generator; IEEESGO is the OpenIPSL block that corresponds to the gas and turbine model, and SEXS is the OpenIPSL block used to model the excitation control system of the generator. Meanwhile, in Figure 10, G1 represents a large power plant connecting to the transmission portion. This is expanded in Figure 12, that shows how G1 is modeled using the HYGOV block from OpenIPSL that corresponds to a hydraulic turbine and governor systems model (HYGOV). To model hydraulic power plants, the synchronous generator GENSAL block in OpenIPSL is used to consider salient fluxes, and excitation control system of the generator (SEXS). The upper left corner of Figure 12 shows how the overall system frequency is varied by introducing a speed change in the governor system of the transmission network generator model, before the re-synchronization process starts.

3.2 Simulation Set-Up Implementation

The Modelica implementation of the simulation set-up block is presented in Figure 13. It is used to set-up the re-synchronization event that leads to the activation of the automatic re-synchronization controller. The circuit breaker 2 remains always closed and keeps line (L3) energized from the transmission side. Circuit breaker 1 is controlled with the following logic. Initially, for first 6 seconds, the breaker is closed and after that it is open. At 6.01 seconds the re-synchronization process starts and the Boolean signal `y3` becomes true. This output is applied as the Boolean input to the automatic re-synchronization unit to start the voltage control. When the Boolean output from the activating unit becomes true, the breaker CB2 is closed. The frequency computation blocks calculate the frequencies of each side of the network. The outputs `y1` and `y2` from these frequency calculation blocks are applied to controller inside G22, while a Boolean constant true signal keeps the circuit breaker CB1 closed to keep the transmission line energized between both breakers.

4 Case Studies

The following case studies are performed using a steam turbine and governor system in the distribution side generator model G22 to analyze the performance of the proposed automatic re-synchronization controller.

4.1 Sequential Control Mode Performance

This case study includes the performance of the re-synchronization controller shown in Figures 14-16. As it can be observed from Figure 14a, soon as the re-synchronization process starts at 6.01 seconds the voltage controller starts working and effectively reduces the oscillations in bus voltages (Bus 4 and Bus 6 voltage). After the re-synchronization occurs at 150 seconds the voltage controller still works minimizing the error in the voltage difference for both the buses (Bus 4 and Bus 6).

From Figure 15a it can be seen that both the transmission and distribution network frequency deviation undergo excursion of -0.5≤ Δf_T≤-0.5 and -0.4≤ Δf_D≤-0.05. The frequency controller in this paper aims to reduce the frequency deviation to zero and that can be seen from Figure 14b; after the network is re-connected with the transmission side, frequency deviation (with respect to that of the transmission network) remains zero which satisfies the goals of the proposed automatic re-synchronization con-

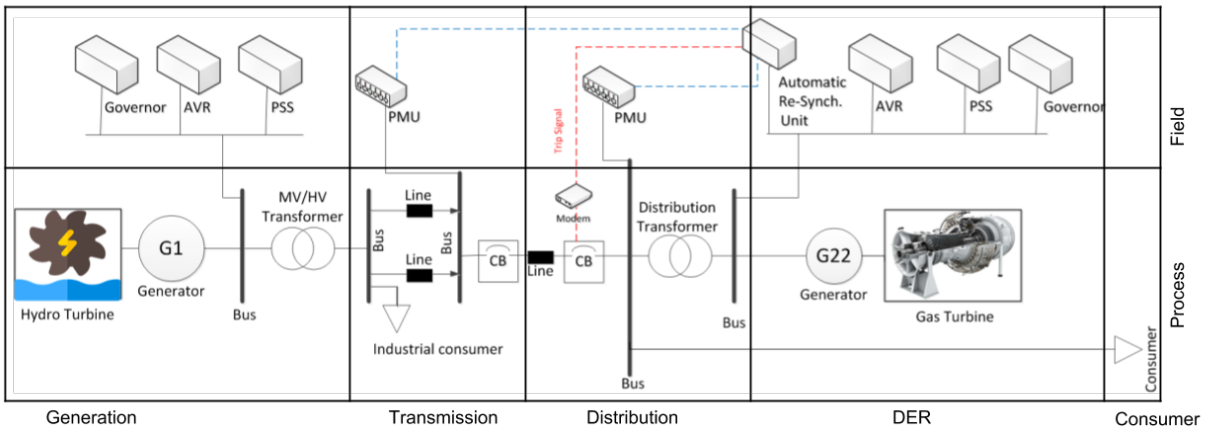


Figure 9. The use case mapped on the SGAM component layer.

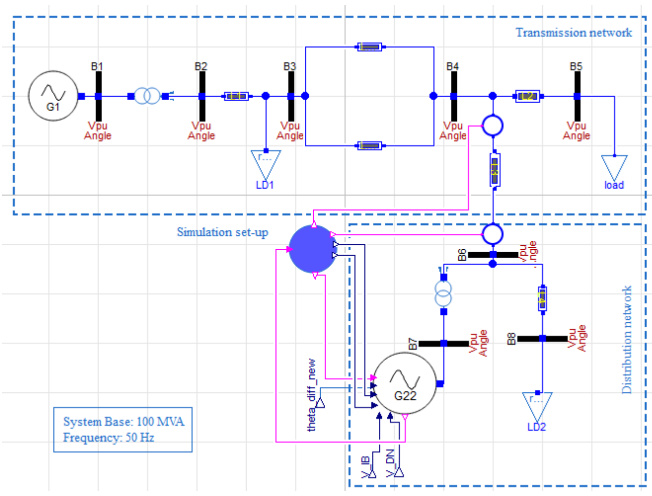


Figure 10. Power network models using components from OpenIPSL in Modelica.

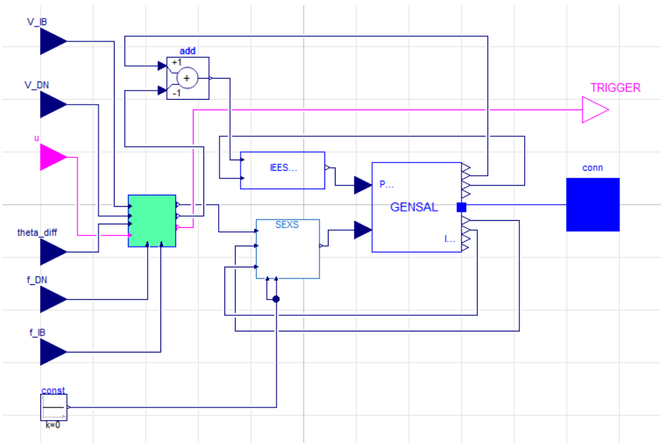


Figure 11. Centralized control structure within the generator model (G22) implemented using Modelica.

troller. In Figure 16a the phase angle difference of the bus voltages are plotted where it can be seen that during and after re-synchronization the re-synchronization the bus voltage angle difference also remains zero, implying ideal automatic re-synchronization. Figure 16b plots the triggered signal to the controlled circuit breaker CB2.

4.2 Performance of the Angle Control Function

When measured by PMUs the angle of the bus voltage phasor switches between +/- π (+/- 180 degrees), therefore the angle difference calculated directly from bus data the may corrupt the performance of the angle controller if it is wrapped. This case study exhibits the performance of the angle control for both wrapped and unwrapped angle calculations for a dispatch of 10 MW from generator G22, as shown in Figure 17. The the angle controller is activated at t= 101 seconds whereas re-synchronization process starts at t= 6.01 seconds . As it can be observed from the red trace, the unwrapped angle difference calculated from the wrapped angle differences produces no transients in its response, which makes the angle controller effective during the automatic re-synchronization process.

4.3 Effect of Angle Difference Controller during the Re-synchronization Process

This case study is performed to illustrate the effect of angle difference on the total time taken for re-synchronization, and also to understand the effect of angle controller on the circuit breaker current magnitude at the time of automatic breaker closure.

After the voltage and frequency control have minimized their control errors below the pre-defined thresholds the output of the angle difference controller plays an important role controlling the time for the automatic breaker re-closing process. The phase angle difference and the control signal to the circuit breaker 1 (CB1), are plotted in Figure 18 for different dispatches from the distribution side generator to demonstrate this effect. As it can be observed from in Figure 18, when the angle difference between the distribution side and transmission side

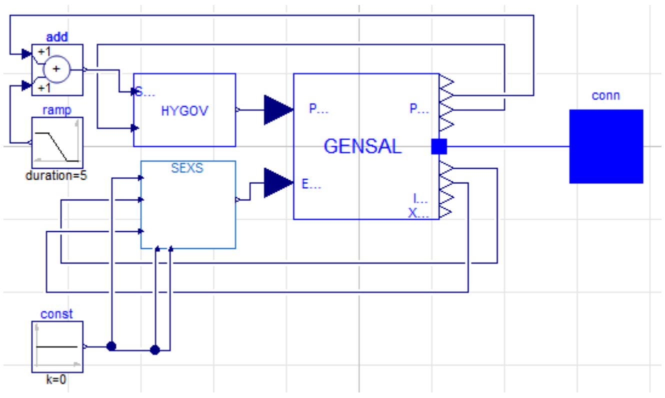


Figure 12. Implementation of the transmission network generator model (G1) in Modelica.

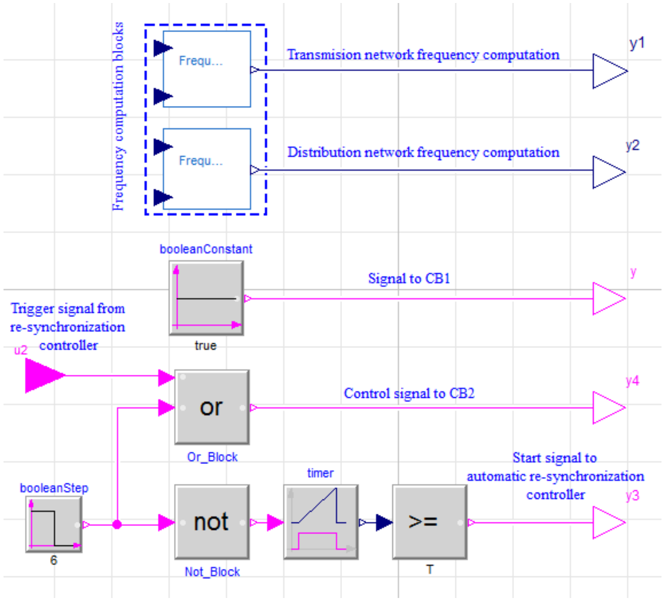
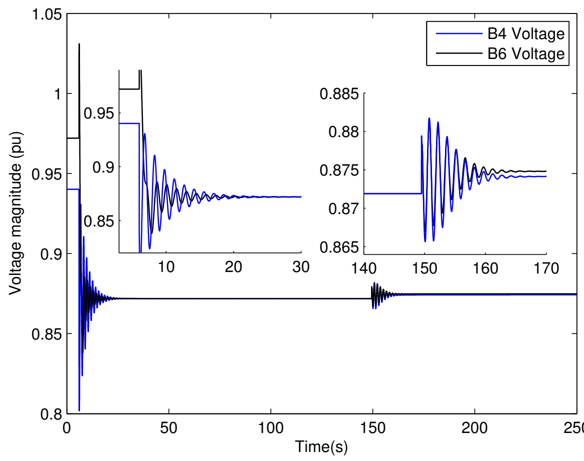
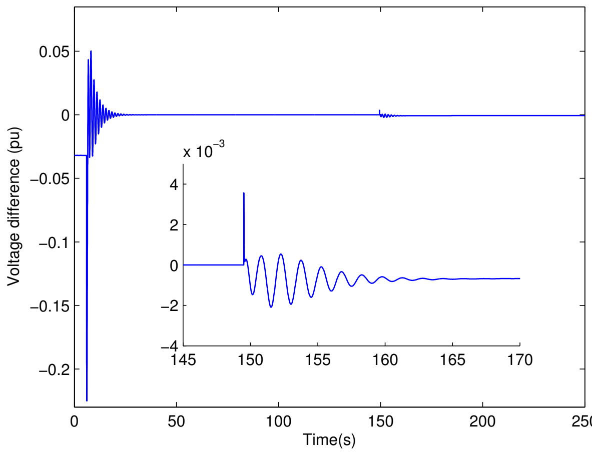

Frequency computation blocks
Transmision network frequency computation
y1
Distribution network frequency computation
y2
booleanConstant
Signal to CB1
y
Trigger signal from re-synchronization controller
u2
true
or
Control signal to CB2
y4
Or_Block
booleanStep
timer
Start signal to automatic re-synchronization controller
y3
not
>=
Not_Block
T

Figure 13. Modelica implementation of the simulation set-up.



(a) Transmission and distribution network bus voltages.



(b) Voltage difference during re-synchronization.

Figure 14. Bus voltages and bus voltage difference during the re-synchronization process for 10MW dispatch from G22.

bus voltage phasor reaches a steady state value, CB1 receives a trigger signal for automatic re-closure. For lower dispatches the angle difference controller is faster because the generator has a larger bandwidth (i.e. available capacity) to minimize the angle difference by speeding up the active power output. Hence there is a large value of distribution generation to (if possible) have available capacity resources to use for frequency ancillary services during re-synchronization to the transmission grids.

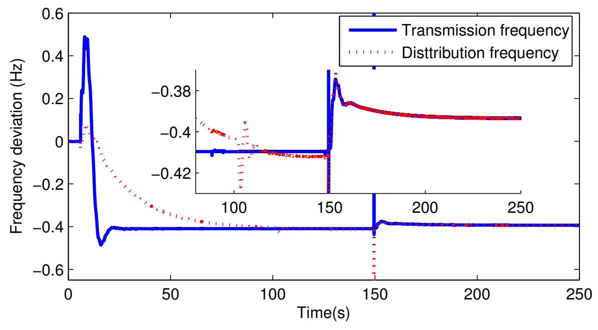
From Figure 19, it can be observed that with angle difference control activated the magnitude of the circuit breaker CB2 current reduces, which implies that presence of the angle control unit reduces unwanted effect of improper re-synchronization. This comes at the cost of a longer re-synchronization time, however, note that this is the time spent on minimizing $\Delta\theta$.

4.4 Effect of Stochastic Load Model

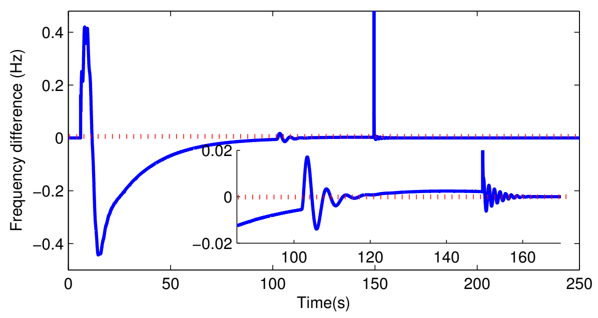
In this case the performance of the re-synchronization controller is analyzed for both deterministic and stochastic load models. Figure 20 shows the plot of the transmission side frequency for both models. As it can be observed, when the stochastic load model introduces uncertainties in the load response, which in turn, affect the voltage phasor values. As a result, the estimated frequency will vary, and consequently, the controller activation time can no longer be determined or designed deterministically. Depending on the variance used for the stochastic load model different answers can be obtained. What this aims to show is the need of stochastic modeling when considering renewable energy sources (RES), as the different thresholds need to be determined a-priori, they need to be computed probabilistically using the stochastic model and a Monte-Carlo like approach; note that this is NOT in today's current practice. This will be subject to future work.

5 Conclusions

The following conclusions can be drawn from the above work. The angle difference control function is required to



(**a**) Plot of frequency deviation in both transmission and distribution network



(**b**) Plot of frequency difference

Figure 15. Frequency deviation and frequency difference during re-synchronization for 10MW dispatch from G22.

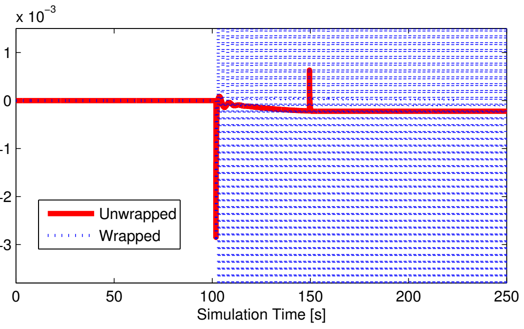


Figure 17. Performance of the angle difference controller due to angle measurement unwrapping.

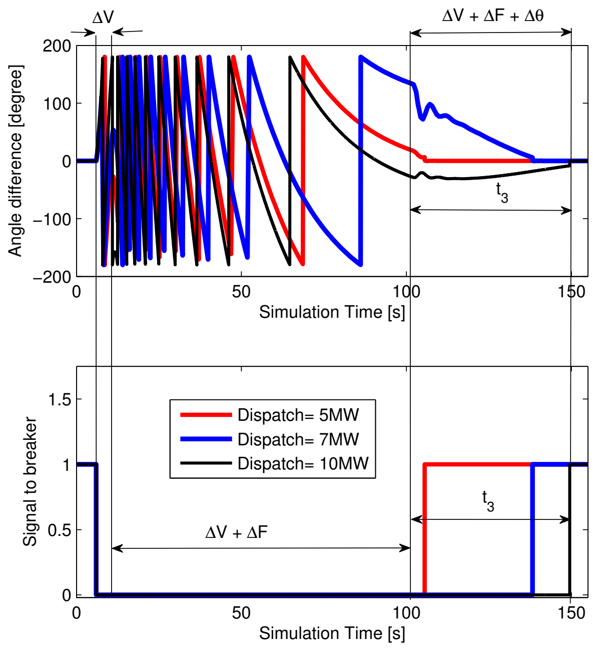
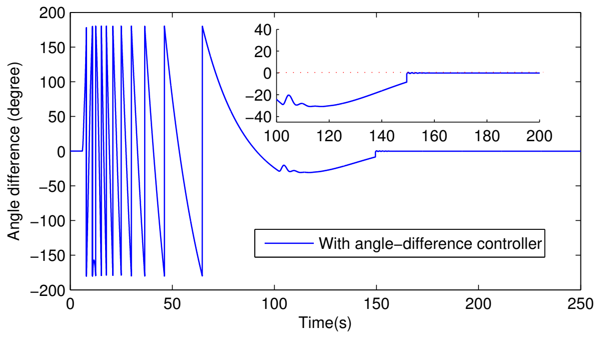
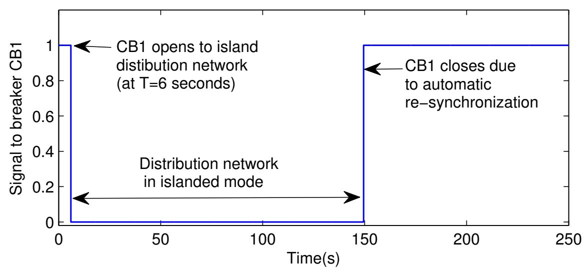


Figure 18. Angle difference controller effect on the re-synchronization time for different dispatches.



(**a**) angle difference



(**b**) signal to breaker CB1

Figure 16. Plot of angle difference and controlled signal to CB1 during re-synchronization for 10MW dispatch from G22

perform seamless automatic re-synchronization process, hence, reducing the circuit breaker power at the time of re-connection. It is necessary to unwrap the bus voltage phase angle otherwise the wrapped bus angle will corrupt the performance of the angle controller. The architecture of the automatic re-synchronization controller performs satisfactorily for both deterministic and stochastic load models.

Further work should involve the performance analysis of the automatic re-synchronization controller for different reporting rates, and data transmission delays for PMU devices. It will be also be interesting to investigate the performance of this controller for different of load uncertainties (noise level) in order to analyze the impact of uncertainties on the system behaviour. Authors of this paper are currently working towards these goals.

Reproducible Research

The models used to obtain the results in this paper are available online on the following Github repository: `https://github.com/`

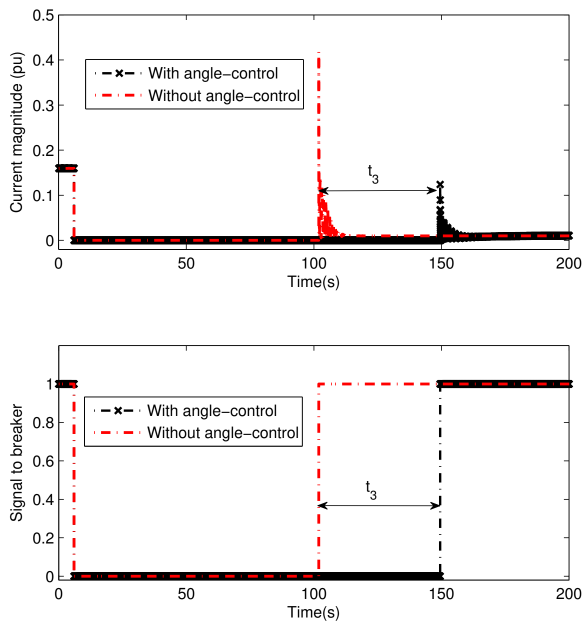


Figure 19. Circuit breaker current magnitude and trigger signal to breaker with and without angle control for 10MW dispatch.

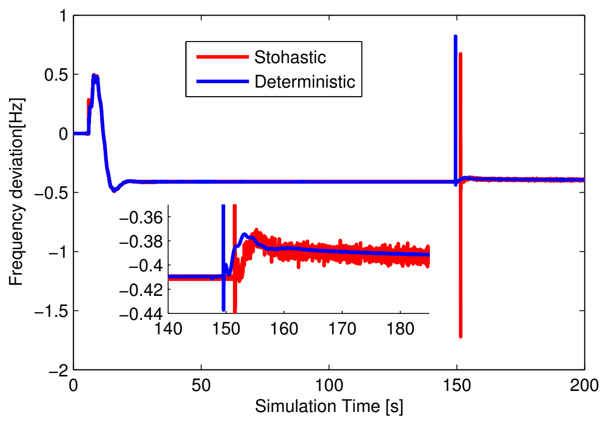


Figure 20. Transmission network frequency deviation with both deterministic and stochastic load models.

ALSETLab/2019_13thModelicaConf_ PMUBasedAutomaticRe-synchronization

Acknowledgment

Biswarup Mukherjee was supported by MHRD, Government of India, for FOSSEE-Phase II at IIT Bombay. The work of L. Vanfretti is supported in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877 and in part by the CURENT Industry Partnership Program.

References

M. S. Almas and L. Vanfretti. RT-HIL implementation of the hybrid synchrophasor and GOOSE-based passive islanding schemes. *IEEE Transactions on Power Delivery*, 31(3):1299–1309, June 2016. ISSN 0885-8977. doi:10.1109/TPWRD.2015.2473669.

T. Assis and G. Taranto. Automatic reconnection from intentional islanding based on remote sensing of voltage and frequency signals. In *2013 IEEE Power Energy Society General Meeting*, pages 1–1, July 2013. doi:10.1109/PESMG.2013.6672101.

N. A. Belyaev, Y. V. Khrushchev, S. V. Svechkarev, A. V. Prokhorov, and L. Wang. Generator to grid adaptive synchronization technique based on reference model. In *2015 IEEE Eindhoven PowerTech*, pages 1–5, June 2015. doi:10.1109/PTC.2015.7232582.

F. Blaabjerg, R. Teodorescu, M. Liserre, and A. V. Timbus. Overview of control and grid synchronization for distributed power generation systems. *IEEE Transactions on Industrial Electronics*, 53(5):1398–1409, Oct 2006. ISSN 0278-0046. doi:10.1109/TIE.2006.881997.

Peter Fritzson. *The Modelica Language*. IEEE, 2004. doi:10.1109/9780470545669.part2. URL `https://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5264368`.

H. Hooshyar and L. Vanfretti. A sgam-based architecture for synchrophasor applications facilitating tso/dso interactions. In *2017 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5, April 2017. doi:10.1109/ISGT.2017.8085977.

Harold Kirkham, Jeffery E. Dagle, and Y. Sun. PMU measurement technology. Technical report, Pacific Northwest National Laboratory (PNNL), 04/2014 2014.

A. Mazloomzadeh, V. Salehi, and O. Mohammed. Soft synchronization of dispersed generators to micro-grids for smart grid applications. In *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, pages 1–7, Jan 2012. doi:10.1109/ISGT.2012.6175812.

F. Milano and A. Ortega. Frequency divider. *IEEE Transactions on Power Systems*, 32(2):1493–1501, March 2017. ISSN 0885-8950. doi:10.1109/TPWRS.2016.2569563.

B. Mukherjee and L. Vanfretti. Modeling of PMU-based islanded operation controls for power distribution networks using Modelica and openIPSL. In *Proceedings of The American Modelica Conference, MA, USA*, October 2018.

M. J. Thompson. Fundamentals and advancements in generator synchronizing systems. In *2012 65th Annual Conference for Protective Relay Engineers*, pages 203–214, April 2012. doi:10.1109/CPRE.2012.6201234.

H. S. Timorabadi and F. P. Dawson. A wide-range synchronization system for AC power systems. In *2006 IEEE International Symposium on Industrial Electronics*, volume 3, pages 1667–1672, July 2006. doi:10.1109/ISIE.2006.295820.

L. Vanfretti, T. Rabuzin, M. Baudette, and M. Murad. itesla power systems library (iPSL): A Modelica library for phasor time-domain simulations. *SoftwareX*, 5:84 – 88, 2016. ISSN 2352-7110. doi:http://dx.doi.org/10.1016/j.softx.2016.05.001.

A Fundamental Time-Domain and Linearized Eigenvalue Analysis of Coalesced Power Transmission and *Unbalanced* Distribution Grids using Modelica and the OpenIPSL

Marcelo de C. Fernandes[1,†] Luigi Vanfretti[1,‡] Janaína G. de Oliveira[2] Maxime Baudette[3]

[1]Dept. of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, USA,
† decasm3@rpi.edu, ‡ luigi.vanfretti@gmail.com
[2]Dept. of Electrical Energy, Federal University of Juiz de Fora, Brazil, janaina.oliveira@ufjf.edu.br
[3]Grid Integration Group, Energy Storage & Distributed Resources Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, baudette@lbl.gov

Abstract

This paper present mathematical modeling and implementation in Modelica language of a coalesced electric power transmission and distribution system model. To this end, a newly developed feature in OpenIPSL that allows to amalgamate power transmission and distribution networks at the equation level is described, two different sample power systems are assembled and three simulations are performed for each of them in a Modelica-compliant software. Dynamic simulations are carried out to perform comparisons between different modeling approaches for a distribution feeder and among different load characteristics. Moreover, each simulation is linearized using a script in ten specific time instants and an eigenvalue comparison is performed. Results show that the conventional positive sequence models may lead to errors about the dynamic behavior of the entire system, specially when considering unbalances in distribution networks.

Keywords: Modelica, Power Systems, Hybrid Models, Linearization, Eigenvalues, Transmission Networks, Distribution Networks

1 Introduction

1.1 Paper Motivation

In the past few years, the world has began to undergo an important energy transition. Major environmental concerns and the need for diversify the energy mix have pushed society to look for alternative energy sources. As a consequence, governments have encouraged investments in renewable energy in order to increase energy system sustainability.

This energy transition has a major importance in the electric power sector, in which renewable sources are being integrated. Renewable technologies accounted for 25% of the world's generation of electricity in 2016 (IEA., 2017). In addition, renewable energy supply has increased 4% a year since 2000 (IEA., 2017), showing that the share of renewable energy sources in power systems will not slowdown in the near future.

Renewable energy sources may be connected to the power grid on medium or low voltage levels as Distributed Generation (DG), bringing challenges to the grid's operation (Boemer et al., 2011). These challenges are a consequence of the lack of understanding and adequate modeling of Distributed Energy Resources (DERs) during the design phase, particularly their control and protection, in studies of their performance when integrated to transmission grids (ENTSO-E, 2014).

Therefore, it has become evident that distribution networks, and all its components, can no longer be neglected from studies assessing power systems and its dynamic behavior (Jain et al., 2016). In order to address these issues, many tools and strategies have been proposed in the last years in order to perform the analysis of joint power transmission and distribution (T&D) systems. These tools, their modeling strategies and simulation approaches are discussed next.

1.2 Background

Modeling assumptions established ever since the beginning of long distance AC power transmission in the early 1900's have led to a decoupled treatment of transmission from distribution, and *vice versa*. Thus, analysis of each of them has been carried out individually, using a different approach for each network in order to analyze their behavior. As an example, the transmission system is often represented on its single-phase equivalent or positive sequence modeling (Tinney and Hart, 1967). This is due to the assumption that bulk power systems can be considered to operate under completely balanced conditions. On the other hand, distribution grids normally undergo unbalances due to many factors, and thus, simulation tools for distribution networks commonly used three-phase models (Garcia et al., 2000).

In fact, many proprietary and commercial software packages made for the study of the power system are built upon the aforementioned assumptions, especially when the software provides phasor time-domain analysis routines. For instance, positive sequence modeling is typical in software packages such as PSS/E (Siemens) originally developed in the U.S.A., Anarede (Cepel, a) and Anatem (Cepel, b) developed in Brazil. On the other hand, three-phase analysis methods are available in CYMDIST (EATON) and OpenDSS (EPRI) both developed in North America. Coupling between models from these type of tools is generally only possible through co-simulation and the common belief of practitioners was that these modeling approaches were largely incompatible (Balasubramaniam and Abhyankar, 2017).

In order to solve this assumed incompatibility between modeling approaches and simulation tools, a hybrid model, named Monotri, was proposed in (Marinho and Taranto, 2008). This model was the first to provide a "physical interface" between a system modeled in positive sequence with another one using three-phase modeling, for power flow studies. Concurrently, the same research group highlighted the importance of considering DERs in dynamic simulations (Assis et al., 2006). Later, the hybrid formulation presented in (Marinho and Taranto, 2008) was extended in (Taranto and Marinho, 2017) to transient stability studies, i.e. dynamic modeling and simulation. This hybrid model has two main benefits: (1) it provides a "physical interface" to couple Transmission and Distribution T&D instead of using co-simulation, and (2) it combines usual modeling approaches that are familiar to power system domain users.

1.3 Modelica Tools for Power Systems

Previous work has shown that the Modelica language is a promising alternative for modeling the complexity of modern power systems (Vanfretti et al., 2013). For instance, (Mirz et al., 2016) present a multi-level approach to model power electronics in power systems using Modelica, while (Casella et al., 2016, 2017) study the feasibility of using Modelica-based tools to solve large power system models.

Along these and other many studies, developers have built several Modelica libraries for power system simulation. The study presented in (Winkler, 2017) lists many libraries for power system analysis, along with their history, modeling principles, library structure, weaknesses and strengths. Among those packages there is the OpenIPSL (Baudette et al., 2018).

According to (Winkler, 2017), the OpenIPSL package has many strengths such as its robustness and its models. OpenIPSL models underwent software-to-software validation against domain-specific proprietary and open source software packages including PSS/E and PSAT (Milano, 2005). In addition, OpenIPSL is friendly to users familiar to typical power system analysis, as it addresses *resistance to change*-factors, associated with the use of new technologies (Vanfretti et al., 2014). OpenIPSL is still being developed in voluntary basis as an Open Source Software project on Github (`http://openipsl.org`) and new features are being added. For example, its latest version comes with an application example package made for modeling positive sequence systems with three-phase networks using the hybrid interface proposed in (Marinho and Taranto, 2008), for which some results are reported in (de Castro Fernandes et al., 2018) (in Portuguese).

In addition, models built using the OpenIPSL package may take advantage of the rich features available in Modelica-based tools, such as linearization. Linear analysis and small-signal studies for integrated T&D systems are not commonly available in power system tools. However, this analysis can be extremely useful to understand the dynamics of DERs at lower voltage levels. With Modelica tools it is possible to perform this analysis without the need of encoding the linear and non-linear models separately. This paper exploits and demonstrates this possibility, by analyzing a hybrid positive-sequence and three-phase T&D model implemented using OpenIPSL.

1.4 Paper Contributions

The main contributions from this paper are listed below:

- Model description and implementation in the Modelica language of a hybrid single-phase $\times$ three-phase element, π-modeled three-phase lines and wye-grounded loads. All using a phasor approach with an OpenIPSL library.

- Exploring the rich features of Modelica language for simulation, linearization and eigenvalue analysis of a small joint-modeled transmission and distribution system, avoiding the co-simulation approach.

- Analyses and comparisons of the positive-sequence versus the hybrid-modeling approach in the stability analysis of power systems.

2 Mathematical Modeling

2.1 Studied Power Network

The power system studied in this paper is described in Figure 1 and it consists of an adaptation of IEEE 14-bus test system, first implemented using OpenIPSL in (Murad et al., 2015). This transmission network was extended to include synchronous machines and a distribution feeder.

To model different modeling impacts of *unbalanced* distribution networks, loads at buses 2 and 11 are modified, one at a time, in different case studies. The modification is based upon an extension of the original load itself, to include a distribution feeder consisting of two buses, a power line and a load. In one set of tests, the load on bus 2 is replaced by this distribution feeder and in another set of tests, the replaced load is on bus 11. The distribution feeder is connected to the respective bus by a transformer. This element must be modeled using the hybrid formulation proposed in (Marinho and Taranto, 2008). The remainder of the power system is modeled using positive-sequence models, including all dynamic components as specified in (Kodsi and Canizares, 2003), starting from the implementation in (Murad et al., 2015). The generators are labeled $G1$ and $G2$, while condensers are $C1$, $C2$ and $C3$, in Figure 1.

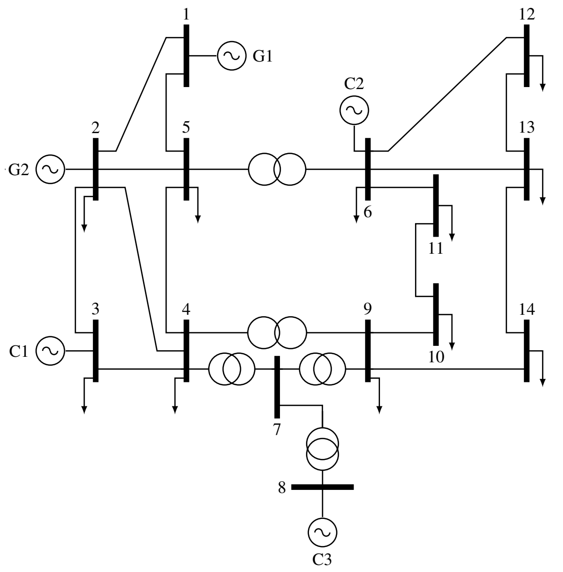


Figure 1. Modified IEEE 14-bus test system diagram.

Positive-sequence buses and branches are available in the OpenIPSL library while the hybrid and three-phase elements are included in the `./ApplicationExamples.ThreePhase` package, within the OpenIPSL distribution. Models for all machines are also available in the OpenIPSL library. The *ThreePhase* package was recently added by the authors (Baudette et al., 2018; de Castro Fernandes et al., 2018) and thus, it is useful to present the mathematical modeling of such elements, along with their implementation in the Modelica language.

2.2 Hybrid and Three-Phase Models

This subsection introduces the mathematical formulation for hybrid and three-phase models. It is important to note that since this paper is related to power system modeling, the value for each variable is represented in per unit (p. u.), which is the representation of the system's quantities as fraction of a base value. For this subsection, values of voltage, current, power, admittance and conductance, are represented in this way. In addition, variables written in bold are matrices and vectors while variables written in regular letters represent scalars. In addition, variables with the upper bar ($^-$) denotes phasors.

The formulation of a hybrid power component is proposed in (Marinho and Taranto, 2008; Taranto and Marinho, 2017) and it consists of a passive π element, as depicted in Figure 2.

Equations for a three-phase π element are:

$$\begin{bmatrix}\mathbf{I}_k^{abc}\\ \mathbf{I}_m^{abc}\end{bmatrix} = \begin{bmatrix}\mathbf{Y}_{ser}^{abc}+\mathbf{Y}_{sht_k}^{abc} & -\mathbf{Y}_{sht_k}^{abc}\\ -\mathbf{Y}_{ser}^{abc} & \mathbf{Y}_{ser}^{abc}+\mathbf{Y}_{sht_m}^{abc}\end{bmatrix}\begin{bmatrix}\mathbf{V}_k^{abc}\\ \mathbf{V}_m^{abc}\end{bmatrix} \quad (1)$$

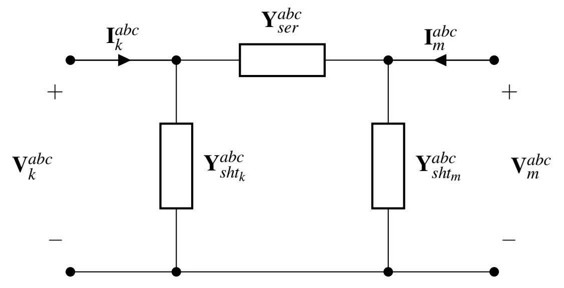


Figure 2. Representation of three-phase, passive π-equivalent component, such as a distribution power line or transformer.

where $\mathbf{V}_x^{abc}$, $\overline{I}_x^{abc}$ are vectors of three-phase voltage and injected current phasors, respectively, for a terminal $x = k,m$, while $\mathbf{Y}_{ser}^{abc}$ and $\mathbf{Y}_{sht}^{abc}$ are series and shunt admittance matrices, respectively.

Assuming that the system in terminal k is completely balanced and that there is no negative component current source in terminal m, it is possible to ignore zero and negative sequence components, resulting in:

$$\begin{cases}\mathbf{V}_k^{abc} = \mathbf{T}_1\overline{V}_k^+ = \begin{bmatrix}1\\ \alpha^2\\ \alpha\end{bmatrix}\overline{V}_k^+\\ \overline{I}_k^+ = \mathbf{T}_2\mathbf{I}_k^{abc} = \dfrac{1}{3}\begin{bmatrix}1 & \alpha & \alpha^2\end{bmatrix}\mathbf{I}_k^{abc}\end{cases} \quad (2)$$

where $\alpha = e^{j\frac{2\pi}{3}}$ and $\overline{V}_k^+$ e $\overline{I}_k^+$ are positive-sequence phasors for voltage and injected current in terminal k, respectively. Therefore, replacing (2) in (1) it is possible to write the final equation for a hybrid model:

$$\begin{bmatrix}\overline{I}_k^+\\ \mathbf{I}_m^{abc}\end{bmatrix} = \begin{bmatrix}\mathbf{M}_{SS} & \mathbf{M}_{ST}\\ \mathbf{M}_{TS} & \mathbf{M}_{TT}\end{bmatrix}\begin{bmatrix}\overline{V}_k^+\\ \mathbf{V}_m^{abc}\end{bmatrix} \quad (3)$$

where

$$\begin{aligned}\mathbf{M}_{SS} &= \mathbf{T}_2\left(\mathbf{Y}_{ser}^{abc}+\mathbf{Y}_{sht_k}^{abc}\right)\mathbf{T}_1\\ \mathbf{M}_{ST} &= \mathbf{T}_2\left(-\mathbf{Y}_{ser}^{abc}\right)\\ \mathbf{M}_{TS} &= \left(-\mathbf{Y}_{ser}^{abc}\right)\mathbf{T}_1\\ \mathbf{M}_{TT} &= \mathbf{Y}_{ser}^{abc}+\mathbf{Y}_{sht_m}^{abc}\end{aligned} \quad (4)$$

A transmission or distribution power line can also be modeled as a three-phase passive π element (Arrillaga et al., 2001) and can also be described by (1). This model is adequate to represent unbalanced transmission lines such as the ones encountered in power distribution systems.

The model of the a three-phase load with grounded-wye connection is depicted in Figure 3. This load model will

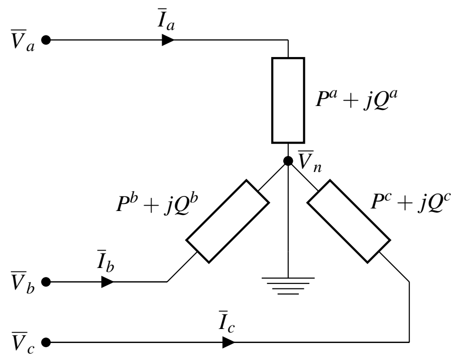


Figure 3. Representation of a generic three-phase load in a grounded wye connection.

be considered for the tests when the distribution feeder has a three-phase representation. For this load model, it is possible to write one equation for each phase independently:

$$\begin{cases} \overline{S}_a = P_a + jQ_a = \overline{V}_{an}\overline{I}_a^* \\ \overline{S}_b = P_b + jQ_b = \overline{V}_{bn}\overline{I}_b^* \\ \overline{S}_c = P_c + jQ_c = \overline{V}_{cn}\overline{I}_c^* \end{cases} \tag{5}$$

To define the active and reactive power on each phase (P_x and Q_x), the ZIP load model (Kersting, 2001) was chosen. It allows the representation of a load as a composition of three types of characteristics: constant power, constant current and constant impedance. In addition, it shows how the power demanded by the load varies according to its terminal voltage V_{xn}. Therefore, it is possible to write (6) for each phase, represented in the equation by under-script x. The load demands P_0 and Q_0 are those for a terminal voltage of V_{xn}^0. The coefficient α shows how much of the load is constant power (superscript ^{p}), current (superscript ^{i}) or impedance (superscript ^{z}).

$$\begin{cases} P_x(V_{xn}) = P_x^0\left[\alpha_x^p + \alpha_x^i\left(\frac{V_{xn}}{V_{xn}^0}\right) + \alpha_x^z\left(\frac{V_{xn}}{V_{xn}^0}\right)^2\right] \\ Q_x(V_{xn}) = Q_x^0\left[\alpha_x^p + \alpha_x^i\left(\frac{V_{xn}}{V_{xn}^0}\right) + \alpha_x^z\left(\frac{V_{xn}}{V_{xn}^0}\right)^2\right] \\ \alpha_x^p + \alpha_x^i + \alpha_x^z = 1 \end{cases} \tag{6}$$

3 Modelica Implementation

This section describes how the models developed in Section 2 were written in Modelica, and implemented within the OpenIPSL package. Because the actual implementation of each new component is rather large, this section aims to illustrate how the models were implemented, and not to document the implementation itself.

3.1 Studied Power Network in OpenIPSL

The IEEE 14-bus test system implemented in Modelica using OpenIPSL models is illustrated in Figure 4. The blue line shows the connection between elements. These connections are made between `PwPins` from different models. The `PwPin` is the main feature upon which the connections of electrical components is made. The pin itself works as an electrical circuit node allowing the flow of complex current variables, and providing two important pieces of information (supposing a `PwPin` named `P`):

- real and imaginary part of the voltage at that particular node (`P.vr` and `P.vi`);
- real and imaginary part of the current flowing through that node (`P.ir` and `P.ii`).

In a single-phase equivalent model, the `pin` has positive sequence voltages and currents. On the other hand, in a three-phase model, one `pin` is necessary for each phase represented. The connection between the machine G1 (namely GenBus1 in Figure 4) to bus B1 is defined using one-line code, as follows:

```
connect(GenBus1.pwPin, B1.p);
```

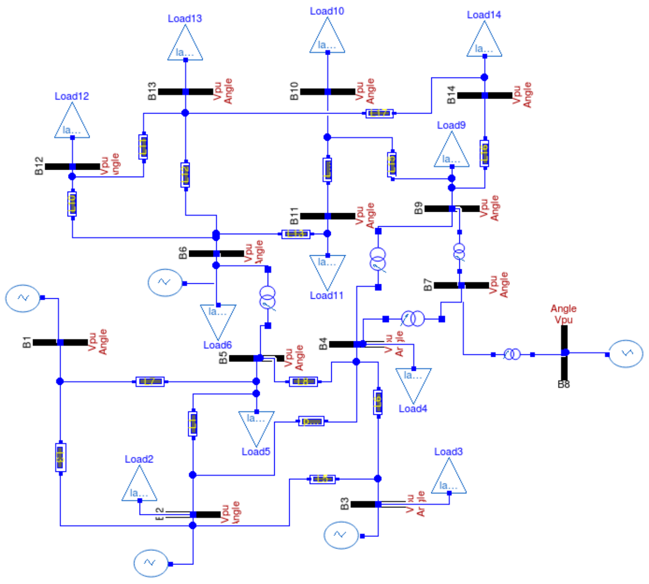


Figure 4. Original IEEE 14-bus system implemented in a Modelica-compliant software using OpenIPSL.

However, for the connection between the grounded-wye connected three-phase load and three-phase bus *Bus632*, three lines of code are needed:

```
connect(Bus632.p1, 3phLoad.A);
connect(Bus632.p2, 3phLoad.B);
connect(Bus632.p3, 3phLoad.C);
```

It is important to note that a `pin` representing phase A in one component must always be connected with respective `pin` representing the same phase in another component.

3.2 Hybrid Transformer

The element that interfaces the positive-sequence model with the three-phase distribution feeder is a transformer in a $\Delta - Y$ connection. This connection is chosen for consistency with the equations (2). In this case, a function `TransformerFcn.D_Yg` was implemented and is imported into the model in order to provide the values of M_{SS}, M_{ST}, M_{TS}, M_{TT} from (3) for a transformer in a $\Delta - Y$ connection. Calculating these matrices with functions is the most efficient approach because each transformer connection has a different Y^{abc}_{ser}, $Y^{abc}_{sht,k}$ and $Y^{abc}_{sht,m}$. This helps in describing, the hybrid transformer model as implemented below. The values for reactance and resistance are only used for illustration.

```
model Transformer_MT
  import ThreePhase.TransformerFcn.D_Yg;
  OpenIPSL.Interfaces.PwPin p;
  OpenIPSL.Interfaces.PwPin A;
  OpenIPSL.Interfaces.PwPin B;
  OpenIPSL.Interfaces.PwPin C;
  parameter Real tap = 1 "Nominal tap ratio
       (Vs/Vp)";
  parameter Real X=0.1 "Reactance (pu in
      system base)";
  parameter Real R=0.01 "Resistance (pu in
      system base)";
protected
  // Calculating M matrices:
  Real[2,2] M_SS = D_Yg(X,R,tap,1);
  Real[2,6] M_ST = D_Yg(X,R,tap,2);
  Real[6,2] M_TS = D_Yg(X,R,tap,3);
  Real[6,6] M_TT = D_Yg(X,R,tap,4);
  // Writing matrix for voltages:
  Real [2,1]Vin = [p.vr; p.vi];
  Real [6,1]Vout = [A.vr; A.vi; B.vr; B.vi;
       C.vr; C.vi];
  // Writing matrix for currents:
  Real [2,1]Iin = [p.ir; p.ii];
  Real [6,1]Iout = [A.ir; A.ii; B.ir; B.ii;
       C.ir; C.ii];
equation
  // Equations related to hybrid interface
  Iin =  A*Vin+B*Vout;
  Iout =  C*Vin+D*Vout;
end Transformer_MT;
```

3.3 Three-Phase Line

The implementation of the three-phase transmission line modeled as a π-element is straight forward using Modelica. However, observe that due to the `PwPin` voltage and current convention, complex numbers are not used but instead two real numbers representing imaginary and real parts are used in the `PwPin`. Therefore, matrices Y^{abc}_{ser} and Y^{abc}_{sht} are designed according to this convention. Using equation (1) it is possible to write the following model:

```
model Line_3Ph
  OpenIPSL.Interfaces.PwPin Ain;
  OpenIPSL.Interfaces.PwPin Bin;
  OpenIPSL.Interfaces.PwPin Cin;
  OpenIPSL.Interfaces.PwPin Aout;
  OpenIPSL.Interfaces.PwPin Bout;
  OpenIPSL.Interfaces.PwPin Cout;
  parameter Real[6,6] Y_ser;
  parameter Real[6,6] Y_sht;
protected
  //Writing the matrix A (Yser+Ysht):
  parameter Real[6,6] A = Y_ser+Y_sht;
  //Writing the matrix B (-Yser):
  parameter Real[6,6] B = -Y_ser;
  // Writing matrix for voltages:
  Real [6,1]Vin = [Ain.vr; Ain.vi; Bin.vr;
     Bin.vi; Cin.vr; Cin.vi];
  Real [6,1]Vout = [Aout.vr; Aout.vi;
     Bout.vr; Bout.vi; Cout.vr; Cout.vi];
    // Writing matrix for currents:
  Real [6,1]Iin = [Ain.ir; Ain.ii; Bin.ir;
     Bin.ii; Cin.ir; Cin.ii];
  Real [6,1]Iout = [Aout.ir; Aout.ii;
     Bout.ir; Bout.ii; Cout.ir; Cout.ii];
equation
  // Equations according to pi model:
  Iin =  A*Vin+B*Vout;
  Iout =  B*Vin+A*Vout;
end Line_3Ph;
```

3.4 Three-Phase Load

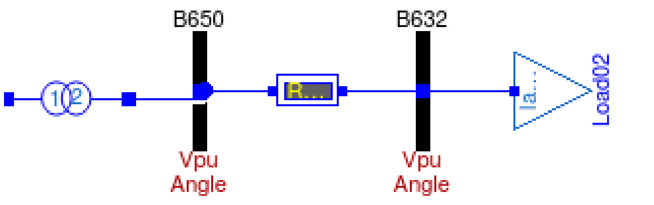
The three-phase load model implementation in Modelica is also straight forward. The implementation of equation (6) for each phase is shown in the Modelica code below. The code shows the example values for active and reactive power. In addition, note that a constant impedance load is being used.

```
model WyeLoad_3Ph
  outer OpenIPSL.Electrical.SystemBase
     SysData;
  import Modelica.Constants.pi;
  parameter Real Sn = SysData.S_b "Power
     rating (MVA)"
  OpenIPSL.Interfaces.PwPin A;
  OpenIPSL.Interfaces.PwPin B;
  OpenIPSL.Interfaces.PwPin C;
  parameter Real[1,3] P =
      [1.234,0.984,1.306] "Active power for
       phase A, B and C (MW)"
  parameter Real[1,3] Q =
      [0.635,0.365,0.619] "Reactive power
     for phase A, B and C (MVAr)"
  parameter Real [3,3] Coef = [0, 0, 100;
     0, 0, 100; 0, 0, 100] "Load
     percentages for constant power,
     current and impedance (%)"
protected
  // Calculating V and V squared:
  Real[1,3] V = [sqrt(A.vr^2 + A.vi^2),
     sqrt(B.vr^2 + B.vi^2), sqrt(C.vr^2 +
     C.vi^2)];
  Real[1,3] V2 = [V[1,1]^2, V[1,1]^2, V
      [1,3]^2];
  // Calculating the polynomial value:
  Real CoA = Coef[1,1] + Coef[1,2]*V[1,1] +
      Coef[1,3]*V2[1,1];
  Real CoB = Coef[2,1] + Coef[2,2]*V[1,2] +
      Coef[2,3]*V2[1,2];
  Real CoC = Coef[3,1] + Coef[3,2]*V[1,3] +
      Coef[3,3]*V2[1,3];
  // Calculating the new values for power:
  Real[1,3] Pnew = (1/(Sn/3))*[P[1,1]*CoA,
      P[1,2]*CoB, P[1,3]*CoC];
  Real[1,3] Qnew = (1/(Sn/3))*[Q[1,1]*CoA,
      Q[1,2]*CoB, Q[1,3]*CoC];
equation
  Pnew[1,1] = A.vr*A.ir + A.vi*A.ii;
  Qnew[1,1] = A.vi*A.ir - A.vr*A.ii;
  Pnew[1,2] = B.vr*B.ir + B.vi*B.ii;
  Qnew[1,2] = B.vi*B.ir - B.vr*B.ii;
  Pnew[1,3] = C.vr*C.ir + C.vi*C.ii;
  Qnew[1,3] = C.vi*C.ir - C.vr*C.ii;
end WyeLoad_3Ph;
```

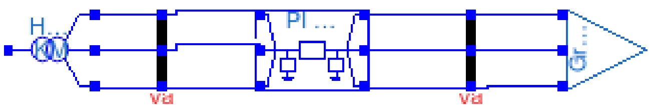
4 Simulation Methodology

In this paper, the original IEEE 14-bus system, depicted in Figure 4, is modified in two different ways and three simulations are performed in each modified system, resulting in a total of six dynamic simulations. The common characteristic among all six experiments is that a small feeder, consisting of two buses, one transformer, one transmission line and one load is added to a transmission bus, effectively extending it to represent a T&D system. In three experiments this feeder replaces the load in bus number 11. In the remaining three experiments, the feeder replaces the load in bus number 02.

Consider the experiments related to the feeder addition to bus 02. One experiment will consist of the addition of a feeder modeled in positive sequence, as depicted in Figure 5(a). The load at the end of the feeder increase for 9 times, every 12 seconds. In the second experiment, the feeder is modeled in three-phase and the connection is made by a hybrid transformer, shown in Figure 5(b). In that experiment, the load increase occurs in a balanced way, again 9 times and every 12 seconds. The last experiment for bus 02 is based in the addition of the same three-phase feeder depicted in Figure 5(b), however, the load increases with phase unbalances.



(a) Positive sequence model.



(b) Three-phase model.

Figure 5. Distribution feeder model diagrams used in the experiments.

The experiments related to the addition of the feeder to bus 11 are similar to what was described for bus 02. The difference between both set of experiments lies in parameter values for the line, load and transformer. All simulations are listed in Table 1.

Table 1. Summary of simulations.

Test	*Description*
I	Distribution feeder connected to bus 02. Positive sequence model. 9 load steps
II	Distribution feeder connected to bus 02. Three-phase model. 9 balanced load steps
III	Distribution feeder connected to bus 02. Three-phase model. 9 unbalanced load steps
IV	Distribution feeder connected to bus 11. Positive sequence model. 9 load steps.
V	Distribution feeder connected to bus 11. Three-phase model. 9 balanced load steps.
VI	Distribution feeder connected to bus 11. Three-phase model. 9 unbalanced load steps.

5 System and Simulation Parameters

This section provides information about the parameters used for the simulation of the system presented in Sections 2 and 3 and in Subsection 4. Table 2 lists the machine models that were used in this study. All machines, generators and condensers, are represented with the same dynamic model. The parameters for these machines along with voltage regulators are described in (Kodsi and Canizares, 2003), which also provides load values and branch parameters. In order to increase the total load of the system, the load values reported in the original IEEE14 system are multiplied by a factor of 1.4, which is the same procedure adopted in (Kodsi and Canizares, 2003). It is very important to mention that the machines do not include automatic speed regulators. Thus, the value for mechanical power is kept constant during all the simulation and equal to the steady state value of electrical power in the initial condition. This means that with load increasing, the system's frequency will drift away from 60 *Hz* to lower values. However, in this performed study the frequency does not change significantly from the base value.

Table 2. Parameters for synchronous machines.

Equipment	*Model used*
Machine	*PSAT.Order6*
Voltage regulator	*PSAT.AVR.AVRTypeII*
Speed regulator	–

Table 3 lists system-wide parameters, such as its frequency and its base power. The latter is used to calculate all per-unit values in the system, such as the resistance and reactance from the hybrid transformer.

Table 3. System parameters.

Description	*Value*
System Frequency	60 *Hz*
Power Base	100 *MVA*

Table 4 presents all the simulation data chosen for running in all simulations (i.e. scenarios I to VI).

Table 4. Simulation parameters.

Description	*Value*
Duration	120 *s*
Interval Size	0.005 *s*
Integration Method	*dassl*
Tolerance	10^{-5}

The values for transformer's reactance (X_{tf}), power line's mutual (Z_m) and self (Z_s) impedance, and load steps(ΔS) are important parameters that need to be explicitly described. They are summarized in Table 5.

6 Simulation Results and Linearization Analysis

All tests described in Section 4 were simulated using a Modelica-compliant software tool. Results for the simulations corresponding to the distribution feeder modeled in positive sequence (simulations I and IV) are shown in Figure 6. Bus 02 voltage in simulation I is depicted in Figure 6(a), while bus 11 voltage in simulation IV, is shown in Figure 6(b). Note that larger oscillations occur in simulation I, which are due to the value of the load step. However, the voltage variation, in per unit (pu), is higher in simulation IV than I. This is due to the location of the distribution feeder where the load is increased.

The comparison between bus 02 voltage in simulations I, II and III is shown in Figure 7(a), during the last load step increase, which occurs between 108 *s* and 120 *s*. In that figure, the blue solid curve represents the voltage curve for simulation I, while the dashed red line, marked with squares, represents simulation II and the solid green line, marked with circles, represents simulation III. It is important to observe that the curve for simulation II overlaps the one for simulation I. This result should be expected because a perfect balanced three-phase system may be represented its positive sequence equivalent model, which implies that the three-phase implementation has been performed correctly.

The second important fact that should be observed is that the green curve, resulting from simulation III, has a different value if compared to the blue and red curves. The difference may be small (approximately 0.05 %) but it is important to be noted. This divergence between the curves should be expected, since the unbalance should reduce the value of positive sequence component.

A similar analysis is done in Figure 7(b), which depicts voltage in bus 11 for simulations IV, V and VI during the last load step increase. The solid blue line curve represents the voltage curve for simulation IV, the dashed red line, marked with squares, represents the voltage for simulation V and the solid green line, marked with circles, represents the result for simulation VI. Note that, again, blue and red curves overlaps, as expected, and the green curve diverges from the other two.

After verifying that the voltage curves behaves as expected in time domain simulations, a script was made in order to explore the easy access to linearization process provided by Modelica language. The script linearizes each of the six simulations one time step, 0.005 *s*, before the load step increase, totaling the value of ten processes for every simulation. From each linear model, fifty eigenvalues are extracted and stored for comparison. As expected, the eigenvalues from the ten linear models coming from simulation I are virtually the same ones coming from simulation II. The total absolute error between the fifty eigenvalues from the ten linear models coming from simulations I and II is 0.00017. The same fact is observed when comparing the eigenvalues from linear models coming from simulations IV and V. In this case, the total error between the eigenvalues is 0.00001.

The next analysis conducted in this paper is made using specific eigenvalue collected from in simulations I and III. It is important to observe that both eigenvalues become apart from each other, and the distance increase with the load step. The difference between this eigenvalue locus is shown in Figure 8. Note that the eigenvalues from the unbalanced case have less negative real part and more negative imaginary part if compared to the respective eigenvalue from the balanced case.

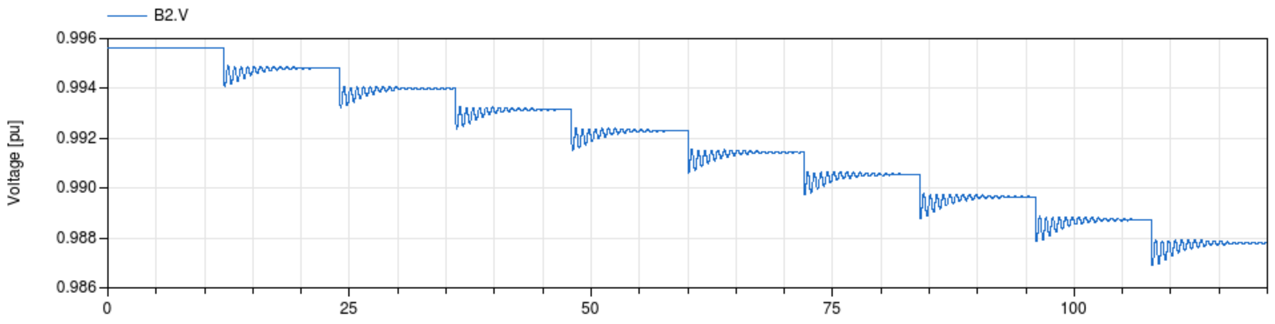
A similar analysis is conducted using the comparison of one specific eigenvalue coming from simulations IV and VI. Again, the eigenvalues become more distant with the load variation. This variation is shown in Figure 9. In this case, the eigenvalue has no imaginary part but the real part of the eigenvalue become less negative with load steps.

7 Discussion

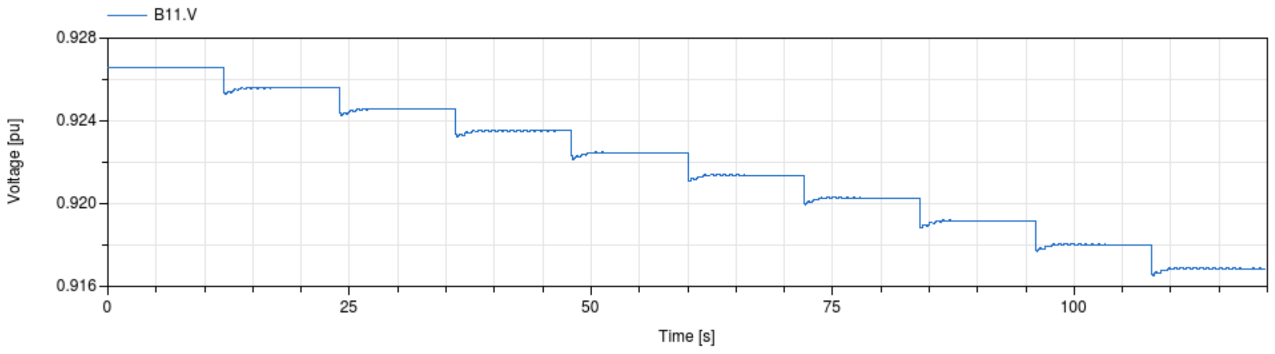
The distance between the eigenvalues from the different linearized systems is of great interest, because it provides crucial information about the system's stability. Here, the error in damping ($\Delta\sigma$) and in angular frequency ($\Delta\omega$) are analyzed. Table 6 presents the difference in damping and angular frequency in the eigenvalues represented in Figure 8. Note that the difference in both damping and frequency is not linear. Table 7 presents the difference in damping for eigenvalues depicted in Figure 9 and, again, the difference is not linear.

Table 5. Simulation system parameters details.

Test	$X_{tf}\ (pu)$	*Line Data* (pu)	$\Delta S\ (pu)$	*Load Description*
I	0.11001	$Z_m = 0.0032 + j0.0099$	$3.038 + j1.778$	Positive sequence increase.
II		$Z_s = 0.0226 + j0.0690$		Divided equally between phases A,B,C.
III				Divided equally between phases A and C.
IV	0.55618	$Z_m = 0.0233 + j0.0494$	$0.490 + j0.252$	Positive sequence increase.
V		$Z_s = 0.2443 + j0.2493$		Divided equally between phases A,B,C.
VI				Divided equally between phases A and C.



(a) Voltage in bus 02 for simulation I, distribution feeder in bus 02.



(b) Voltage in bus 11 for simulation IV, distribution feeder in bus 11.

Figure 6. Voltage behavior in buses 02 and 11 during the 120 seconds of simulation.

Table 6. Comparison between eigenvalues of tests I and III.

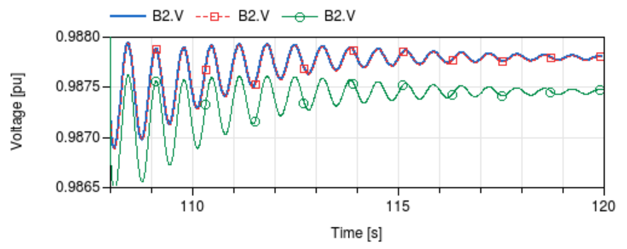
Load Step #	**Error in Damping** $(\Delta\sigma)$	**Error in Frequency** $(\Delta\omega)$
1	-0.00003	0.00003
2	-0.00010	0.00011
3	-0.00030	0.00027
4	-0.00059	0.00047
5	-0.00094	0.00076
6	-0.00143	0.00110
7	-0.00193	0.00156
8	-0.00258	0.00210
9	-0.00335	0.00273

Table 7. Comparison between eigenvalues of tests IV and VI.

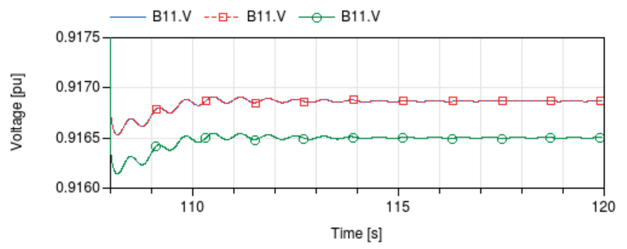
Load Step #	**Error in damping** $(\Delta\sigma)$
1	-0.00002
2	-0.00009
3	-0.00021
4	-0.00038
5	-0.00062
6	-0.00093
7	-0.00133
8	-0.00183
9	-0.00245

8 Conclusions

This paper presented mathematical modeling and implementation in Modelica language of three-phase models for



(a) Comparison between voltage in bus 02 for simulations I, II and III.



(b) Comparison between voltage in bus 11 for simulations IV, V and VI.

Figure 7. Voltages in buses 02 and 11 for corresponding simulation sets, during the last load step.

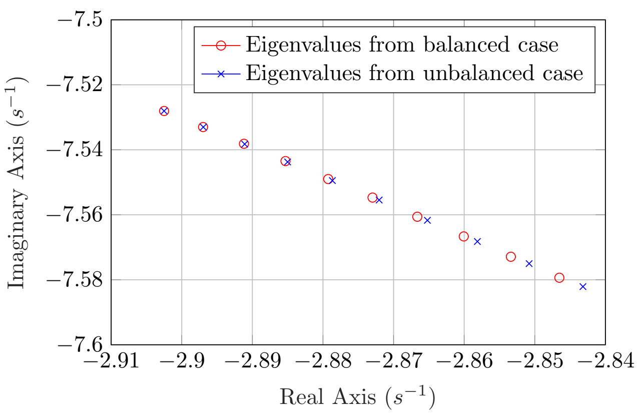


Figure 8. Comparison between specific eigenvalue coming from simulations I and III.

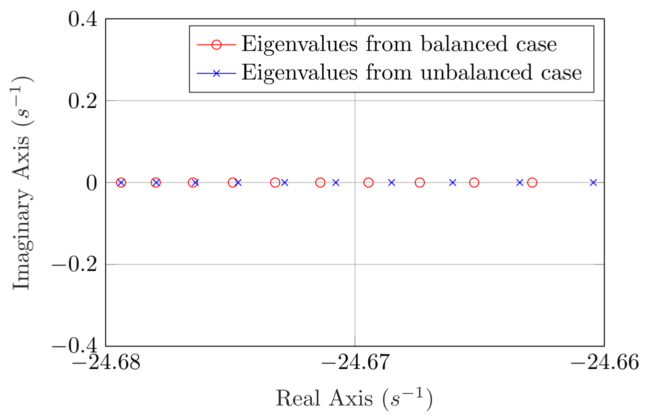


Figure 9. Comparison between specific eigenvalue coming from simulations IV and VI.

transmission lines and a hybrid positive sequence three-phase models for T&D power networks. These models are used to represent a three-phase distribution feeder connected to benchmark IEEE 14-bus transmission system, modeled in its positive sequence equivalent. Six different simulations are realized to compare the different models for the distribution feeder (positive sequence versus three-phase) and to study different conditions to load increase in two specific buses (balanced and unbalanced).

The results from dynamic simulations presented interesting and expected results. Voltage behavior observed using positive sequence models is the same one when the three-phase model had a balanced load increase. Furthermore, results coming from unbalanced load increase show that a positive sequence model might lead to erroneous results of the system dynamic behavior. This fact is corroborated by the linearization analysis performed in this paper. The comparison between eigenvalues coming from the linearized systems shows that, in fact, a positive sequence model for the distribution feeder could lead to wrong conclusions about system stability.

In addition, it is important to highlight that linear analysis is easily conducted in this paper using Modelica-compliant software. Software packages commonly used to study power systems do not necessarily have linear analyses tool, and are encoded separately. This analysis is important specifically to design controllers in the power system. Thus, in this specific matter, the Modelica language and compliant tools may assist in the development of new computational software tools to analyze complex power systems that should emerge in the near future.

The main results of this paper are particularly meaningful, as distributed generation at low-voltage levels is being introduced, especially in the form of photo-voltaic generation; which imminently will give rise to unbalances in distribution networks with single-phase feeders. This type of distribution networks are being referred to as “Active” Distribution Networks (ADNs), which are expected to increase in the near future. While the studies in this paper did not address ADNs specifically, the main result already highlights the importance of unbalances, and naturally, the addition of components with dynamics in distribution feeders will increase this impact. Hence, future work will analyze the impact on power system stability when modeling DG in distribution feeders, and in particular, the impact of their control response. In addition, the study of larger and more complex systems would be of great value and the authors consider it as a future study.

Acknowledgements

This work was supported in whole or in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877, and in part by the CURENT Industry Partnership Program.

References

J. Arrillaga, C. P. Arnold, and B. J. Harker. *Computer Modelling of Electrical Power Systems*, volume 2. Wiley Online Library, 2001.

T. M. L. Assis, G. N. Taranto, D. M. Falcao, and A. Manzoni. Long and short-term dynamic simulations in distribution networks with the presence of distributed generation. In *Power Engineering Society General Meeting, 2006. IEEE*, pages 7–pp. IEEE, 2006.

K. Balasubramaniam and S. Abhyankar. A combined transmission and distribution system co-simulation framework for assessing the impact of volt/var control on transmission system. In *2017 IEEE Power Energy Society General Meeting*, pages 1–5, July 2017. doi:10.1109/PESGM.2017.8274633.

M. Baudette, M. Castro, T. Rabuzin, J. Lavenius, T. Bogodorova, and L. Vanfretti. OpenIPSL: Open-instance power system library—update 1.5 to "iTesla power systems library (iPSL): A modelica library for phasor time-domain simulations". *SoftwareX*, 7:34–36, 2018.

Jens C Boemer, Karsten Burges, Pavel Zolotarev, Joachim Lehner, Patrick Wajant, Markus Fürst, Rainer Brohm, and Thomas Kumm. Overview of german grid issues and retrofit of photovoltaic power plants in germany for the prevention of frequency stability problems in abnormal system conditions of the entso-e region continental europe. In *1st international workshop on integration of solar power into power systems*, volume 24, 2011.

F. Casella, A. Bartolini, S. Pasquini, and L. Bonuglia. Object-oriented modelling and simulation of large-scale electrical power systems using modelica: A first feasibility study. In *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, pages 6298–6304. IEEE, 2016.

F. Casella, A. Leva, and A. Bartolini. Simulation of large grids in OpenModelica: reflections and perspectives. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, number 132, pages 227–233. Linköping University Electronic Press, 2017.

Cepel. ANAREDE. `https://bit.ly/2Mjkd1F`, a.

Cepel. ANATEM. `https://bit.ly/2FzTKw4`, b.

M. de Castro Fernandes, J. G. de Oliveira, L. Vanfretti, M. Baudette, and M. A. Tomim. Modeling and simulation of a hybrid single-phase/three-phase system in modelica. In *2018 Simposio Brasileiro de Sistemas Eletricos (SBSE)*, pages 1–7, May 2018. doi:10.1109/SBSE.2018.8395775.

EATON. Cymdist. `https://bit.ly/2HkxHer`.

ENTSO-E. *Dispersed Generation Impact on CE Region Security*. European Network of Transmission System Operators of Electricity, ENTSO-E, 2014.

EPRI. OpenDSS. `https://bit.ly/2zV4cLB`.

P. A. N. Garcia, J. L. R. Pereira, S. Carneiro, V. M. da Costa, and N. Martins. Three-phase power flow calculations using the current injection method. *IEEE Transactions on Power Systems*, 15(2):508–514, 2000.

IEA. *World Energy Outlook 2017*. Organization for Economic Co-operation and Development, OECD, 2017.

H. Jain, K. Rahimi, A. Tbaileh, R. P. Broadwater, A. K. Jain, and M. Dilek. Integrated transmission & distribution system modeling and analysis: Need & advantages. In *Power and Energy Society General Meeting (PESGM), 2016*, pages 1–5. IEEE, 2016.

W. H. Kersting. *Distribution System Modeling and Analysis*. CRC press, 2001.

S. K. M. Kodsi and C. A. Canizares. Modeling and simulation of IEEE 14-bus system with FACTS controllers. *University of Waterloo, Canada, Tech. Rep*, 2003.

J. M. T. Marinho and G. N. Taranto. A hybrid three-phase single-phase power flow formulation. *IEEE Transactions on Power Systems*, 23(3):1063–1070, 2008.

F. Milano. An open source power system analysis toolbox. *IEEE Transactions on Power Systems*, 20(3):1199–1206, Aug 2005. ISSN 0885-8950. doi:10.1109/TPWRS.2005.851911.

M. Mirz, L. Netze, and A. Monti. A multi-level approach to power system modelica models. In *Control and Modeling for Power Electronics (COMPEL), 2016 IEEE 17th Workshop on*, pages 1–7. IEEE, 2016.

M. A. A. Murad, F. J. Gómez, and L. Vanfretti. Equation-based modeling of three-winding and regulating transformers using modelica. In *2015 IEEE Eindhoven PowerTech*, pages 1–6, June 2015. doi:10.1109/PTC.2015.7232503.

Siemens. PSS/E software. `https://sie.ag/2DpsTR9`.

G. N. Taranto and J. M. T. Marinho. Simulation of integrated transmission and distribution networks with a hybrid three-phase/single-phase formulation. *Interface*, 2, 2017.

W. F. Tinney and C. E. Hart. Power flow solution by newton's method. *IEEE Transactions on Power Apparatus and systems*, (11):1449–1460, 1967.

L. Vanfretti, W. Li, T. Bogodorova, and P. Panciatici. Unambiguous power system dynamic modeling and simulation using modelica tools. In *2013 IEEE Power Energy Society General Meeting*, pages 1–5, July 2013. doi:10.1109/PESMG.2013.6672476.

Luigi Vanfretti, Tetiana Bogodorova, and Maxime Baudette. A modelica power system component library for model validation and parameter identification. In *Proceedings of the 10th International Modelica Conference; Lund; Sweden*, number 96, pages 1195–1203. Linköping University Electronic Press, 2014.

D. Winkler. Electrical power system modelling in modelica–comparing open-source library options. In *Proceedings of the 58th Conference on Simulation and Modelling (SIMS 58) Reykjavik, Iceland, September 25th–27th, 2017*, number 138, pages 263–270. Linköping University Electronic Press, 2017.

Towards Pan-European Power Grid Modelling in Modelica: Design Principles and a Prototype for a Reference Power System Library

Andrea Bartolini[1] Francesco Casella[2] Adrien Guironnet[3]

[1]Dynamica s.r.l., Italy, andrea.bartolini@dynamica-it.com
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy, francesco.casella@polimi.it
[3]RTE, France, adrien.guironnet@rte-france.com

Abstract

This paper presents the PowerGrids library, which is aimed at the modelling of large-scale power transmission and distribution system. The requirements and design principles of the library are extensively discussed, as well as some key implementation details. The library represents a prototype implementation of the presented requirements and design guidelines, and will form the basis for the future development of an industrial-grade open-source library to be used by European TSOs and DSOs.

Keywords: Power System Modelling, Power Generation and Transmission, Pan-European Power System

1 Introduction

Power systems are undergoing major changes due to the increasing penetration of renewable energies, to the booming development of high-voltage direct current lines and to the difficulty to build new AC lines, that lead power system operators to design complex automata to fully use the grid potential. All these changes are deeply affecting the power system dynamics that are evolving from a well-known behaviour, mainly driven by the synchronous generator dynamics, to a more uncertain and complex behaviour driven by a combination of power-electronic based components without inertia, complex system-wide controls and traditional machines.

In this context, the Transmission System Operators (TSO) ability to assess the system stability is questioned and recent events have shown the need for a new frame for time-domain stability studies (ENTSO-E SG SPD Report, a; ENTSO-E System Protection and Dynamics WG Report; Yan et al., 2018). In order to accompany this deep and global change in a secure way, power system actors have to take adapted and coordinated decisions. This demands a transition from current closed tools towards flexible and transparent power system simulation approaches, enabling all players to run collaborative studies in a straightforward and easy way.

Despite previous efforts led by the European Network of Transmission System Operators for Electricity (ENTSO-E) association with the development of a standard exchange format - Common Grid Model Exchange Standard (CGMES) - different software tools still give different results for the same data set, even for small networks (ENTSO-E SG SPD Report, b). Without having access neither to the final modelling implementation nor to the mathematical methods used for the solution, it remains very arduous to understand the reasons of the results difference, and thus to agree upon common actions at the pan-european level. Having open-source shared models will be a first step towards a better understanding of the overall power system behaviour and ease the power system actors cooperation.

Modelica appears as a good candidate to build such models and initiatives at the European level are currently conducted to promote this vision – the necessity to construct and share a reference Modelica library – using the results from several previous efforts (Winkler, 2017). Indeed, the European projects Pegase and iTesla have respectively validated the usability of Modelica for power system modelling on elementary components (Pegase; Chieh et al., 2011) and proved the possibility to get results from Modelica models that are similar to those obtained with existing power system simulation software (iTesla; Bogodorova et al., 2013; León et al., 2015). Due to their intended goal, the library inherited from these projects consists mainly in a direct porting of several tools models that stick to a procedural way of modelling (Fortran-style code), which is difficult to interpret, extend, and maintain.

Two open-source libraries have also been developed in the Modelica community for electromechanical power system modelling: one is the Modelica.Electrical.QuasiStationary library, which uses complex phasors and contains only very basic component models, thus not specifically designed for large scale power system models. The other one is the PowerSystems library, whose design is fundamentally based on the concept of replaceable phase system, which can have an arbitrary number of independent voltage and current components, and an arbitrary number of reference phases. The very high level of abstraction, coupled with an extensive use of inheritance, allows to accommodate a very wide scope in a unified framework spanning DC, one- and three-phase AC in both quasi-static and dynamic regimes. The end result is a code base which is very general, elegant and concise; however, the price to pay for these features is that the code is quite difficult to comprehend for non-Modelica experts. For a more detailed review of open-source Modelica libraries in this field, the interested reader is referred to (Winkler, 2017).

The authors are also aware of the existence of commercial Modelica libraries for power system, and also worked on internal developments in this field that were mainly intended for experimentation purposes, see, e.g., (Casella et al., 2016). However, they won't be discussed here, since a key requirement of the next-generation pan-european power system modelling tool is the accessibility and open-source nature of the modelling code.

All the previously mentioned works contribute to demonstrate the added value of using Modelica for power systems models, but a more generic and high-level reflection, especially on the modelling side, is required to ensure the widest possible adoption of Modelica by the power system actors, i.e. Transmission System and Distribution System Operators. This is the main objective of the prototype library introduced in this paper and could be achieved by relying on the following library features:

- the library architecture was carefully designed to take full advantage of the declarative modelling approach of Modelica, which allows to write models that are close to their textbook descriptions, without at the same time being too difficult to understand for Modelica beginners because of the extensive use of advanced object-oriented features;
- specific power system models have been written in order to be easily understood, reused and potentially extended by power system experts without much experience in the Modelica language, rather than by expert Modelica developers;
- the library has been designed to eventually evolve into an industrial-grade tool.

The paper is organized as follows. Section 2 defines the library scope while section 3 states the necessary requirements to embrace this scope. In section 4, the architecture and design of the PowerGrids library are discussed, showing how they can fulfil the requirements. Section 5 is a presentation of the prototype library models and their validation, both at an individual level and at a system level, is detailed in section 6. Section 7 is devoted to conclusions and future perspectives of this work.

2 Scope

The scope of the library described in this paper is to build electro-mechanical power transmission and distribution system models, possibly spanning entire countries or even the whole pan-european system, though of course smaller system models for regional studies or for didactic purposes should also be covered.

The electro-magnetic behaviour of transmission lines, transformers and loads is assumed to always be close to sinusoidal steady-state, so that the relationships between three-phase voltage and current systems can be represented by phasors, while fast electro-magnetic transients in those components involving current and voltage state variables are neglected.

The library described in this paper currently only considers balanced three-phase systems, though it would be possible to extend the approach to deal with unbalanced systems by introducing the direct, inverse and zero sequence representation. The system dynamic behaviour is generated by the mechanical inertia of rotating synchronous generators and by their internal electro-magnetic transients, represented by lumped-parameter models, which have much larger time constants than the transmission lines, in the range 0.1-10 s, as well as by the dynamics of all continuous-time and event-based control systems.

The library should allow to efficiently simulate dynamic islanding transients, whereby a synchronous system can be split into two or more synchronous islands by the opening of some circuit breakers, which could operate at significantly different frequency for significant amounts of time, and possibly be re-synchronized afterwards.

The system can be assumed to always operate at frequencies close to the reference value (50 Hz for Europe), since power systems cannot operate reliably when the system frequency deviates by more than a few Hz from the reference. Simulations need to be reliably initialized at steady-state or sufficiently close to it also in the case of very large-scale systems.

Last, but not least, the models from this library are expected to be used in two ways. One is to use them to build complete power system models in Modelica, using Modelica tools to turn them into simulation code. However, even though encouraging experiments have been carried out using Modelica tools to handle system models with thousands of generators, lines, and loads, as reported in (Casella et al., 2017), current Modelica compiler technology, based on the expansion of models down to individual scalar equations, still cannot handle realistic pan-european grid models within acceptable limits in terms of code generation time (a few minutes at most) and of executable simulation code size (a few MBytes).

A major breakthrough is thus needed in the Modelica compiler technology to avoid the burden of code duplication relative to components that are instantiated hundreds or thousands of times in the system model. In the meantime, it should be possible to also use models from the library within domain-specific simulation tools, such as RTE's Dynaωo (Guironnet et al., 2018), that require only one instance of C-code for each model appearing multiple times in the system, build the residuals and Jacobians for the DAE solver using said C-code and ad-hoc algorithms, and finally use DAE open-source solvers to compute the simulation results.

3 Requirements

The modelling approach for the physical components represented in the library should be fully declarative and equation-based, with no compromises. The code describing the physical behaviour of components should be as close as possible to the original formulation of the equations that are found in textbooks or technical reports. This allows to achieve four important objectives:

1. the models are largely self-documenting and can easily be traced back to authoritative sources;
2. it is immediately clear to a domain expert by just reading the code what a given model actually represents and what the modelling assumptions are;
3. it is easy to turn any new component model who is devised by experts in terms of basic physical equations on the paper into the model code;
4. it is easy to adapt or customize existing models to fit new and possibly unexpected simulation scenarios in the future.

The model-solver separation principle should always guide the code development. Models should be written to be clear, compact, elegant and easily understood, without any need to explicitly structure them in a way which is oriented to their solution, as it is common practice in traditional Fortran, Matlab, C, or C++ based models.

Basic data types, model building blocks and base classes should be readily available in the library, to guide modellers that are not Modelica experts in the development of new models that fully exploit the power of the Modelica language, avoiding them the hassle of taking care of the tedious and repetitive parts of the modelling effort, and allowing them to focus on the core parts of their models.

For controllers such as Automatic Voltage Regulators (AVR), governors or Power System Stabilizers (PSS), which are usually specified in terms of block diagrams, the same representation should be used in the library, so that again the model is as close as possible to the original source (e.g. block diagrams taken from IEEE standards or recommendations), self-documented and immediately recognizable by a domain expert.

On the other hand, an essential point is that the actual implementation of blocks, either in terms of equations or of lower-level block diagrams, must be fully accessible, so that the exact behaviour of the each block is clearly and unambiguously defined. This is often a weak point in closed-source simulation tools, in which the behaviour of some non-trivial blocks such as anti-windup controllers, that can actually be implemented in subtly different ways, may be different from one tool to the other, leading to different behaviour of the same system model depending on which tool it is run (ENTSO-E SG SPD Report, b).

It is assumed that the results of a power-flow calculation are available, specifying the voltage modulus and phase and the entering active and reactive power flow at each three-phase port of each component. These could be computed by a separate tool, or possibly by a corresponding Modelica power-flow model, see section 5.5.

Last, but not least, it is important to point out who is going to write the code of the more sophisticated component models. Most Modelica libraries are written by Modelica experts and are expected to be used more or less out of the box by domain experts and practitioners. In the case of the library discussed in this paper, instead, one fundamental goal is to develop an open library that is readily accepted by the community of transmission and distribution systems operators, as well as by students in this field, also for developing new models. This requires the existing source code to be easily read and understood, and new models to be easily developed or adapted, by people who are domain experts but are not seasoned Modelica library developers, having had only some basic training in Modelica.

The most commonly used language in this area are Fortran, C/C++ and possibly Matlab or Python, and people are normally used to a procedural approach to modelling, so the shift to the a-causal, declarative approach of Modelica already requires a significant effort to become second nature to the modeller. From this point of view, providing basic data types, objects and templates (base classes), as well as some fully worked out reference model implementations can be very useful to make the learning curve less steep.

Most importantly, advanced Modelica features should then be used judiciously as long as they can actually make reading and writing the code easier, by using basic types and classes which are already defined in the library. On the other hand, very elaborate inheritance-based library architectures, possibly involving replaceable classes and multiple inheritance, should rather be avoided, as they could make understanding the code difficult for people without a lot of experience and practice in using Modelica, ending up in a steep barrier to the acceptance of the library, and ultimately hindering its diffusion in the potential user community.

4 Design of the PowerGrids Library

In this section, the basic architecture of the PowerGrids library is presented in a bottom-up fashion, showing all the basic data types and base classes that can be used to described actual models in a very compact and clear way, as will be demonstrated in Section 5.

4.1 Complex Variables

Following the declarative modelling paradigm, phasors should be represented by complex variables, in order to have a compact and immediately understood formulation of equations, as in textbooks. Complex numbers are available in Modelica since 2013 and should be used to their full extent. The explicit expansion of equations into their real and imaginary parts is tedious, ugly and error prone, makes the model unnecessarily obscure, and should thus be left to the Modelica compiler, not to the modeller.

In fact, Modelica tools still turn out sometimes to be a bit crude in handling Complex numbers in the most efficient way, but this should by no means be a reason to work around these tool limitations by expanding models to scalar values manually. To the contrary, this should be a good reason to push Modelica tool vendors to improve their handling so that there is no performance penalty whatsoever when they are employed instead of writing manually expanded equations using Real numbers.

4.2 Units and Scaling

Physical variables, in particular connector variables, are always defined given in SI units. This guarantees consistency at system and model level, avoids the need of introducing conversion factors in physical equations, and also allows to use unit checking to spot modelling errors resulting in dimensionally inconsistent equations. As it is the standard practice, engineering units can be used for convenience in the user interface, for parameter input and for plotting, by setting the `displayUnit` attributes according to the typical values found in high-voltage transmission systems, i.e., kV for voltage, kA for current, MW for active power, MVA for complex apparent power and MVAR for reactive power.

Use of per-unit variables is restricted to those cases in which their use makes the equations more compact and thus easier to write and understand, such as models of synchronous machines. In this case, the reference textbooks use per-unit quantities when writing the equations, so the application of the principle that the Modelica code should be as close as possible to the textbook equation suggests to also write the Modelica code in the same way, using component-specific (not system-wide) base quantities declared as parameters.

If model equations using SI units were solved directly, the numerical accuracy of the solution would be severely hampered because of badly scaled problems, since the typical model will contain some variables with order of magnitude 1 (the p.u. variables), some others (currents and voltage variables) around 10^3–10^4, and some others (power variables) in the range 10^8–10^9. In fact, one of the reasons why p.u. variables and engineering units are used explicitly in traditional power system simulation software is to ensure the good numerical conditioning of the problem. However, this can make the code a bit confusing and possibly lead to modelling mistakes, due to mix-up of per-unit variables and parameters using different base quantities (system-wide and component-specific) with variables and parameters in engineering units.

The proper way to address this problem in a declarative way in Modelica is to avoid introducing explicitly the scaled variables in the model, to define the nominal power and voltage as parameters of each component, and then to set the `nominal` attribute for all physical variables to a value that can be derived from them, e.g.

```
Modelica.SIunits.Current I(nominal = SNom/VNom);
```

When generating the simulation code, the Modelica tool uses the `nominal` attribute as a scaling factor for the variables, and is also able to automatically derive a scaling factor for equation residuals using the nominal values of the variables and the values of the Jacobian, see (Casella and Braun, 2017) for further details. The end result is equivalent to the use of per-unit variables of traditional power system simulation codes. However, according to the model-solver separation principle, this process is totally transparent to the modeller.

4.3 Connectors and System Object

AC components in power trasmission and distribution systems interact through 3-phase connections, which are assumed to be balanced in the context of this work, since this is the most commonly used assumption for large-scale system studies. The efficient simulation of such systems when they are close to sinusoidal steady-state requires a representation of 3-phase systems leading to almost constant variables in that case.

To achieve this objective, both the Modelica.Electrical.QuasiStationary and the PowerGrids libraries represent 3-phase systems relative to a rotating frame of reference, whose angle is defined by a source component and propagated through connectors to all the components of a synchronous system; the former library uses complex phasors, while the latter uses a dq0 decomposition. In order to handle this properly, overconstrained connectors need to be used, whereby the Modelica tool analyzes the connection graph and automatically removes the redundant equations involving the reference angle that are generated when meshed grids are built.

This design is very elegant and fully object-oriented, but as of Modelica 3.4 it has the fundamental limitation that the topology of the graph is statically determined at compile time and cannot be changed at runtime. It is thus possible to model systems containing multiple synchronous systems (e.g. two AC grids connected by an HVDC line with AC/DC interfaces), but it is not possible to efficiently model systems that are split in two or more disconnected synchronous islands during a simulation, e.g. because of the opening of circuit breakers that connect different areas in the system, as well as their re-synchronization and re-connection. As the scope of the library discussed in this paper should include this kind of simulations, it is not possible to use connectors to propagate the reference frame information.

Therefore, the AC connector in the PowerGrids library is defined as

```
connector TerminalAC
  Types.ComplexVoltage v;
  flow Types.ComplexCurrent i;
end TerminalAC;
```

where `v` is the phase-to ground, RMS voltage phasor, and `i` is the line RMS current phasor. Both phasors describe the magnitude and phase of the three-phase balanced system with respect to a reference rotating frame, that needs to be the same for all connectors in each connection set. There are then three possible options regarding reference systems, which are selected in the system object and then accessed via the inner-outer mechanism.

The first option can be chosen when the system is connected to an infinite bus, which prevents the frequency of all voltages and currents to drift significantly away from 50 Hz, lest synchronism is lost. In this case, all components use the same reference, which rotates at the fixed frequency of 50 Hz (or 60 Hz for US and Japan, the reference frequency being defined in the system object).

The second option can be chosen in the case of a single system of components operating in synchronism, except for possible electro-mechanical swings, whose frequency can drift away from 50 (or 60) Hz for long intervals, e.g., if secondary frequency control is lacking or not effective enough for some reason. In this case the frequency output of one component, usually the angular frequency of a large synchronous machine rotor in the system, is connected to the reference frequency input of the system object, from which all other system components access it via the inner/outer mechanism. System models with multiple synchronous islands (e.g. two AC grids connected by an HVDC line) can be modelled by including each island in a sub-model with its own system object and an outer connector, which will then in turn be connected together with the HVDC line.

The third option allow to handle dynamically splitting and re-synchronizing synchronous sub-islands in the system efficiently. In this case, the system object contains a description of the grid topology (nodes and branches), it receives the connection status of all branch components (lines, transformers) in the grid by means of boolean inputs, and outputs the reference frequency and activation status to all nodes in the system (machines and loads).

During initialization, and every time one such input changes, e.g. because of a line trip, the system object runs a topological analysis on the grid, determines what are the synchronously operating islands, selects one node of each island as the source of the reference frequency following some criterion, and outputs the reference frequency values to all the nodes of the island accordingly. If newly formed islands end up operating steadily at different frequency, all their phasors will be nearly constant, speeding up variable step-size solvers; this would not be the case if the second option was selected, because only the phasors of the island containing the single reference generator would be constant, while all others would end up rotating at constant, non-zero speed.

In case islands are formed that would break the simulation, e.g. because they have loads but no generators, the activation signals of all the nodes become false, so that the corresponding models can be switched to an "off state", e.g. zero current for loads, zero power for generators.

The third option is currently not yet implemented in the library, but is made possible by the library architecture; it was successfully experimented within the feasibility study reported by (Casella et al., 2017). A proper implementation is not trivial, because it needs to include efficient algorithms for topology analysis, that will be run online each time an event corresponding to a breaker opening or closing is triggered. Such algorithms should be implemented efficiently as external C functions, possibly porting them from the code-base of existing power system simulation tools.

Setting up a model to work with the third option requires a fairly large amount of signal connections between node and branch components of the grid and the system object, which is not really consistent with a fully object-oriented modelling approach. On the other hand, large grid models will most likely not be built manually using a GUI, but rather generated automatically from grid description such as the XML-based CIM standard (R.Viruez et al., 2017), so this is not necessarily a big issue. It would be in fact be interesting to extend the overconstrained connector semantics of Modelica to also handle run-time topological changes of the connection graph, in order to pass the reference phase/frequency information through the connectors in fully object-oriented way, but relying on this mechanism would make the library described in this paper depending on a non-standard experimental feature of the language for a very long time, which is out of question. The discussion of such a mechanism for future improvements of Modelica would be nevertheless interesting, but goes beyond the scope of this paper.

DC connections, as will be needed for HVDC links, can be represented by using the standard connectors of the Modelica.Electrical.Analog library.

4.4 Ports and Common Base Classes

AC components are connected via `TerminalAC` connectors, which contain the minimum amount of independent quantities needed to represent the physical interaction of two or more components by means of connection equations, namely, the phase-to-ground and line current RMS phasors. In fact, there are many other related variables that could be used both for modelling and monitoring purposes: the phase-to-phase voltage phasor, the per-unit voltage and current phasors, the voltage and current modulus and phase angle, the active, reactive, and complex power flowing through the connector, etc.

All these quantities and the equations relating them to the phase-to-ground voltage and line current phasors are thus defined once and for all in the `PortAC` model, so they don't need to be re-defined every time a new component model is developed. In fact, since some of these

```
partial model OnePortAC
  parameter Types.Voltage UNom;
  parameter Types.ApparentPower SNom;
  parameter Boolean portVariablesPu = true;
  parameter Boolean portVariablesPhases = false;
  parameter Boolean generatorConvention = false;
  parameter Types.Voltage UStart = UNom;
  parameter Types.Angle UPhaseStart = 0;
  parameter Types.ActivePower PStart = SNom;
  parameter Types.ReactivePower QStart = 0;

  PowerGrids.Interfaces.TerminalAC terminal;
  PortAC port(
    final v = terminal.v, final i = terminal.i,
    final UNom = UNom, final SNom = SNom,
    final portVariablesPu = portVariablesPu,
    ...
    final PStart = PStart,
    final QStart = QStart,
    ...);
  outer Electrical.System system;
end OnePortAC;
```

Figure 1. Code of OnePortAC base class

quantities are not always needed in all models, they are conditionally defined based on boolean parameters. For example, `portVariablesPu` activates the definition of per-unit port variables, as well as the corresponding binding equations.

Power transmission and distribution system models are usually represented by single-line diagrams, whereby components are only connected to the ground internally where needed (e.g. for shunt admittances in transmission lines), but there is no need to show the connections to ground explicitly at the system level. Hence, each port corresponds to a single connector, not to a pair of connectors, as in the case of DC component models.

It is then possible to define two base classes, one for one-port components such as generators, load, and capacitor banks, that correspond to nodes in single-line connection diagrams, and the other for two port-components such as transformers and transmission lines. These base classes contain the terminal connector(s), the instance(s) of the port(s), and the start values of port voltage phase and angle as well as of the active and reactive power flowing into the port, which will be the basis for initialization, see Section 4.5. Fig. 1 shows the implementation of `OnePortAC`, abridged for conciseness.

All one- and two-port AC components can then be written by extending from these two base classes, directly using port variables such as the active power entering the port `port.P`, the per-unit voltage phasor `port.vPu` or the voltage phase angle `port.UPhase` in the model equations.

It is then possible to define the `OnePortACdqPU` base model, which extends `OnePortAC` by adding the equations that define Park's transformation between voltages and currents in the port rotating frame of reference and the direct and quadrature per-unit voltages `vdPu,vqPu` and currents `idPu,iqPu` in the machine rotating frame of reference. This allows to directly write the machine models using per-unit variables in the rotor frame of reference, as it is normally done in textbooks, without bothering about defining Park's transformation, which is already provided by the base class of the library.

This design is extremely user-friendly even for a novice Modelica developer, because it is very straightforward. It provides many of the variables that are usually involved in power system component models, and ultimately allows to focus on writing the actual model in a declarative way, without wasting time on coding standard and well-known basic definitions and transformations times and again.

4.5 Strategies for Initialization

The traditional strategy for initialization in power system simulation tools is to have dedicated initialization models to calculate initial values for the system states based on boundary values. Indeed, TSOs only have observability on their own network, which means that the initial values available are the bus values - voltage, active and reactive flows into the bus - and not the set point values for the different injections (machines, loads, static VAR compensators, etc.). It is then necessary to derive these initial values from the bus values: the causality of this problem is opposite to the one from the time-domain problem in which the fixed set points enable to calculate the bus values. In current power system software, the approach is to manually derive the sequence of computations required to solve the system for the initial values starting from the boundary values.

This strategy, that was also followed in the design of the iPSL library, has the advantage of splitting the system-wide initialization problem into a large number of small initialization problems, which are solved locally for each component connected to the grid, and is historically proven to be reliable also on very large-scale systems. On the other hand, it follows a traditional procedural approach, which is particularly inconvenient because it basically requires to re-write all the model equations two times, one for the simulation and one for the initialization, the only difference being the causality of the solution.

The PowerGrids library implements two declarative initialization strategies, based on the use of initial equations, that can be selected from the system object. The starting point in both cases are the initial values of the voltage magnitude and angle, active and reactive power at the component ports, see Fig. 1. These values can be the result of a power-flow computation carried out with a specialized tool, or they could be computed running a Modelica power-flow model, see Section 5.5, or they could come from the TSO/DSO online grid monitoring system.

A key provision in both cases is the computation of start values for a subset of variables that show up in a nonlinear fashion in all models, e.g., the machine angle,vwhich is the argument of sine and cosine functions, and the direct and quadrature values of rotor current and voltage, which are multiplied together to get the active and reactive power, in synchronous machine models. This is performed in a declarative way by writing as many initial equations as necessary to compute those start values (which are `fixed =false` parameters) from the port start values.

It is important to notice that the number of such equations is a small fraction of the total equations number in the model, and also that it is not necessary to write them down explicitly in the way they will be solved. It is thus easy to check that they are correct by inspection, because they are basically the same that show up in the **`equation`** section, except that they have the start values as unknowns instead of the model variables, and that they lack all derivative terms (quasi-static initialization).

The first strategy replicates the traditional one, albeit in a declarative way. In this case, initial equations fix the currents values that are injected into the grid by the generators, according to the results of the power-flow calculations. The dependency analysis carried out by the Modelica tool on the initialization problem, resulting in the Block-Lower-Triangular (BLT) transformation, determines that, starting from those values, it is possible to solve the network equations, including transformers, transmission lines and loads, to determine the corresponding voltages. The corresponding problem is a very large, sparse system of nonlinear equations; the convergence of the iterative solver is guaranteed by the start values that have previously been computed based on the power flow.

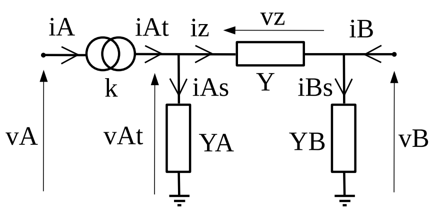
Once that is solved, both currents and voltages are known at the generators' boundaries, so the BLT analysis will automatically determine that the initialization problems of each generator can be solved independently. These will be automatically solved with the opposite causality with respect to the simulation problem, again relying on the computed start values to ensure the convergence of implicit nonlinear equations, ultimately computing all the initial state values, as well as consistent values of the governor and voltage regulators set-points, which are given by `fixed=false` parameters.

The second strategy instead is to directly initialize the whole system in steady-state, by adding initial equations in the generators stating that the derivatives of all internal states are zero. In this case, a very large nonlinear system of equations will emerge from the BLT transformation of the initialization problem, including all the components of the system. Solver convergence can again be facilitated by the start values mentioned previously, together with the use of homotopy to deal with the controller and governor saturations within the generator models, by first solving a problem without controller saturations and then by gradually introducing them, by means of the `homotopy()` operator.

This strategy can be used effectively to ensure a perfect steady-state initialization without the need of any relaxation transient, in case the models used for the power-flow analysis are not exactly consistent with the models used in the Modelica system, e.g. because of simplifications in the loads. It is however hardly feasible in the presence of discrete system-wide automata acting, e.g., on tap changers or phase shifters depending on the values of certain voltages, currents, or power flows, because the solution of the ensuing nonlinear mixed Real-Integer problem could be infeasible.

It is also important to make sure that there are no system-level singularity or ill-conditioning. For example, if the system is not connected to an infinite bus, some primary frequency control needs to be present in the generator models. Otherwise, the steady-state initialization problem will be both over- and under-determined, since on one hand the system frequency will not be well-defined, while on the other hand the excess active power resulting from the mismatch in the balance between the generators' output, the line losses, and the load consumption, will have nowhere to go.

Figure 2. PiNetwork



5 Prototype Library Models

5.1 Transmission Lines and Transformers

The base class `PiNetwork` defines a generic pi-network with one series admittance Y, two shunt admittances at each port Y_A and Y_B, and an ideal transformer with complex tranformatio ratio k at port A, as shown in Fig. 2. It extends `TwoPortAC` by adding the following 7 equations:

```
equation
// Kirchhoff's laws
  iAt = iz + iAs;
  iBs = iz + iB;
  vAt = vz + vB;
// Constitutive equations
  vAt  = k * vA;
  iA   = CM.conj(k)*iAt;
  iAs = YA * vAt;
  iBs = YB * vB;
  iz = Y * vz;
```

From this base model, it is possible to derive many different models through inheritance. A transmission line is obtained by making **`final`** `k = 1` and by setting all the admittances to be equal to parameters, which are kept consant through the simulation; this is a commonly adopted assumption, since the system frequency doesn't change much from the reference value. A transmission line with breakers can instead be obtained by keeping Y, Y_A and Y_B as time-varying variables, whose values are determined by **`when`** clauses depending on the status of boolean breaker opening signals. A fixed ratio transformer is obtained by making Y, Y_A, Y_B, and k equal to parameters, while transformers with tap changers and phase shifters can be modelled by making the real part or the phase of k the result of discrete equations modelling the control logic.

In the latter case, while the description of the physical behaviour is described declaratively in an **equation** section, the control logic is described in an **algorithm** section, which is more suited to describe the control logic and state machine that govern the component's behaviour.

Inheritance is used consistently in this case in order to factor out common features, avoid code duplication and keep the code base lean. However, the design is straightforward also for model developers who are Modelica newbies, since there is only one line of inheritance, whereby each child object adds some specialization to define a more specialized object, whose scope and definition is easy to understand and corresponds to commonly used concepts in power systems.

```
equation
// Flux linkages
lambdadPu = (MdPu+LdPu)*idPu + MdPu*ifPu + MdPu*iDPu;
lambdafPu = MdPu*idPu+(MdPu+LfPu+mrcPu)*ifPu+(MdPu+mrcPu)*iDPu;
lambdaDPu = MdPu*idPu+(MdPu+mrcPu)*ifPu+(MdPu+LDPu+mrcPu)*iDPu;
lambdaqPu = (MqPu+LqPu)*iqPu+MqPu*iQ1Pu+MqPu*iQ2Pu;
lambdaQ1Pu = MqPu*iqPu+(MqPu+LQ1Pu)*iQ1Pu +MqPu*iQ2Pu;
lambdaQ2Pu = MqPu*iqPu+MqPu*iQ1Pu+(MqPu+LQ2Pu)*iQ2Pu;
// Equivalent circuit equations in Park's coordinates
if neglectTransformerTerms then
  udPu = raPu*idPu - omegaPu*lambdaqPu;
  uqPu = raPu*iqPu + omegaPu*lambdadPu;
else
  udPu = raPu*idPu-omegaPu*lambdaqPu+der(lambdadPu)/omegaBase;
  uqPu = raPu*iqPu+omegaPu*lambdadPu+der(lambdaqPu)/omegaBase;
end if;
ufPu =  rfPu *ifPu  + der(lambdafPu)/omegaBase;
0    =  rDPu *iDPu  + der(lambdaDPu)/omegaBase;
0    =  rQ1Pu*iQ1Pu + der(lambdaQ1Pu)/omegaBase;
0    =  rQ2Pu*iQ2Pu + der(lambdaQ2Pu)/omegaBase;
// Mechanical equations
der(theta) = (omegaPu - omegaRefPu) * omegaBase;
2*H*der(omegaPu) =
  (CmPu*PNom/SNom-CePu) - DPu*(omegaPu-omegaRefPu);
CePu = lambdaqPu*idPu - lambdadPu*iqPu;
PePu = CePu*omegaPu;
PmPu = CmPu*omegaPu;
omega = omegaPu*omegaBase;
```

Figure 3. Equation section of the synchronous machine model

5.2 Loads

Load models can be implemented in a straighforward way by extending the `OnePortAC` base class and by adding the equations for the active and reactive power flows. For example, the following model defines a voltage-dependent PQ load:

```
U_URef = port.U / URef;
port.P = PRef*U_URef^alpha;
port.Q = QRef*U_URef^ beta;
```

`PRef` and `QRef` can be determined by parameters, or by time-varying variables equal to an expression that is assigned with a binding equation when instantiating the component, or by an input connector.

5.3 Synchronous Machine

Synchronous machines are a key component of power transmission systems. The PowerGrids library contains a full-fledged 4-windings machine model, to check how far it is possible to push the declarative approach with non-trivial models. Magnetic saturation was not included for lack of time, but it could be added easily to the model.

The corresponding iPSL library model, a port of the Eurostag machine model, is really difficult to understand: the formulation heavily leans towards the procedural, and would require a lots of extra documentation to explain why each equation is written in the way it is. Considering also the initialization model, which is essential in order to use the model, the model spans over 400 lines of code. Once the original equations have been rewritten and partially solved towards their solution to write the code, reverse-engineering them back to their original formulation is conceptually (as well as practically) impossible, because some information was lost in the process. Therefore, porting the iPSL library model into the PowerGrids model was not really possible.

The right approach to write the model is instead to get back to the original sources, in this case the theory manual of the Eurostag software, which unfortunately lacks some crucial details, and eventually to the classic book (Kundur, 1994). The machine model is built by extending from the `OnePortACdqPU` base class, that pre-defines all the variables down to the per-unit voltages and currents on the rotor direct and quadrature axes, and then by just adding the equations taken from the textbook that define the relationship between currents and magnetic field, the relationship between magnetic field, speed, and voltage, and the mechanical power balance on the machine rotor.

The ensuing code, reported Fig. 3 is self-documenting: a reader familiar with synchronous machine theory will immediately recognize the model and understand exactly what kind of behaviour it represents.

A further important point is worth discussing. The equations shown in Fig. 3 use physical parameters (inductances, resistances) which are hardly accessible. External parameters that can be experimentally determined are usually given instead: some per-unit inductances and some time constants. External parameters can be computed explicitly from internal, see (Kundur, 1994) with different degrees of approximation, but not the other way round.

One can then extend the internally-parameterized model, make the internal parameters **final** and with `fixed = false`, add the external parameters, and finally add the initial equations relating the external parameters to the internal ones as stated in the textbook. The Modelica tool will then automatically generate the code to solve those equations backwards, computing the internal parameters from the external one.

The iPSL/Eurostag library model, instead, contains procedural code to carry out this operation, which is difficult to understand unless accompanied by extensive documentation, and a lot more difficult and error-prone to write.

It turns out that these equations are easily solved if the approximate relationships are used, but they can give convergence problem if the more accurate relationships are used. This issue is brilliantly solved by the use of the `homotopy()` operator, whereby the approximate relationships are used for the simplified model and the accurate ones are used for the actual model.

5.4 Controllers

Controllers such as AVRs, PSSs, and governors, are defined in IEEE standards or guidelines, e.g. (IEEE PES-TR1), (IEEE Std 421.5-2016) by means of block diagrams. The `PowerGrids.Controls` library contains the implementation of basic building blocks which are not already available in the Modelica Standard library, either directly by means of equations, or by means of lower-level block diagrams, if this is the way the block behaviour is specified in the original document. The idea as usual is to keep the Modelica representation of component behaviour as close as possible to the original source documents.

A few representative controller models are also implemented from the above-mentioned sources, e.g., the IEEE ST4B Automatic Voltage Regulator, the IEEE PSS2A and PSS2B Power System Stabilizers, and the IEEE TGOV1 turbine governor.

5.5 Power Flow

In case one wants to run a transmission system model without the need of a separate tool to compute the power-flow, one option is to set the power-flow problem up using Modelica, and to solve it with a Modelica tool. The `PowerGrids.Electrical.PowerFlow` package contains models built for this purpose. Some of them, e.g. the transmission line and the infinite bus, extend the regular models by just changing some appropriate default parameter values. Others, like the PV generators, the PQ load, and the slack node, are easily built by extending from `OnePortAC` and simply adding one or two equations to determine the corresponding variables.

This is not meant to be a replacement of existing power-flow tools, particularly for very large-scale models that may include additional outer loops around the inner power flow calculations, such as PV/PQ node switching, distributed slack node or automata simulations. However, the ability of solving smaller power-flow problems, whose results are required to initialize the dynamic models, could be interesting for smaller-size studies, for example by students that do not have access to commercial power flow tools.

6 Model Verification and Validation

Each component of the PowerGrids library has been verified in simple simulations with known analytical solution.

Specifically, the transmission line, transformer, and load models were verified in a number of simple cases with few non-zero parameters, for which the manual computation of voltages and power flows is straightforward.

The synchronous machine model was verified by successfully replicating the results of a worked-out exercise in (Kundur, 1994), checking that all internal parameters are computed correctly, and that the steady-state behaviour corresponds to the values reported in the book. The dynamic behaviour of the synchronous machine was validated in a simple system also including a transformer, a transmission line, and an infinite bus, starting in steady state and applying step changes to the mechanical power input and to the excitation voltage input.

All the above-mentioned test cases were replicated using the iPSL library and collected in the companion library `PowerGridsIPSLValidation`. The results of all tests matched with very good accuracy.

The test system described in (ENTSO-E SG SPD Report, b), which also includes a governor, an automatic voltage reguator, and a power system stabilizer following IEEE standards, was built with the PowerGrids library. All the test results of the report were reproduced satisfactorily.

The testing activity (as well as the library development) was carried out using the OpenModelica tool, using both the standard ODE mode, where the system model is causalized by the tool and then integrated with an ODE solver, and the experimental DAE mode (see (Braun et al., 2017)) that directly solves the DAEs. Note that, although the DAE equations are sparse, the ODE equations are not, due to the instantaneous interaction among all the generators induced by the phasor-based, quasi-static grid model, making DAE mode integration mandatory for systems with more than a few generators.

Finally, the suitability of the library to describe large-scale systems was tested with an ad-hoc scalable test grid model. The model contains a rectangular grid of $N \times M$ nodes, where each node is connected to its four neighbours by equal transmission lines; a synchronous generator is connected via a step-up transformer to each node, and also feeds a local load. The grid is initialized with a trivial power flow, whereby the active and reactive power of each generator is consumed by the local load, so that the grid is unloaded and all the grid nodes have the same voltage.

Tests were carried out successfully with OpenModelica on a Xeon 2650 CPU with 72 GB of RAM, using the IDA DAE solver and the Kinsol sparse nonlinear solver (both using KLU as sparse linear solver), on systems of size up to $N = 64$ and $M = 64$, which correspond to about 4000 nodes, 750.000 differential equations, and one million initial equations. However, the simulation performance above 500 nodes scales quite badly with the size, most likely due to the high number of cache misses caused by the large size of the executable code (over 100 MB). Above this size, Modelica tools capable of exploiting arrays when generating simulation code would be required. Research is currently starting in this area, but no such tool is yet available at the time of this writing.

7 Conclusion and Future Work

This paper lays out the requirements for an electro-mechanical power generation and transmission system modelling library. Such a library could be used by TSOs and DSOs for daily activity up to and including pan-european network stability assessment.

The prototype library PowerGrids presented in this paper was developed according to these requirements using the full power of the Modelica language, namely equation-based and fully declarative models, model-solver separation, complex variables, SI units and clear textbook references, while keeping the source code of library accessible for non-Modelica experts, which is not the case with power system existing libraries. The authors believe that the approach taken in the design of this library maximizes the likelihood that experts in power transmission systems in both industry and academia make the transition to Modelica for their modelling work, avoiding the need of a too steep learning curve that could hinder it.

This prototype library also offers an option for students and academics to work with the same data and models. It provides them a direct and transparent way to the used equations thus facilitates the potential discussions between modelling expert and helps improving the general quality of power system models available.

The PowerGrids library will serve as a basis for a larger and more consequent work to provide a reference library for electro-mechanical and electro-magnetic grid models which will ease a wider acceptance of Modelica in the power system community, particularly for network stability studies. This library development will hopefully be conducted in a Horizon 2020 European research project currently under preparation.

It is planned to release the PowerGrids library as open source during the year 2019.

References

T. Bogodorova, M. Sabate, G. León, L. Vanfretti, M. Halat, J.-B. Heyberger, and P. Panciatici. A modelica power-system library for phasor time-domain simulation. In *Proc. 4th IEEE PES ISGT Europe*, 2013.

Willi Braun, Francesco Casella, and Bernhard Bachmann. Solving large-scale Modelica models: new approaches and experimental results using OpenModelica. In *Proc. 12th International Modelica Conference*, pages 557–563, Prague, Czech Republic, May 15–17 2017. doi:10.3384/ecp17132557.

Francesco Casella and Willi Braun. On the importance of scaling in equation-based modelling. In *8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT 2017*, pages 3–7, Wessling, Germany, Dec 1 2017. doi:10.1145/3158191.3158192.

Francesco Casella, Andrea Bartolini, Simone Pasquini, and Luca Bonuglia. 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*, pages 0–6, Firenze, Italy, Oct. 24-27 2016. IEEE, IEEE. ISBN 978-1-5090-3474-1.

Francesco Casella, Alberto Leva, and Andrea Bartolini. Simulation of large grids in OpenModelica: reflections and perspectives. In *Proc. 12th International Modelica Conference*, pages 227–233, Prague, Czech Republic, 2017. doi:10.3384/ecp17132227.

A. Chieh, P. Panciatici, and J.Picard. Power system modeling in Modelica for time-domain simulation. In *Proc. PowerTech*. IEEE, June 2011.

ENTSO-E SG SPD Report. Analysis of CE inter-area oscillations of 1st december 2016. Technical report, a. URL `https://www.entsoe.eu/publications/system-operations-reports/`.

ENTSO-E SG SPD Report. Documentation on controller tests in test grid configurations. Technical report, b. URL `https://www.entsoe.eu/publications/system-operations-reports/`.

ENTSO-E System Protection and Dynamics WG Report. Oscillation event on 3 december 2017. Technical report. URL `https://www.entsoe.eu/publications/system-operations-reports/`.

A. Guironnet, M. Saugier, S. Petitrenaud, F. Xavier, and P.Panciatici. 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, Oct 21–25 2018.

IEEE PES-TR1. Dynamic models for turbine-governors in power system studies. Technical report, IEEE-PES, 2013.

IEEE Std 421.5-2016. IEEE recommended practice for excitation system models for power system stability studies. Technical report, IEEE, 2016.

iTesla. iTesla: Innovative Tools for Electrical System Security within Large Areas. URL `http://www.itesla-project.eu`.

Prabha Kundur. *Power System Stability and Control*. McGraw-Hill, 1994.

G. León, M. Halat, M. Sabate, J.-B. Heyberger, F.J. Gomez, and L. Vanfretti. Aspects of power system modeling, initialization and simulation using the Modelica language. In *Proc. 2015 IEEE PowerTech*, Eindhoven, The Netherlands, 29 Jun–2 Jul 2015. IEEE.

Pegase. Pegase: Pan european grid advanced simulation and state estimation. URL `http://www.fp7-pegase.com`.

R.Viruez, S. Machado, L.-M. Zamarre no, G. León, F. Beaude, S. Petitrenaud, and J.-B. Heyberger. A modelica-based tool for power system dynamic simulations. In *Proc. 12th International Modelica Conference*, Prague, Czech Republic, May 15–17 2017.

D. Winkler. Electrical power system modelling in Modelica - comparing open-source library options. In *Proc. 58th SIMS*, Reykjavik, Iceland, Sep 25–27 2017.

R. Yan, N. Al-Masood, T. Kumar Saha, F. Bai, and H. Gu. Anatomy of the 2016 south australia blackout: a catastrophic event in a high renewable network. *IEEE Trans. on Power Systems*, 33(5), Sep 2018.

SESSION 6A: BUILDINGS 4

The WaterHub Modules: Material and Energy Flow Analysis of Domestic Hot Water Systems

Bruno Hadengue[1,2] Andreas Scheidegger[1] Eberhard Morgenroth[1,2] Tove A. Larsen[1]

[1]Eawag, Swiss Federal Institute of Aquatic Science and Technology, 8600 Dübendorf, Switzerland
bruno.hadengue@eawag.ch
[2]ETH Zurich, Institute of Environmental Engineering, 8093 Zürich, Switzerland

Abstract

Domestic Hot Water (DHW) systems are large energy consumers in newly-built residential buildings. Mitigation measures involve more efficient hot water appliances and distribution systems, waste heat recovery systems, or changes in consumer habits. However, the implementation of these measures must be investigated carefully, as combinations may lead to unforeseen systemic interactions limiting their potential. In this article, we present tools to identify and optimize these interactions. The WaterHub modules were developed for Material and Energy Flow Analyses (MEFA) of domestic hot water systems. Two modules are available: (i) the WaterHub Modelica library includes models for MEFA system definition, and (ii) The HydroGen Python module provides methods for the stochastic generation of appliance-specific hydrographs, used as input data for the simulation of the system energy and water flows. First, we describe the technical aspects of these modules. Second, we provide an example of how they may be used in a didactic scenario analysis of a heat recovery device.

Keywords: Domestic Hot Water Systems, Material and Energy Flow Analysis, Modelica Library, stochastic demand modeling

1 Introduction

If space heating is historically the largest energy consumer in Swiss households, the share of Domestic Hot Water (DHW) is growing larger as the energy-efficiency of buildings envelopes dramatically increased over the last decades (Meggers and Leibundgut, 2011). In the Netherlands, with a climate similar to Switzerland, Frijns, Hofman, and Nederlof (2013) have shown that 50% of the natural gas demand of newly-built houses is attributed to warm water production.

A broad range of technologies is available to reduce the share of DHW primary energy consumption at household level: improvements regarding hot water production and distribution, measures targeting warm water demand, and wastewater heat recovery strategies may positively impact the system efficiency (Lazarova, Choo, and Cornel, 2012).

However, the energetic system integration of upcoming DHW technologies must be investigated carefully. Sitzenfrei, Hillebrand, and Rauch (2017) highlighted inter-level competition when decentralized heat recovery appliances at shower-level were implemented simultaneously to sewer-level energy recovery facilities. The simulated performance drop of the latter reached in this case 40%. We hypothesize that other system combinations at household level may show similar competitive or synergetic behaviors that shall be identified in order to promote optimal strategies.

In addition to technological interactions, behavioral interactions impact system integration strategies. It is widely recognized that energy and water consumption at household level are strongly influenced by consumer behavior (Pakula and Stamminger, 2015). Nevertheless, the influence of consumption patterns on the systemic energy-efficiency and cost-efficiency of selected technologies is seldom acknowledged. Hendron et al. (2009) were among the first researchers to recognize the issue. Later, Kenway et al. (2012) performed one of the first Material and Energy Flow Analysis (MEFA) of water-related energy flows in households, making use of stochastic demand equations. Although reliable models of water consumption are emerging (Weber et al., 2005; Blokker et al., 2010; Hendron et al., 2010; Penn et al., 2017), many energy-focused investigations of DHW systems still lack the handling of realistic consumption flows.

We hypothesize that investigations of system integration of DHW technologies will be significantly facilitated by the development of a modeling tool fulfilling the following requirements:

1. Flexible and straight-forward definition of complex DHW systems. The modeler should be able to graphically construct technical models, integrating and combining DHW technologies from a library of submodels.
2. The modeling environment should follow the formalism of MEFA approaches as first described by Brunner and Rechberger (2004), to allow for scenario analyses and provide a standard framework for additional comparisons, e.g., Life Cycle Assessments (LCA) or Multi Criteria Decision Analysis (MCDA).
3. In the model, consumer interaction with DHW appliances (showers, taps, dishwasher, etc.) should trigger

upstream (water heaters, distribution systems) and downstream (heat recovery systems, water recycling units) energy and water flows. The model is demand-triggered.

4. The modeler should have full control of water consumption patterns, feeding either (i) stochastic flows or (ii) deterministic flows to single DHW appliances in the technical model.

Tools specifically designed to model DHW systems have emerged within the TRNSYS (Maguire et al., 2011) or the Microsoft .NET environments (Springer et al., 2008), with state-of-the-art treatment of heat losses in pipes. However, these tools do not fulfill all above requirements, as they lack the flexibility of a library of models the modeler can pick from (1), the MEFA formalism (2), and full control over the consumption scenarios to be fed to the model (3).

Modelica, an equation-based, object-oriented programming language, offers high flexibility, hierarchical libraries of models and easy graphical definition of complex systems. Existing Modelica libraries developed for whole building simulation purposes are very powerful and versatile tools that meet (and usually exceed) requirement (1), such as the *Buildings Library* (Wetter, Zuo, and Nouidui, 2014), or more generally IEA Annex 60 based libraries (Christoph et al., 2015). However, these libraries were not developed to follow the MEFA formalism (2), lack the demand-triggered model behavior (3) and full control over domestic hot water flows (4).

Although these libraries may be adapted to meet the above requirements, the purpose of using MEFA formalism (requirement 2) is to provide environmental researchers, already acquainted with similar analyses such as LCA, tools that do not require extensive knowledge in building simulation practices. In this sense, the MEFA requirement allows for significant simplification of the models developed in libraries originally meant for building or district simulations. We thus considered more appropriate to build a dedicated Modelica library rather than downgrading and complementing existing libraries. However, work is conducted to assess the use of the Annex 60 library base interfaces, in order to facilitate potential future integrations. We add that the choice of outsourcing the generation of model water flows to an external Python module was motivated by the complexity of stochastic modeling in Modelica.

We present in this article the WaterHub modules, combining a custom Modelica library for DHW technologies and a Python module for the modeling of demand flows. We describe the implementation and typical use of the tools, and provide an implementation example.

2 The WaterHub Modules

Figure 1 shows a typical workflow using the WaterHub modules for the simulation of DHW system properties, for instance the systemic energy efficiency. We present in this section the MEFA formalism overarching the WaterHub modules and technical descriptions of the modules. The early stage of development of the modules does not yet allow for an open release, however access to the Git repositories will be granted on request.

2.1 MEFA Formalism

Material Flow Analysis (MFA) and its extended version Material and Energy Flow Analysis (MEFA) are described by Brunner and Rechberger (2004) as the “systematic assessment of the flows and stocks of materials [and energy] within a system defined in space and time”. The main idea of MEFA is to quantify all flows of *materials*, i.e. conserved quantities, between *processes* (any transport, transfer, transformation or storage of materials) present in a given system. Energy and mass conservation governs the system:

$$E_{in} - E_{out} = E_{stored} \tag{1}$$

$$M_{in} - M_{out} = M_{stored} \tag{2}$$

This approach has the advantage of clearly separating *flows* (or *fluxes*) and *processes*. A process can hence be as complex as required by the application without impacting the nature of the flows, as it essentially sets the value of its output flows using transfer coefficients on input flows. Moreover, an MFA often form the basis of LCA, as they share part of their formalism.

2.2 WaterHub Modelica Library

The WaterHub Library contains models for the definition of MEFA systems, using a bottom-up modeling approach. Although inspired by the Modelica Standard Library (MSL) *Fluid* library, the models inputs and outputs were simplified to water and energy flows only, in an attempt to stick to the MEFA formalism. The following packages are available:

- **Base Classes**: Defines the *WaterPort* and *HeatPort* connectors, for the water and energy flows, respectively. Water flows are described by a volumetric *flow* in liters per second and a *temperature*. Expressing flows with volume instead of mass units is standard in DHW studies. The specific volumetric heat capacity of water is set to 4179.6 J $l^{-1}K^{-1}$, corresponding to water at 25 °C. Energy flows are expressed in Watts and have no related effort variable.
- **Blocks**: Contains the interface models with the Python HydroGen module (see Section 2.3). Most used is the `Sources.HydrographFromFile` model that connects the output file from HydroGen to appliances from the End-Use package.
- **Appliances**: Technologies at the interface between the water consumer and the DHW system: showers, taps, washing machines, dishwashers, WC, etc. In

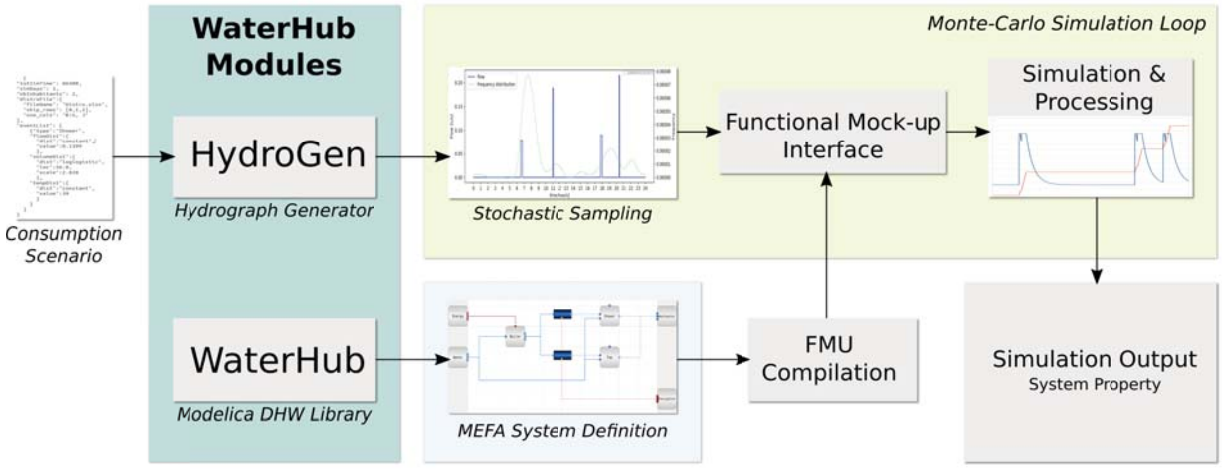


Figure 1. Typical workflow highlighting the use of the WaterHub modules for the simulation of DHW system properties (e.g., energy efficiency, absolute water or energy consumption, etc.). The HydroGen module and WaterHub Modelica library provide the tools for straightforward Monte-Carlo processes.

addition to the energy and water inputs and outputs, these models have a data port, allowing the modeler to link the model flows to an external file containing the flows demanded by the appliance (outputs of the HydroGen module, for instance, see section 2.3).

- **ImportExport**: Includes infinite sources and sinks for imported/exported water and energy flows (flowing in/out of the system). We note here that the flows are not strictly mono-directional, but the existence of sources and sinks suggests a natural flow direction.
- **Pipes and Carriers**: Contains models for water and energy carriers. Note that water pipe models are usually based on first principles, although this is not mandatory.
- **DHW Systems**: Building blocks of DHW systems, e.g., boilers, reservoirs, etc. Includes also water or wastewater-related technologies aiming at recovering resources from wastewater, e.g., wastewater heat exchangers or heat pumps, greywater treatment units, etc. Depending on the needed complexity, models may be based on first principle heat exchanges or emulate steady-state performance. Similarly, additional energy requirements due to pressure losses or pumping subsystems may be modeled with appropriate detail.

2.3 HydroGen Python Module

The module provides methods for the stochastic generation of appliance hydrographs, i.e. time-resolved flow curves. *Events* for a specific appliance are uniquely characterized by (i) a starting time, (ii) a flow rate, (iii) a temperature and (iv) a total event volume. These properties are sampled from distributions defined by the modeler, as described by Scheepers and Jacobs (2014). The modeler provides a *consumption scenario*, a JSON-formatted file containing information for the HydroGen module to generate appliance-resolved domestic water events. The purpose of the following code is to exemplify the JSON structure of a consumption scenario. Here, HydroGen will stochastically generate shower events for a family of five:

```
{
"totSimTime": 86400,
"simDays": 1,
"nbInhabitants": 5,
"distroFile":{
  "fileName": "DistroFile.xlsx",
  "skipRows": [0,1,2],
  "useCols": "B:G, J"
},
"eventList": [
    {"type":"Shower",
    "flowDist":{
      "dist":"loglogistic",
      "loc":0.127,
      "scale":4.158
      },
    "volumeDist":{
      "dist":"loglogistic",
      "loc":55.97,
      "scale":2.828
      },
    "tempDist":{
      "dist":"normal",
      "loc":39,
      "scale":2
      }
    }
  ]
}
```

The following sections describe the variables and parameters present in this exemplary consumption scenario.

2.3.1 Simulation variables

`totSimTime`, `simDays` and `nbInhabitants` are general simulation variables. `totSimTime` defines the time span, in seconds, over which the events should be generated (default is 86400, i.e. 1 day). Note that this number must be consistent with the simulated period in the consequent simulation of the system energy and water flows. `simDays` defines the number of days the flows should be aggregated upon (default is 1). Lastly, `nbInhabitants` defines the number of consumers for the modeled appliance. In the present version of HydroGen, the number of events is sampled from a truncated normal distribution with parameters based on the number of inhabitants. Next step will be to implement a modeler-defined discrete distribution for a full control of the number of events.

2.3.2 distroFile

The `distroFile` key lets HydroGen compute the frequency distributions from an excel or comma-separated values (`.csv`) file. `distroFile` contains the average time-resolved flows (liters per time-step) for each appliance. Flows are converted into frequencies, and the starting time of events is sampled using an inverse transform sampling process as described by Devroye (1986).

2.3.3 eventList

`eventList` describes the type of events to be generated. In the above example, HydroGen will generate "Shower" events. The events types are organized as a list, each component of the list containing four fields:

- `type` is a string and describes the event type. It should correspond to one column of `distroFile`. Types can for instance be shower, WC, kitchen, wash basin, etc.
- `flowDist` is the flow distribution, from which the event flow is sampled.
- `volumeDist` is the total volume distribution, from which the event total volume is sampled.
- `tempDist` is the event temperature distribution, from which the event temperature is sampled.

Each distribution contains a `dist` field describing which distribution should be used for the sampling of the event parameter, and the required parameters describing the distribution shape. The distributions are constructed based on the `NumPy.random` Python package, and the shape parameters are consistent with their NumPy counterparts. In the present version, the modeler can select one of the following distributions: log-logistic, Weibull, Gamma, lognormal, normal, Rayleigh or uniform. The modeler can also set `dist` as `constant`, and the software will return a constant value that overrides the stochastic sampling procedure.

2.3.4 Output

HydroGen2.0 produces a `.csv` file that can be used with models from the WaterHub Appliances package, as described in Section 2.2. The file is formatted to be compatible with the `WaterHub.Blocks.Sources.HydrographFromFile` model, inspired by the `CombiTimeTable` from the MSL. The file contains three columns: (i) time, (ii) demanded flow (liters/second) and (iii) demanded temperature (K). We provide here an example (with default `totSimTime` = 86400 seconds):

```
#1
float FlowTable(86400, 3)
0, 0.1, 311.0
1, 0.1, 311.0
2, 0.0, 0.0
    ...
86399, 0.0, 0.0
86400, 0.0, 0.0
```

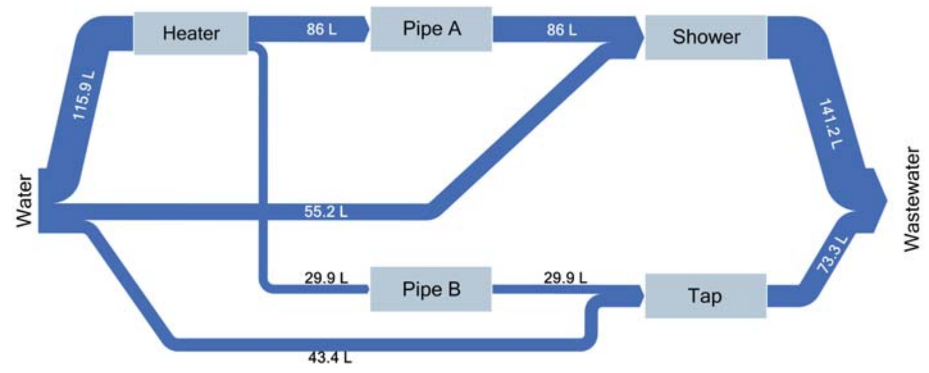
2.4 Typical Workflow

The HydroGen Python module and the WaterHub Modelica library are designed to ease the workflow of DHW systems MEFAs and allow fast scripting of single simulations or Monte-Carlo processes. The workflow is schematically shown in Figure 1. Using the WaterHub Modelica library, the modeler defines the DHW system, including all appliances, sources, sinks and water-related technologies contained in the system following the MEFA formalism. The model is saved as a Modelica file (`.mo`). Within a Python simulation environment, the model is compiled into a Functional Mock-up Unit (FMU). In this work, the JModelica.org platform from Modelon was used (Åkesson et al., 2010). HydroGen methods are used for the stochastic generation of hydrographs, based on the modeler-defined *consumption scenario*, and fed as input for each of the DHW system appliances using the Functional Mock-up Interface (FMI) standard (the open-source PyFMI python package, part of the JModelica.org platform, was used) (Blockwitz et al., 2012). Each Monte-Carlo iteration provides the FMU with a new set of hydrographs through the FMI. Average system properties and their associated distribution are consequently computed, giving insights into the dynamics of the system flows.

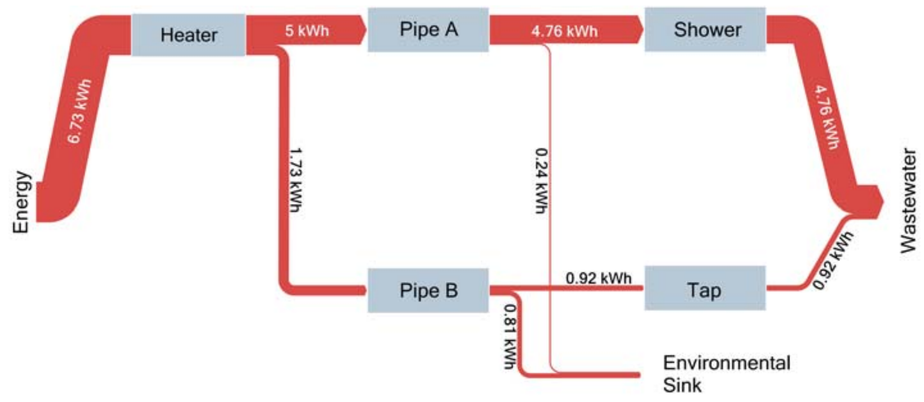
The variability across Monte-Carlo iterations is currently dominated by the user consumption behavior, as the variability intrinsic to the system, for instance system parameters, is considered negligible. Future versions of the modules will account for system variability through the implementation of weather variables and/or cold water temperature profiles, for instance.

3 Example Scenario Analysis

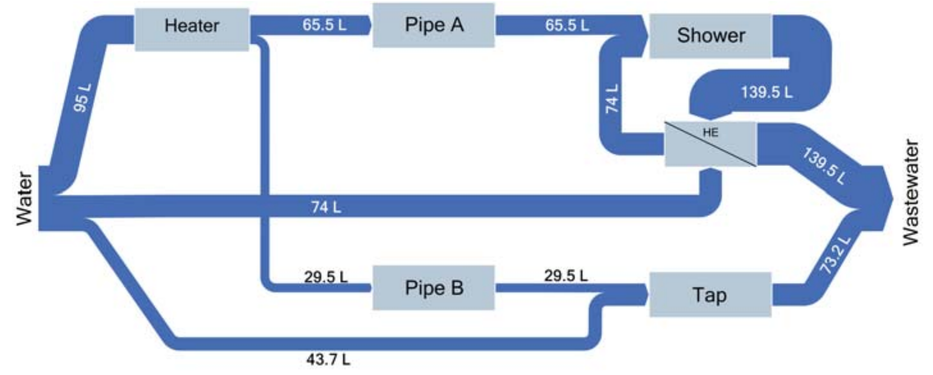
Typical questions about system integration of water technologies are: (i) how has the system performance changed with the implementation of the technology, and (ii) how does the rest of system react, i.e. whether the performance of upstream or downstream technologies is impacted by



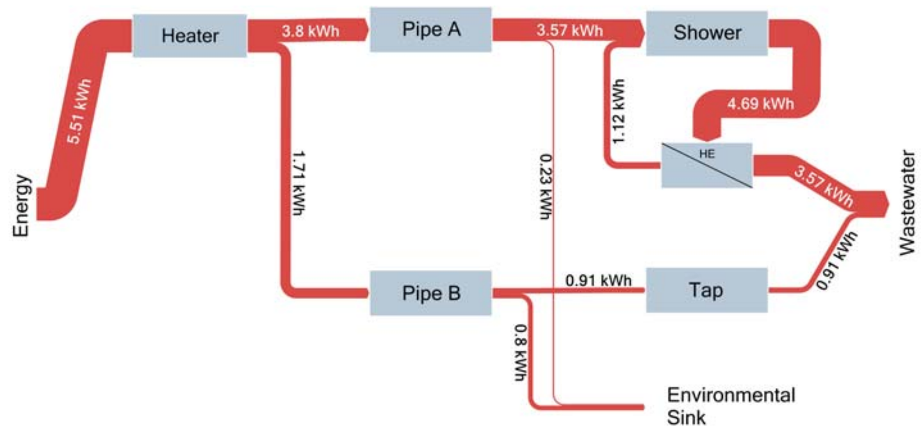
(a) Water flows in *base scenario*.



(b) Energy flows in *base scenario*.



(c) Water flows in *heat recovery scenario*.



(d) Energy flows in *heat recovery scenario*.

Figure 2. Sankey diagrams of DHW systems, daily averages of water and energy flows for the 2000-iteration Monte-Carlo process. MEFA processes are shown as grey boxes, system imports and exports are not outlined.

the implementation. We present here an example scenario analysis. The objective is to provide a usage example of the WaterHub modules for the analysis of a heat exchanger implementation at shower-level. We investigate how the heat exchanger impacts the system energy efficiency and the systemic consequences of its integration.

Two technical scenarios were constructed graphically using the models contained in the WaterHub Modelica library. The *base scenario* includes the following models:

- Sources and sinks (MEFA *imports* and *exports*):
  - Infinite water and energy supply
  - Infinite wastewater sink
  - Infinite energy (or environmental) sink.
- An ideal instantaneous water heater
- Two ideal appliances, a shower and a kitchen tap. Appliance flows are sampled from data generated by Butler et al. (1995); Friedler and Butler (1996); Scheepers and Jacobs (2014)
- Two 15m water pipes, losing energy to the surrounding environment (temperature = 23 °C). The plug-flow modeling of the pipes followed the methodology described by Hanby et al. (2002). The model was validated by computing values similar to experimentally measured overall heat transfer factors in flowing conditions (about 0.62 $\mathrm{Wm^{-1}K^{-1}}$ for a 1/2" copper pipe of type L)(Hiller, 2006).

The *heat recovery scenario* includes all components from the *base scenario*, but the shower appliance is in addition connected to a simple local heat exchanger that preheats incoming cold water (set at 10 °C) with the shower outlet water. The heat exchanger was modeled with a constant steady-state performance curve, using Newton law of cooling and approximating the heat exchange by two thermally connected straight pipes with fluid flows in the same direction. The governing equations read

$$T_1^{out} = T_1^{in} + \frac{\Delta T}{(1 + j_1/j_2)}\alpha \tag{3}$$

$$T_2^{out} = T_2^{in} - \frac{\Delta T}{(1 + j_2/j_1)}\alpha \tag{4}$$

$$\alpha = (1 - e^{\frac{\gamma}{j_1+j_2}L}) \quad , \tag{5}$$

with T_i^X being the temperature of pipe i at position X, ΔT the temperature difference at pipe inlets, and j_i the flow in pipe i. The variable α (Equation 5) describes the efficiency of the heat exchanger, a function of the heat exchange coefficients and exchange surface area (contained in parameter γ), length of pipes L, and flows j_i. In this fictitious example, α is approximated by a constant value. With $\alpha = 0$, no heat is transferred between the pipes, while all the available heat is transferred as $\alpha = 1$. We set $\alpha = 0.7$.

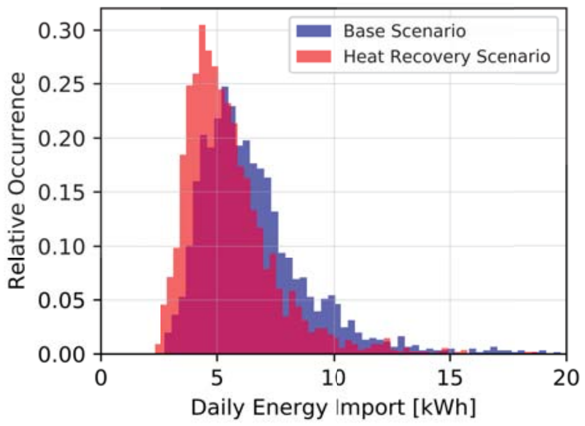


Figure 3. Daily energy imports in the *base scenario* and the *heat recovery scenario*.

The results of a 2000-iterations Monte-Carlo process following the workflow described in Section 2.4 are shown in Figure 2 as Sankey diagrams. The diagrams present the average daily water and energy flows of the *base scenario* and the *heat recovery scenario*. The small water consumption differences are numerical errors due to the limited number of Monte-Carlo iterations and can be neglected in the analysis. Under the assumptions of this didactic example, we recognize how the implementation of the heat exchanger has modified the average flows, increasing the cold/hot water ratio in the shower appliance from 0.64 to 1.13, saving on average 1.22 kWh per day for the heating of hot water (from 6.73 kWh to 5.51 kWh primary energy import). As the heater must now provide a daily average of 95 l of hot water instead of 115.9 l in the *base scenario*, we may suggest that a new dimensioning process is required to optimize the water heater to the new demand. In addition to mean flows, insights into system dynamics may be very important. The frequency analysis of the heater energy demand in Figure 3 shows that the daily energy import distribution is narrower when a heat exchanger is implemented, further indicating that as the heater will operate in a lower, narrower range, the power required by the heater may also be reduced. In addition, we note that the heat exchanger has no impact on the energy lost by the water pipe, indicating the two subsystems are not interacting.

4 Conclusions

The WaterHub modules are dedicated tools for the analysis of water and energy flows at household level, following the Materials and Energy Flow Analysis (MEFA) formalism, thus providing easy-to-use tools for Life Cycle Assessment (LCA) or stand-alone simulations of Domestic Hot Water (DHW) systems.

The WaterHub modules are useful for the identification and analysis of technological and behavioral interactions within DHW systems. In the simple application shown in the example of Section 3, we showed that the addition of appliance-level heat recovery systems may impact the design and dimensions of upstream water heaters.

The WaterHub Modelica library provides the modeler with sub-models for graphical MEFA system definition. The HydroGen Python module provides the modeler with methods for the stochastic generation of appliance-specific hydrographs. The hydrographs are linked to models from the Appliances package included in the WaterHub Modelica library. In this sense, the modeled DHW system is demand-based, triggering upstream and downstream flows by simulating human-appliance interactions. The clear distinction between technical scenarios (built with the WaterHub Modelica library) and consumption scenarios (processed into stochastic flows by the HydroGen Python module) facilitates scenario analyses, as shown in the provided example, by allowing the modeler full control over appliance flows and focusing solely on water and energy flows. The distinction is emphasized by the use of two separate modules: the Python module handles the stochastic modeling and the Modelica environment provides an excellent framework for the simulation of DHW flows.

5 Acknowledgements

This work has been financially supported by InnoSuisse within the SCCER EIP project.

References

J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit. Modeling and optimization with Optimica and JModelica.org - Languages and tools for solving large-scale dynamic optimization problems. *Computers & Chemical Engineering*, 34(11):1737–1749, 2010. doi:10.1016/J.COMPCHEMENG.2009.11.011.

T. Blockwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In *Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany*, pages 173–184, 2012. doi:10.3384/ecp12076173.

E. J. M. Blokker, J. H. G. Vreeburg, and J. C. Van Dijk. Simulating Residential Water Demand with a Stochastic End-Use Model. *Journal of Water Resources and Management*, 136(1):19–26, 2010. doi:10.1061/ASCEWR.1943-5452.0000002.

P. H. Brunner and H. Rechberger. *Practical Handbook of Material Flow Analysis*. Boca Raton, FL : CRC/Lewis, 2004. ISBN 1566706041 9781566706049.

D. Butler, E. Friedler, and K. Gatt. Characterising the Quantity & Quality of Domestic Wastewater Inflows. *Water Science and Technology*, 31(7):13–24, 1995. doi:10.2166/wst.1995.0190.

N.-G. Christoph, M. Wetter, M. Fuchs, P. Grozman, L. Helsen, F. Jorissen, M. Lauster, D. Mülle, D. Picard, P. Sahlin, and M. Thorade. IEA EBC Annex 60 modelica library - An international collaboration to develop a free open-source model library for buildings and community energy systems. In *14th Conference of International Building Performance Simulation Association, BS 2015; Hyderabad; India*, pages 395–402, Hyderabad, 2015. International Building Performance Simulation Association.

L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986.

E. Friedler and D. Butler. Quantifying the Inherent Uncertainty in the Quantity and Quality of Domestic Wastewater. *Water Science and Technology*, 33(2):65–78, 1996. doi:10.2166/wst.1996.0038.

J. Frijns, J. Hofman, and M. Nederlof. The Potential of (Waste)Water as Energy Carrier. *Energy Conversion and Management*, 65:357–363, 2013. doi:10.1016/j.enconman.2012.08.023.

V.I. Hanby, J.A. Wright, D.W Fletcher, and D.N.T Jones. Modeling the Dynamic Response of Conduits. *HVAC & Research*, 8(1):1–12, 2002. doi:10.1080/10789669.2002.10391287.

R. Hendron, J. Burch, M. Hoeschele, and L. Rainer. Potential for Energy Savings Through Residential Hot Water Distribution System Improvements. In *ASME 2009 3rd International Conference on Energy Sustainability, Volume 2*, pages 341–350. ASME, 2009. doi:10.1115/ES2009-90307.

R. Hendron, J. Burch, and G. Barker. Tool for Generating Realistic Residential Hot Water Event Schedules: Preprint. In *Golden, CO: National Renewable Energy Laboratory*, 2010.

C. C. Hiller. Hot Water Distribution System Piping Heat Loss Factors - Phase I: Test Results. In *ASHRAE Transactions*, volume 112, pages 436–446, 2006.

S. J. Kenway, R. Scheidegger, T. A. Larsen, P. Lant, and H. P. Bader. Water-Related Energy in Households: A Model Designed to Understand the Current State and Simulate Possible Measures. *Energy and Buildings*, 58:378–389, 2012. doi:10.1016/j.enbuild.2012.08.035.

V. Lazarova, K. Choo, and P. Cornel. *Water-energy Interactions in Water Reuse*. IWA Publishing, 2012. ISBN 184339541X.

J. Maguire, M. Krarti, and X. Fang. An Analysis Model for Domestic Hot Water Distribution Systems: Preprint. In *5th International Conference on Energy Sustainability and Fuel Cells*, 2011.

F. Meggers and H. Leibundgut. The Potential of Wastewater Heat and Exergy: Decentralized High-Temperature Recovery with a Heat Pump. *Energy and Buildings*, 43(4):879–886, 2011. doi:10.1016/j.enbuild.2010.12.008.

C. Pakula and R. Stamminger. Energy and Water Savings Potential in Automatic Laundry Washing Processes. *Energy Efficiency*, 8:205–222, 2015. doi:10.1007/s12053-014-9288-0.

R. Penn, M. Schütze, M. Gorfine, and E. Friedler. Simulation Method for Stochastic Generation of Domestic Wastewater Discharges and the Effect of Greywater Reuse on Gross Solid Transport. *Urban Water Journal*, 14(8):1–7, 2017. doi:10.1080/1573062X.2017.1279188.

H. M. Scheepers and H. E. Jacobs. Simulating Residential Indoor Water Demand by Means of a Probability Based End-Use Model. *Journal of Water Supply: Research and Technology*, 63(6):476–488, 2014. doi:10.2166/aqua.2014.100.

R. Sitzenfrei, S. Hillebrand, and W. Rauch. Investigating the Interactions of Decentralized and Centralized Wastewater Heat Recovery Systems. *Water Science and Technology 2*, 75(5): 1243–1250, 2017. doi:10.2166/wst.2016.598.

D. Springer, L. Rainer, M. Hoeschele, R. Scott, and R. Solutions. HWSIM: Development and Validation of a Residential Hot Water Distribution System Model. In *ACEEE Summer Study on Energy Efficiency in Buildings*, pages 267–277, 2008.

A. Weber, J. Nipkow, A. Tschui, and M. Poretti. Methode zur Berechnung des Jahresenergieverbrauchs von Warmwasser-anlagen. Technical report, Amstein + Walthert AG, 2005.

M. Wetter, W. Zuo, and T. S. Nouidui. Modelica Buildings Library. *Journal of Building Performance Simulation*, 7(4): 253–270, 2014. doi:10.1080/19401493.2013.765506.

Comparison of a usual heat-transfer-station with a hydraulic modified version under the aspect of exergy saving

Anna Vannahme[1] Tobias Ramm[1] Mathias Ehrenwirth[1] Tobias Schrag[1]

[1]Institute of new Energy Systems, University of Applied Sciences Ingolstadt, Deutschland,
anna.vannahme@thi.de

Abstract

In this paper, a modeling approach for comparing two heat-transfer-stations (HTS) is presented. By comparing a usual HTS with a modified HTS, where the return temperature on primary side of the district heating network (DHN) is used for heating the domestic warm water (DWW), it can be shown that utilizing the return flow of the heating positively contributes to a reduction of temperatures within a DHN and in this way saves exergy. The simulation model is implemented in *Modelica*.

Keywords: heat-transfer-station, hydraulic system, control design, district heating, single-family house

1 Introduction

In order to achieve the German government's aims of covering 60 % of gross final energy consumption by renewable energy sources in 2050 (Bundesministerium für Wirtschaft und Energie, 2018), it is necessary to expand DHN and increase the economic efficiency of existing DHN. Especially in rural areas with a low specific heat demand, the economical operation of DHN is challenging. To cope with these challenges, this paper focusses on non-retrofitted single-family homes as heat consumers, which accounts for a high proportion of houses in rural areas. The return temperatures are a decisive factor for reducing heat losses and improve overall efficiency. The hydraulic setup of a HTS has a great impact on the level of return temperatures (Johansson *et al.*, 2009). For this reason, a measurement based and simulative comprehensive investigation of different HTS will be done. The first step in this comprehensive investigation is to simulate and analyse different HTS. A test rig for space-heating systems in residential buildings is currently under development to carry out measurement based investigations of HTS and to validate the simulation model. Later on, this test rig will be combined with *Modelica* and thus used for *Hardware-in-the-loop* (HiL) simulation.

In the field of measurement based investigations, the *University of Applied Sciences* in Munich has investigated HTS for apartment buildings with focus on DWW heating and lowering return temperatures (Stadtwerke München *et al.*, 2014). Due to associated more complex hydraulic setups, apartment houses are mostly the objects of such investigations (Knierim, 2007; Triesch and Weinmann, 2008; Overhage, 2016). The aqotec GmbH makes a concept proposal for a HTS, where the primary return of DHN from the heating system is used for heating the DWW (aqotec GmbH, 2011). This concept is adopted in this research article and implemented within the simulation model.

In the recent years, modelling of DHN by using *Modelica* increased (Heissler *et al.*, 2016; Fuchs *et al.*, 2013). Also, HiL simulations with *Modelica* are currently conducted (Baltzer *et al.*, 2014; Knorr *et al.*, 2016).

As aforementioned, the first steps are to simulate different HTS. Therefore, a usual HTS with a DWW storage is compared with a modified HTS, where the DWW storage is used for further cooling of the primary return of DHN. The question posed in this article is, whether a slight change in the hydraulic system and control strategy can profitably reduce the return temperatures in periods with space heating. The question will first be answered on an exergetic basis and then by analysing the return temperatures. An investigation based on exergy analysis is quite common (Falk, 2018; Sartor, 2017; Mollenhauer *et al.*, 2016; Jentsch, 2018). Also, a *Modelica* based tool for the dynamic exergy analysis is developed by Sangi *et al.* (2017).

To introduce the key terms of a district heating system, Figure 1 shows a schematic setup of a DHN, which consists of the following components:

- A central heating station, e.g. a combined heat and power unit or biomass heating plant
- One piping for the warm water feed and one piping for the returning water, which flowing back from the consumers

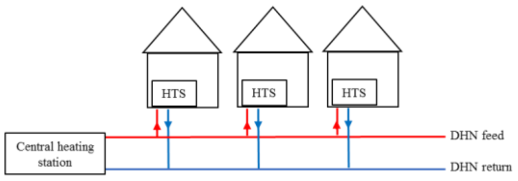


Figure 1. Scheme of a district heating network.

A HTS is the connection between the DHN and the heating system of a building. Usually, the feed temperature of DHN ranges between 70 °C and 130 °C (Dötsch *et al.*, 1998). The heating system of residential buildings has two different load profiles. One is the heating system, (radiator or floor heating system) the other one is the DWW. Table 1 shows the temperature level of each system. Usually, the temperature of the heating circuit is controlled depending on the outdoor temperature. The supply temperature for radiator systems in Table 1 is an orientation value for non-retrofitted buildings. In general, the HTS includes a hydraulic separation between the water in the DHN and the water inside the heating system. In any case, the DWW preparation is separated for hygienic reasons.

Table 1. Supply temperature level of heating systems for residential buildings (Diefenbach *et al.*, 2002; Euroheat & Power, 2008)

Heat	Supply temperature level
Radiator system	70 °C
Floor heating	35 °C
DWW	60 °C

Typically there exist two types of HTS for single-family houses, which can be distinguish in the way of (DWW) preparation (Euroheat & Power, 2008):

- The indirectly connected substation for instantaneous water heaters (see Figure 2)
- Heaters with warm water storage tank (see Figure 3)

The HTS with instantaneous water heater has two parallel plate heat exchangers. In the case of heat demand, the domestic cold water (DCW) gets heated instantaneously. Therefore, a controller on the primary side is necessary, which receives the actual temperature from the secondary side and adjusts the mass flow on the primary side accordingly. There are different possibilities for controlling this mass flow such as motor-controlled or thermostatic controlled valves. In the first case, a controller gets the input from the temperature sensor and modulates a control signal according to the adjusted set point value, in the second case the thermostatically operated valve operates without the supply of auxiliary energy.

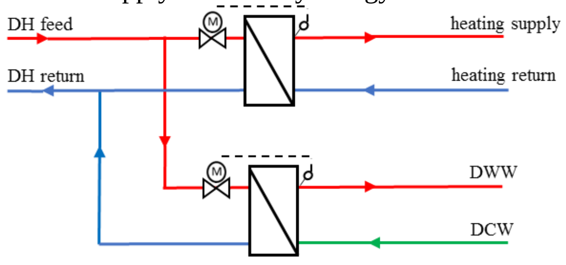


Figure 2. Instantaneous HTS.

There are two ways to design a HTS with a heat storage tank. The storage tank can be connected either on the primary or on the secondary side of the DHN. On the primary side, an external or internal heat exchanger ensures the hydraulic separation from the heat transfer medium of the DHN. Usually, an internal stainless steel pipe is used to charge the DWW storage tank. Furthermore, external heat exchangers may be used to load the heat storage.

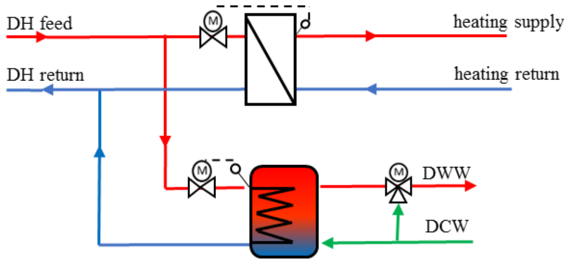


Figure 3. Storage HTS.

While the HTS with an instantaneous water heater requires a high nominal heat flow rate, the HTS with storage has the disadvantage of high return temperatures, unless a special control algorithm for the stratification of the heat storage is implemented. The hygiene is the particular advantage of instantaneous HTS. Storing warm water below 60 °C increases the formation of Legionella. Therefore, a daily heating up above 60 °C is also recommended for single-family houses (Deutsche Vereinigung des Gas- und Wasserfaches, 2004). Correlating the DWW demand with an appropriate design of the heat storage ensures the daily exchange of water and thus a hygienic drinkable water.

The integration of a heat storage tank opens various opportunities such as the absorption of the heat from the return of the heating system on the primary side.

Therefore, a modified system of the HTS with storage tank, hereinafter referred to as "Advanced Storage System" (see Figure 4), is proposed. Compared to the conventional setup of a HTS with storage tank, hereinafter called "Basic Storage System" (see Figure 3), the advanced configuration is expanded by a three-way-valve (2) (see Figure 4). By means of valves (1) and (2) the charging of the heat storage by a *Direct Digital Controller* (DDC) is controlled. If the temperature inside the heat storage is too low for satisfying the DWW demand, the storage tank is charged from the DHN feed, valve (1) opens and the output (B) of valve (2) is closed (see Figure 4). The storage tank is charged when a defined temperature is reached in the top layer.

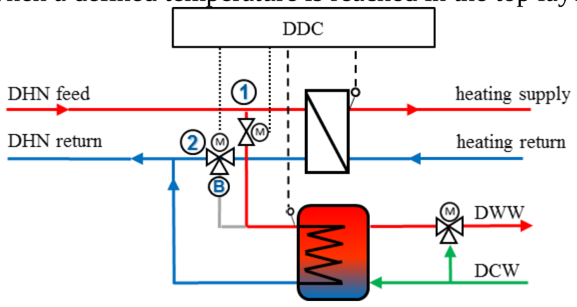


Figure 4. Modified storage HTS charging from DHN feed (case 1).

If the heat storage tank is charged in the top layer, the lower layers may still be colder than the heating return. This allows transporting the returning mass flow of the heat exchanger through the heat storage tank to cool down further the returning mass flow (see Figure 5). In this case, valve (1) closes and valve (2) opens output (B).

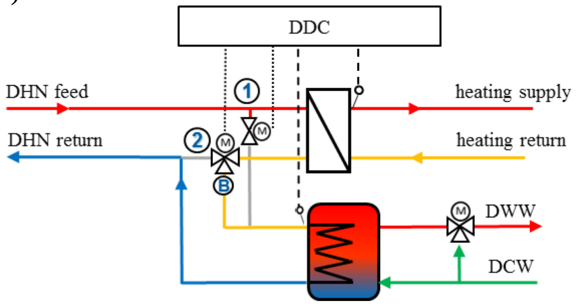


Figure 5. Modified storage HTS in case of return cooling (case 2).

If the lower layers of the heat storage tank have reached the temperature of the heating return, the heating return is routed directly to the return of the network (Figure 6). In this case, valve (1) closes and valve (2) closes output (B).

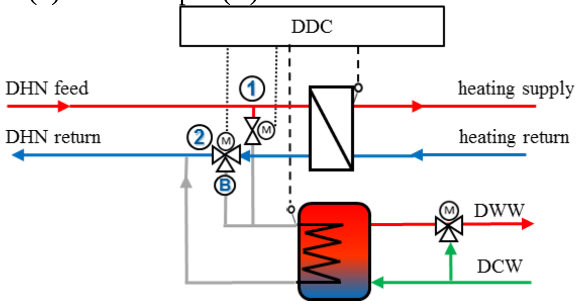


Figure 6. Modified storage HTS in case of fully charged storage (case 3).

2 Methodology

For the simulation of the HTS with subsystems like buildings and heating networks in future work, a modular, acausal approach with *Modelica* language in combination with the software tool *Dymola* is chosen. The modular approach enables to modify subsystems without the necessity to change other parts of the modeled energy system. The specific acausal modeling approach of *Modelica* is useful to simulate the HTS later on in a bidirectional way.

For a first estimation of the exergetic saving, a simulation model of both the conventional as well as the more complex HTS with storage tank was proposed. This enables a fair comparison of both systems under the same boundary conditions. The HTS can be tested regarding their function even under extreme conditions, which can be rarely observe in reality.

Figure 7 illustrates a simplified scheme of the modeled HTS. To emulate the DHN, an ideal source and sink with constant pressure as supply and return are used. The return path of heating and DWW is in each case a boundary with a defined temperature and mass flow rate, which emulates the respective heat demand. The value of the mass flow rate comes from a connected table. The tables contain the heat demand profile according to VDI 4655 (2008). All boundary models are taken from the *Modelica Standard library (Modelica Association, 2017)*. On a first estimation, heat losses of the pipes and the heat storage are neglected.

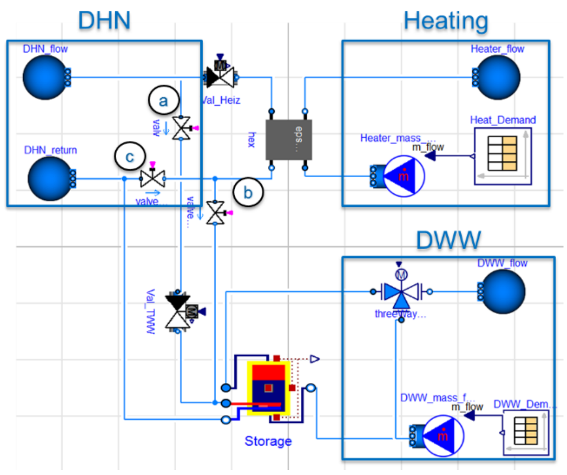


Figure 7. Simplified model of HTS, with return cooling. DHN source and sink (upper left), Heating supply and return (upper right), DWW supply and return (below right), return using control-valve a to c.

The heat exchanger with constant effectiveness is taken from the *Modelica Buildings library* (Wetter *et al.*, 2014). The effectiveness of the heat exchanger is well-designed with a temperature range of 3 K.

A stratified storage with an internal heat exchanger from buildings library is chosen to simulate the heat storage (Wetter *et al.*, 2014). The control strategy for charging the heat storage tank in the “Advanced Storage System” is shown in Figure 8. The first condition inside the storage tank is that the upper third part should be charged between 50 °C and 60 °C. If this isn’t the case, the storage tank is charged by the DHN feed by opening valve (a) and closing valve (b) until it reaches 60 °C in the top layer of the ten modeled layers. If the third layer (counted from top) drops below 50 °C, the upper third of storage is recharged, to maintain a sufficient drinking water temperature at the outlet of the heat storage greater or equal 50 °C (DIN 1988-200, 2012). If the upper third of storage ranges between 50 °C and 60 °C, the lower part of the heat storage is used for cooling down the returning mass flow of the DHN. Therefore, the temperature within the layers one to seven must be lower than the return from the heater. In this case, valve (b) (see Figure 7) opens and valves (a) and (c) are closed. The storage is completely charged, if the lowermost layer’s temperature equals the temperature of the heating return. At this moment valve (c) opens and the other two close. This process is controlled by

discrete valves, which are opened or closed via the conditions in Table 2.
The comparison of the HTS takes place by ensuring that the "Basic Storage System" has the same state of charge within the storage layers than the "Advanced Storage System". The systems will be compared by the used exergy, the course of the return temperatures and by viewing the quantity of necessary loads from DHN feed.

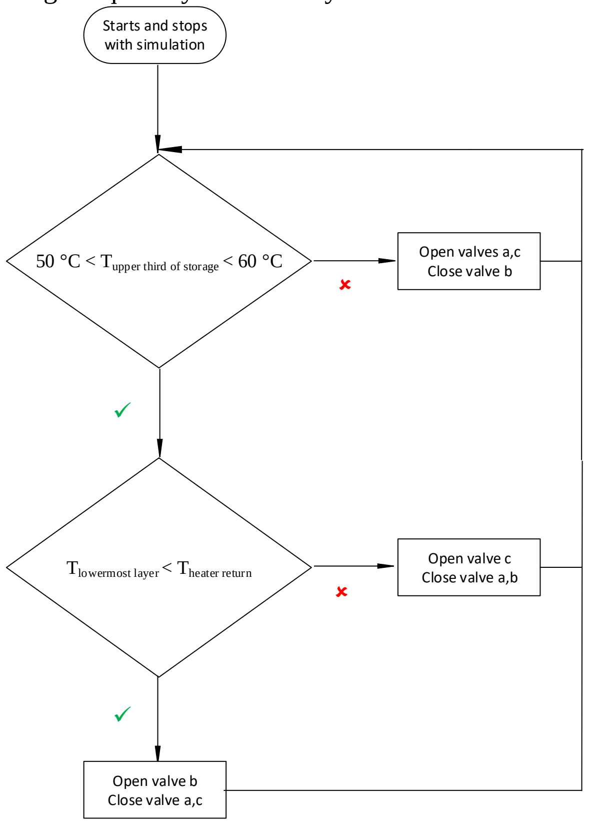

Starts and stops with simulation
50 °C < T upper third of storage < 60 °C
Open valves a,c Close valve b
✗
✓
T lowermost layer < T heater return
Open valve c Close valve a,b
✗
✓
Open valve b Close valve a,c

Figure 8. Flow chart for the control strategy.

Table 2. Logic table for control the three cases described in Figure 4 to Figure 6.

	Conditions inside the storage		
	$50°C<T_{upper\ third\ of\ storage}<60°C$	$T_{lowermost\ layer}<T_{heater\ return}$	Open valve(s)
Case 1	False	True	a,c
Case 2	True	True	b
Case 3	True	False	c

3 Modeling

The HTS connects the DHN with the buildings heating demand. In this investigation, an ideal source and sink with constant pressures and a constant DHN supply temperature emulate the DHN. The building's overall demand consists of both, the heating and DWW demand. These demands are calculated from the reference load profiles given by VDI 4655 (2008). The demand profiles are passed as mass flow rate with one minute.

The heating system has a supply temperature of 70 °C and a return of 55 °C, which is typical for non-retrofitted buildings (Diefenbach et al., 2002). To ensure that the mass flow rate on the primary side of the heat exchanger is high enough to satisfy the heat demand on the secondary side, a valve is controlled by a PID controller, which reading the actual state temperature of the supply heating circuit and checking, if this value reaches the set point temperature. If the temperature is below 70 °C, the valve opens further and if the temperature is higher than 70 °C, the other way round.

The DWW storage is charged via an internal stainless steel pipe. A hysteresis controller is implemented to control the discrete valve (a), which opens for charging the storage from the DHN feed and closes if a defined temperature is reached. In case 2 (see Figure 5) the return flow of the heating process passes through the internal heat exchanger to charge the storage until the lowest layer has an equal temperature. Subsequently the discrete valve (b) is closed by the controller so that the return flow of the heating process flows directly into the DHN return.

On the secondary side of the storage, an outlet temperature of at least 50 °C is required, as mentioned above. To avoid scalding, the set point temperature for the DWW is chosen 43 °C oriented on the recommendation DIN EN 806-2 (2005). Therefore a three-way-valve from the *IDEAS library* (Jorissen *et al.*, 2018) is implemented, which mixes the ten-degree DCW (Baumgarten *et al.*, 2014) with the warm water leaving, to mix the DWW temperature of 43 °C.

4 Simulation

The simulation model was fed with the parameters according to Table 3. The parameters for the DHN, represent a small network, which found often in rural areas. The pressure difference depends on the number of consumers and the extension of the DHN. Thus, a pressure difference of one bar is assumed (FairEnergie, 2015). The heating requirements are based on a non-retrofitted four person single-family house with 140 m² floor area and a specific heating demand of 100 kWh/m²a (Dott *et al.*, 2013). The DWW heat demand is base on a specific value of 500 kWh/a per person stated in VDI 4655 (2008). The DWW storage is usually designed for 30 l to 40 l per person and day, so that once a day the content is fully consumed and refilled. For an optimal heat transfer, the internal stainless steel pipe is designed over the whole height of the storage tank.

The control was implemented as shown in the flow chart in Figure 8. A verification of the "Advanced Storage System" is given by analysing the correct switching of the valves (a) to (c) (see Figure 7). The correct function of the PID controller is checked by observe the actual and set point temperatures. At a feed temperature of 80 °C the return temperatures ranging between 55 °C and 60 °C for the both systems is plausible. The validation should be done on a laboratory test rig, which is outlined in detail below.

Table 3: Simulation parameters for the DHN, HTS and control.

Model component	**Parameter**	**Value**
District heating	Feed temperature	80 °C
	Pressure source/sink	5,4 bar/4,4 bar
Heat exchanger	Effectiveness	0,9
Heating	Supply temperature	70 °C
	Return temperature	55 °C
DWW	Desired temperature	43 °C
DWW Storage	Volume	150 l
	Height	1,4 m
	T_initial	20 °C
Storage charge Controller	Charge temperature	60 °C
	Discharge temperature	50 °C

4.1 Results

The results were analysed for a period of ten winter days based on exergy. Exergy is the part of the total energy of a system that can do work when placed in the thermodynamic equilibrium with its environment.

The determination of the exergy E is based on the exergy factor and the integrated heat flow (1). To compare the used exergy correctly, the status of both heat storages has to be determined after simulation.

Amount of used exergy E:

$$E = f_{Exg} \cdot \int_{t_1}^{t_2} \dot{Q}\ dt \tag{1}$$

where:

f_{Exg}: Exergy factor

$\dot{Q}$: Heat flow in kW

The computation of the exergy factor is done using the equation (2), because the temperatures in the period under review are changing. The equation is based on the approach proposed by Bargel (2010):

$$f_{Exg} = 1 - \frac{T_{amb}}{T_{flow} - T_{return}} \cdot ln(\frac{T_{flow}}{T_{return}}) \tag{2}$$

where:

T_{amb}: Ambient temperature = 293.15 K

T_{flow}: Flow temperature in K

T_{return}: Return temperature in K

The saved exergy with the “Advanced Storage System” in comparison to the „Basic Storage System” correlates to a saving of 83% with respect to the DWW demand. This finding matches the observation of the quantity of necessary charging cycles from DHN feed (see Figure 9). In other words, almost the whole exergy needed for DWW can be covered by the heating return.

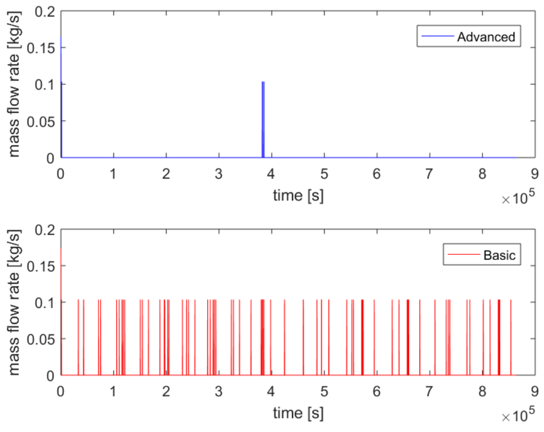


Figure 9. Quantity of charging from DHN feed.

Figure 9 depicts that the “Advanced Storage System” is loaded only three times by DHN feed. The first time at the beginning (as the storage temperature has a initial temperature of 20 °C), the second and third time in quick succession during a high DWW demand.

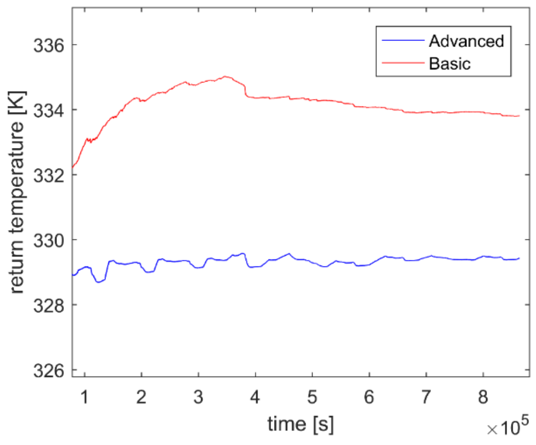


Figure 10. Comparison of averaged return temperatures. The average is weighted by the mass flow rate.

Figure 10 represents that the return temperature of the “Advanced Storage System” is approximately 5 °C lower compared to the return temperature of the “Basis System”. As the mass flow rates are temporarily different, the return temperatures are calculated like written in equation (4). That is why the course of the return temperatures at Figure 10 becomes more and more stable.

$$T_{return,n} = \frac{\sum_i^n (T_{return,i} \cdot \dot{m}_{return,i})}{\sum_i^n \dot{m}_{return,i}} \tag{4}$$

where:

n:	current simulation step
i:	1 to n, with step = 1
$T_{return,n}$:	Return temperature at current simulation step in K
$T_{return,i}$:	Return temperature at simulation step i in K
$\dot{m}_{return,i}$:	Mass flow rate at simulation step i in K

5 Discussion

As mention above, the results hold for non-retrofitted buildings with a high heating demand and a high supply temperature for the heating system. The results indicate a significant saving of exergy, because in case of DHN it cumulates. A reduced mass flow rate drawn from DHN is also an advantage of the "Advance Storage System" and it is the reason for the decreased need of exergy. Furthermore, this result is achieved for a well-designed heat exchanger with a temperature difference of 3 K. If the heat exchanger has a lower effectiveness, the return temperature has more than 58 °C, so that the maximum storage tank temperature of 60 °C could be reached solely by return cooling. In this case, more exergy from the heating return is available. For a more realistic estimation of the advantages of the "Advanced Storage System", the models should be simulated by using a heat exchanger with a realistic, non-constant effectiveness.

6 Outlook

One of the next steps will be the examination of the system behavior and system efficiency in case of a higher resolution of the input data, e.g. 10 s, and therefore a more realistic representation of the DWW demand. Furthermore, both the heat losses of the storage and a realistic model of the pipes should be included.
The objective to decrease return temperature by means of HTS will be pursued as follows:
First, more HTS will be investigated, e.g. a HTS with DWW storage tank in comparison with a HTS with instantaneous DWW heat exchanger application. A second example is given by the HTS shown in Figure 11. A HTS with storage offers the opportunity to integrate a decentralised heat source in a low temperature district heating network (LTDHN), which have usually feed temperatures of 40°C. Thus, it should be investigated whether second heat sources such as a heating rod connect with a photovoltaic array, solar thermal collectors or a stove could be of use for the realization of a LTDHN. This HTS, consisting of a storage with a volume of 1 m³ for single-family houses, is charged by LTDHN. Therefore, the solar thermal system raises the temperature for the DWW demand. The thermal stratification system provides a temperature-dependent stratification of the heating return.

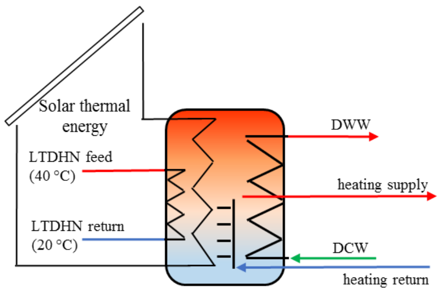


Figure 11. HTS with storage for heating and DWW

After the simulative investigation of different HTS, a measurement based investigation will be performed on a currently developed test rig. As a result, the simulation models of the HTS will be validated. Later on, this test rig for space-heating systems in residential buildings will be extended to a HiL simulation as shown by Baltzer *et al.* (2014). The software simulation of buildings will also be embedded into the HiL simulation. This is one reason to pursue the concept of a digital twin, e.g. done for the Rooftop building in the field of building and equipment simulation by Nytsch-Geusen *et al.*, for a residential building. This allows a testing of the HTS under realistic boundary conditions and alternating user behavior.

References

aqotec GmbH (2011), "Ratgeber zur Optimierung der Sekundäranlage beim Fernwärmeabnehmer", Leitfaden 2011.

Baltzer, S., Lichius, T., Gissing, J., Jeck, P., Eckstein, L. and Küfen, J. (2014), "Hardware In The Loop Simulation with Modelica - A Design Tool for Thermal Management Systems", in *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden, March 10-12, 2014*, Linköping University Electronic Press, pp. 401–408. doi:10.3384/ecp14096401.

Bargel, S. (2010), "Entwicklung eines exergiebasierten Analysemodells zum umfassenden Technologievergleich von Wärmeversorgungssystemen unter Berücksichtigung des Einflusses einer veränderlichen Außentemperatur", Dissertation, Fakultät für Maschinenbau, Ruhr-Universität Bochum, Bochum, 2010.

Baumgarten, C., Rechenberg, J., Richter, S., Chorus, I., Vigelahn, L. and Schmoll, O. (2014), *Wassersparen in Privathaushalten: sinnvoll, ausgereizt, übertrieben?: Fakten, Hintergründe, Empfehlungen*, Umweltbundesamt, Dessau-Roßlau.

Bundesministerium für Wirtschaft und Energie (BMWi) (2018), "Sechster Monitoring-Bericht zur Energiewende. Die Energie der Zukunft", Berichtsjahr 2016.

Deutsche Vereinigung des Gas- und Wasserfaches (2004), *Trinkwassererwärmungs- und Trinkwasserleitungsanlagen: Technische Maßnahmen zur Verminderung des Legionellenwachstums; Planung, Errichtung, Betrieb und Sanierung von Trinkwasser-Installationen, DVGW-Regelwerk / Technische Regel - Arbeitsblatt W*, Vol. 551, April 2004, Wirtschafts- u.

Verlagsges. Gas und Wasser mbH; DVGW Deutscher Verein des Gas- und Wasserfaches e.V, Bonn.

Diefenbach, N., Loga, T., Born, R., Großklos, M. and Herbert, C. (2002), “Energetische Kenngrößen für Heizungsanlagen im Bestand. Eine Untersuchung im Auftrag des Ingenieurbüros für energieeffiziente Gebäudetechnik Ventecs, Bremen”.

DIN 1988-200 (2012), *Technische Regeln für Trinkwasser-Installationen <TRWI>: Technische Regel des DVGW, Deutsche Norm*, Vol. 1988,200, Stand: Mai 2012, Beuth, Berlin.

DIN EN 806-2 (2005), *Planung, Deutsche Norm: EN*, 806-2, Deutsche Fassung EN 806-2: 2005, Beuth Verlag GmbH, Berlin.

Dötsch, C., Taschenberger, J. and Schönberg, I. (1998), *Leitfaden Nahwärme*, Stuttgart.

Dott, B.R., Haller, M.Y., Ruschenburg, J., Ochs, F. and Bony, J. (2013), “The Reference Framework for System Simulations of the IEA SHC Task 44/HPP Annex 38. Part B: Buildings and Space Heat Load”, A technical report of subtask C. Report C1 Part B. doi:10.13140/2.1.2221.4727.

Euroheat & Power (2008), “Euroheat & Power. Guidelines for District Heating Substations”.

FairEnergie (2015), “Technische Anschlussbedingungen Fernwärme Heizwasser (TAB-HW) für den Anschluss an die Fernwärmenetze der FairEnergie GmbH”.

Falk, P.M. (2018), “Evaluation of district heating systems based on exergy analysis. Bewertung von Nahwärmenetzen basierend auf Exergieanalyse”, Fachbereich Maschinenbau, Technische Universität Darmstadt, 2018.

Fuchs, M., Dixius, T., Teichmann, J., Lauster, M., Streblow, R. and Müller, D. (2013), “EVALUATION OF INTERATCTIONS BETWEEN BUILDINGS AND DISTRICT HEATING NETWORKS”, *13th Conference of International Building Performance Simulation Association, Chambéry, France, August 26-28*, pp. 96–103.

Heissler, K.M., Franke, L., Nemeth, I. and Auer, T. (2016), “Modeling low temperature district heating networks for the utilization of local energy potentials”, *Bauphysik*, Vol. 38 No. 6, pp. 372–377. doi:10.1002/bapi.201610038.

Jentsch, A. (2018), “Kalte Nahwärme als Zukunftsoption? Exergie-basierter Vergleich von Nahwärmenetzen”, *BWK: Das Energiemagazin*, No. 70/6, pp. 12–13.

Johansson, P.-O., Lauenburg, P. and Wollerstrand, J. (2009), *IMPROVED COOLING OF DISTRICT HEATING WATER IN SUBSTATONS BY USING ALTERNATIVE CONNECTION SCHEMES, 22nd International Conference on Efficiency, Cost, Optimization Simulation and Environmental Impact of Energy Systems, August 31-September 3, 2009, Foz do Iguacu, Brazil*, Sweden.

Jorissen, F., Reynders, G., Baetens, R., Picard, D., Saelens, D. and Helsen, L. (2018), “Implementation and verification of the IDEAS building energy simulation library”, *Journal of Building Performance Simulation*, Vol. 11 No. 6, pp. 669–688. doi:10.1080/19401493.2018.1428361.

Knierim, R. (2007), “Rücklauftemperatur: Ungehobener Schatz für Versorger und Kunden. Weitere Erlöse aus ungenutzter Wärmeenergie.”, *EuroHeat&Power*, Vol. 2007 No. 3, 56-62, 64-65.

Knorr, M., Schinke, L., Beyer, M., Seifert, J., Mehrfeld, P. Nürenberg, M. Huchtemann, K., Müller, D. and Riedesser, F. (2016), “INSTATIONÄRE ENERGETISCHE BEWERTUNG VON WÄRMEPUMPEN- UND MIKRO-KWK-SYSTEMEN - SIMULATION UND EMULATION”, TU Dresden, RWTH Aachen, Universität Stuttgart, Dresden, Aachen, Stuttgart, 2016.

Modelica Association (2017), “Modelica - A Unified Object-Oriented Language for Systems Modeling. Language Specification”, Version 3.4.

Mollenhauer, E., Christidis, A. and Tsatsaronis, G. (2016), “Evaluation of an energy- and exergy-based generic modeling approach of combined heat and power plants”, *International Journal of Energy and Environmental Engineering*, Vol. 7 No. 2, pp. 167–176. doi:10.1007/s40095-016-0204-6.

Nytsch-Geusen, C., Kaul, W. and Kharaz, S., “DER DIGITALE ZWILLING IN DER ENERGETISCHEN GEBÄUDE- UND ANLAGENSIMULATION”, in *BauSIM2018 - 7. Deutsch-Österreichische IBPSA-Konferenz 26.-28. September 2018,*, pp. 311–318. doi:10.5445/IR/1000085743.

Overhage, A. (2016), “Maßnahmen zur Erreichung niedriger Rücklauftemperaturen. Einflussmöglichkeiten von Nutzern und Energieversorgungsunternehmen”, *AGFW Seminar: Maßnahmen zur Erreichung niedriger Rücklauftemperaturen*.

Sangi, R., Müller, D., Thamm, A. and Jahangiri, P. (2017), “Dynamic exergy analysis - Modelica-based tool development: A case study of CHP district heating in Bottrop, Germany”, *Thermal science and engineering progress TSEP*, Vol. 4 No. RWTH-2018-221637. doi:10.1016/j.tsep.2017.10.008.

Sartor, K. (2017), “Simulation Models to Size and Retrofit District Heating Systems”, *Energies*, Vol. 10 No. 12, p. 2027. doi:10.3390/en10122027.

Stadtwerke München, Hochschule für Angewandte Wissenschaften München and Ebert-Ingenieure (2014), “Breitenanwendung von Niedertemperatur-Systemen als Garanten für eine nachhaltige Wärmeversorgung. LowEx-Systeme Abschlussbericht zum Forschungsvorhaben: LowEx-Fernwärme-Systeme im Rahmen des Förderkonzeptes EnEff:Wärme - Forschung für energieeffiziente Wärme- und Kältenetze”. doi:10.2314/GBV:856917435.

Triesch, F. and Weinmann, E. (2008), “Nutzen innovativer Anschlussanlagen für den Fernwärmekunden. Reduzierung der Rücklauftemperatur.”, *Euroheat & Power*, No. 4, 78-80,82,84-88,90.

VDI 4655 (2008), *Referenzlastprofile von Ein- und Mehrfamilienhäusern für den Einsatz von KWK-Anlagen, VDI-Richtlinien*, Vol. 4655, Mai 2008, Beuth, Berlin.

Wetter, M., Zuo, W., Nouidui, T.S. and Pang, X. (2014), “Modelica Buildings library”, *Journal of Building Performance Simulation*, Vol. 7 No. 4, pp. 253–270. doi:10.1080/19401493.2013.765506.

Evaluating the Resilience of Energy Supply Systems at the Example of a Single Family Dwelling Heating System

Anne Senkel Carsten Bode Gerhard Schmitz

Institute of Engineering Thermodynamics, Hamburg University of Technology, Denickestr. 17, 21073 Hamburg, Germany, {anne.senkel, c.bode, schmitz}@tuhh.de

Abstract

Since the 1980s, the term "resilience" occurs more and more frequently in energy system analysis. However, the consideration and definition of the term primarily occurs in a qualitative way. This papers introduces a quantitative method to assess an energy system's resilience by using physical key figures to reflect the maintained functionality during and after a disturbance.

The presented method is used to evaluate the resilience of a heating system of a single family dwelling when its pump or boiler fails. It can be shown that the introduced resilience index mirrors the drop of the system's functionality and is also able to point out weak spots and the most efficient system improvements. This provides the foundation of comparing the resilience of complex systems even though it is important to pay attention to comparable assessment settings.

Keywords: Resilience, Energy Supply Systems Assessment, Heating

1 Introduction

By publishing its special report on the impact of global warming of 1.5 °C, the IPCC stressed the necessity to decarbonize the energy sector in the medium-term (Allen *et al.*, 2018). On the other hand, already today the increasing integration of renewable energies in the electricity sector leads to rising numbers of interventions and adaptions of the operational planning of power plants to avoid overloading power line sections. In Germany, these measures lead to the "redispatch" of 20.438 GWh in 2017, compared to 4.956 GWh in 2012 (Bundesnetzagentur, 2018). Therefore, not only costs and efficiency, but also the security and resilience of the energy system have to be taken into account while moving towards a sustainable energy supply.

The term resilience originates from the Latin word "resilire" (jumping back, rebounding, returning) and was already used in the 17th century to describe physical counter-reactions (Gößling-Reisemann, Hellige and Thier, 2018). Later, it also became established in psychology, sociology and ecology. However, a universal definition of resilience has not been established yet. In general, the definition by Holling (1973) which defines resilience as a "measure of the persistence of systems and of their ability to absorb change and disturbance", is used widely.

Caused by the discussions about safety of nuclear power plants and climate change, resilience increasingly occurs in the energy sector since the 1980s. Since then, several research projects have focused on defining resilience in the context of energy systems.

A general overview of the historical development of the resilience term and guidelines for designing a resilient system are given by Gößling-Reisemann, Hellige and Thier (2018). Fichter *et al.* (2010) include vulnerability analysis as analytical category in their resilience considerations. Resilience itself is used as a normative category and characterized by introducing system structures that increase resilience. Several ways of qualitative analysis are also provided by (Thoma, 2014) who approaches resilience as a holistic concept including technological, social and economic aspects. Further qualitative assessments of resilience were conducted by Roege *et al.* (2014), Molyneaux *et al.* (2012) and Madni and Jackson (2009).

A quantitative evaluation method was presented by Cimellaro *et al.* (2009) who used a functionality curve to evaluate the dimensions of resilience rapidity, redundancy, robustness and resourcefulness for earthquake disasters. Francis and Bekera (2014) extend this approach by implementing the fragility of the system. A similar approach was conducted by Nan and Sansavini (2017) who additionally considered the performance loss. To assess the resilience of energy systems in this work, the approaches of Francis and Bekera (2014) and Nan and Sansavini (2017) are adapted and the resulting definition presented in the following section.

2 Definition of the Resilience Index

To be able to evaluate the resilience of an energy system, it is necessary to define a key figure that reflects the performance of the system. Nan and Sansavini (2017) call this the "measure of performance" (MOP) while Francis and Bekera (2014) use the term "performance level". Since these values are fictional quantities, this approach needs to be adapted for the assessment of the results of dynamic simulations. Therefore, a physical value needs to be found. For the electricity sector, this could be for example the grid frequency. For the heating sector, the supply or the room temperature is more suitable.

Each of these physical key figures x is defined by a set point x_{set} and a tolerance band $[x_{\mathrm{min}}, x_{\mathrm{max}}]$ in which deviations of x are tolerated. In Figure 1, a characteristic plot of such a key figure is depicted.

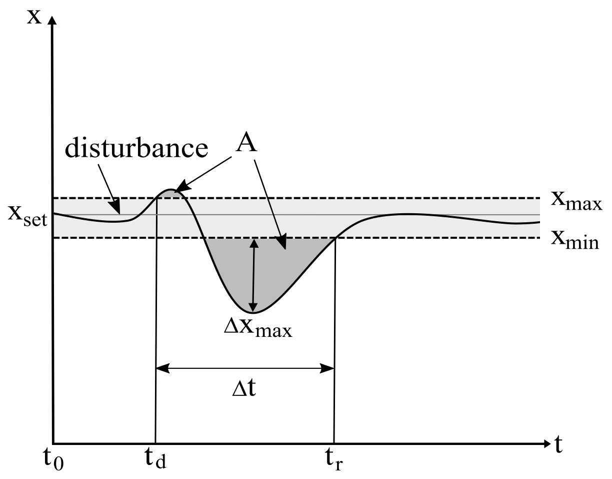


Figure 1. Characteristic course of a key figure in a disturbed system.

Following the definitions of Francis and Bekera (2014) and Nan and Sansavini (2017), the system's *restorative*, *absorptive* and *adaptive capacity* can be evaluated by assessing the progression of this key figure.

For the *restorative capacity*, the *Recovery Time RT* of a system will be computed as,

$$RT = \frac{t_{\mathrm{r}} - t_{\mathrm{d}}}{\Delta t_{\mathrm{norm}}} \tag{1}$$

with t_{d} as the point in time when the key figure leaves the tolerance band for the first time, and t_{r} as the point in time when the key figure returns into the tolerance band and remains permanently within it. While Nan and Sansavini (2017) divide this time period by the deviation of the key figure, this paper introduces Δt_{norm} as normalization time to achieve dimensionless values.

For the *absorptive capacity*, Francis and Bekera (2014) and Nan and Sansavini (2017) both propose to examine the *Maximum Deviation MD* of the key figure, Δx_{max}. Since a tolerance band was introduced, this approach is adapted to only consider the deviation of x from the minimum or maximum value of the tolerance band (c.f. Figure 1). Therefore, Δx is defined as,

$$\Delta x = \begin{cases} x - x_{\mathrm{max}} & \text{if } x \geq x_{\mathrm{max}} \\ 0 & \text{if } x_{\mathrm{min}} < x < x_{\mathrm{max}} \\ x_{\mathrm{min}} - x & \text{if } x \leq x_{\mathrm{min}} \end{cases} \tag{2}$$

which leads to $\Delta x = 0$ for deviations within the tolerance band so that normal fluctuations are not punished. The maximum of Δx is defined as Δx_{max} and used in the following considerations. While Francis and Bekera (2014) divide this value with the set point x_{set}, Nan and Sansavini (2017) use no weight factor but weight their MOP at the beginning resulting in MOPs between 0 and 1. For the introduced physical key figures, it is therefore necessary to define a normalization deviation Δx_{norm} as well,

$$MD = \frac{\Delta x_{\mathrm{max}}}{\Delta x_{\mathrm{norm}}} \tag{3}$$

Additionally, Nan and Sansavini (2017) introduce the *Performance Loss PL* as an indicator of the *absorptive capacity*,

$$PL = \frac{\int_{t_{\mathrm{d}}}^{t_{\mathrm{r}}} \Delta x \, dt}{A_{\mathrm{norm}}} \tag{4}$$

which basically represents the area between the actual course of the key figure and its set point. This figure is also adapted to fit to the introduction of the tolerance band and therefore defined as area between tolerance band and the course of the key figure (depicted as A in Figure 1). When using this definition in combination with physical values, a weight factor A_{norm} will be necessary as well.

Finally, both papers introduce a *Recovery Ability RA* to evaluate the system's *adaptive capacity*,

$$RA = \frac{x(t_{\mathrm{r}})}{x(t_0)} \tag{5}$$

where a higher system's functionality after the disturbance $x(t_{\mathrm{r}})$ than before $x(t_0)$ indicates that the system learned from the incidence and adapted to it. However, this index is not suitable when examining physical key figures. For example, a higher net frequency would not indicate a better system's performance but rather further instabilities.

By combining the recovery time, the maximum deviation and the performance loss, an *Irresilience Index IRI* is introduced,

$$IRI = RT \cdot MD \cdot PL \tag{6}$$

and with this a *Resilience Index RI* is defined,

$$RI = \frac{1}{1 + IRI} \tag{7}$$

where $RI = 0$ represents a completely irresilient system and $RI = 1$ a resilient system. This general definition can be applied to any energy supply system (electricity, heat, gas, etc.) as long as an appropriate key figure for which a tolerance band can be defined, is used.

3 Case Study of a Single Family Dwelling

In the following section the introduced resilience index will be applied to a heating system of a single family dwelling (SFD).

3.1 Model Structure

The model of the heating system was built using Modelica® (Modelica Association, 2019) in Dymola (Dassault Systèmes, 2018) using the TransiEnt Library (Andresen *et al.*, 2015; Hamburg University of Technology, 2017) and the ClaRa Library (Brunnemann *et al.*, 2012; Hamburg University of Technology, TLK-Thermo GmbH, XRG Simulation, 2012) for the components of the heating system. The heating demand of the single family dwelling is modeled using the Buildings Library (Wetter *et al.*, 2014). Therefore the heat exchange with the environment is considered at a low resolution by aggregating the heat transfer through the walls and windows, the heat capacity and the heat gains and losses through solar irradiance, ventilation and internal sources each in one instance (Senkel, 2017). The icon layer of the heating system model is depicted in Figure 2.

In the heating system, the heat is produced by a gas boiler and transferred to the building through a heat exchanger. The produced heat flow of the boiler is set to obtain the supply temperature given by the heating curve of the system. To adjust the heat transferred to the building, a thermostat is integrated that varies its opening in order to obtain a room temperature of 22 °C (according to EN 15251 (European Commitee for Standardization, 2007)). The installed pump regulates its mass flow to keep a constant pressure loss resulting in a smaller mass flow when the thermostat is closing in. Furthermore an expansion vessel is integrated to balance the pressure in the system. The most important parameters of the system are collected in Table 1.

The following results were simulated with the solver Dassl in a 15-minutes resolution with a tolerance of 0.0001.

Table 1. Parameters of the heating system.

Parameter	*Value*
Set room temperature	22 °C
Nominal supply/return temperature	60/40 °C
Nominal mass flow	0.2 kg/s
Nominal pressure	1 bar
Nominal heat flow of gas boiler	7 kW
Heat transfer coefficient of the building envelope	0.4 W/(m^{2}K)
Heat transfer coefficient of the windows	3 W/(m^{2}K)
Floor area	100 m²
Window area	16 m²
Minimal outdoor temperature	-12 °C

3.2 Scenarios

First, the shown system is simulated with the weather data of Hamburg in the period between January 30th and February 2nd, 2012 (Lange, 2014). This time period was selected due to the occurring low outside temperatures which lead to an enhancement of the considered effects. According to EN 15251 (European Commitee for Standardization, 2007), an operational temperature of 22 °C within a tolerance band of ±2 K is recommended for ambient temperatures below 16 °C. Figure 3 shows

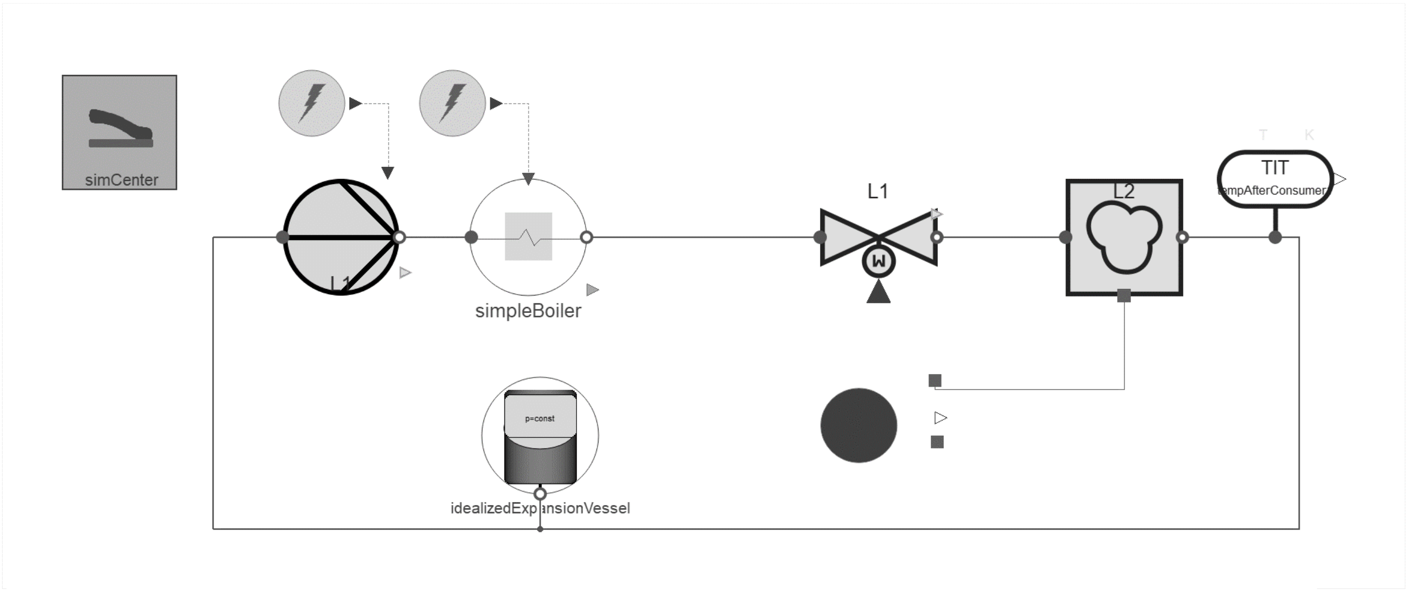


Figure 2. Model structure of Reference System.

that the system is able to provide this service offering as long as it is undisturbed. In this case the temperature only slightly varies from its set point due to fluctuations in the outside temperature and solar irradiance.

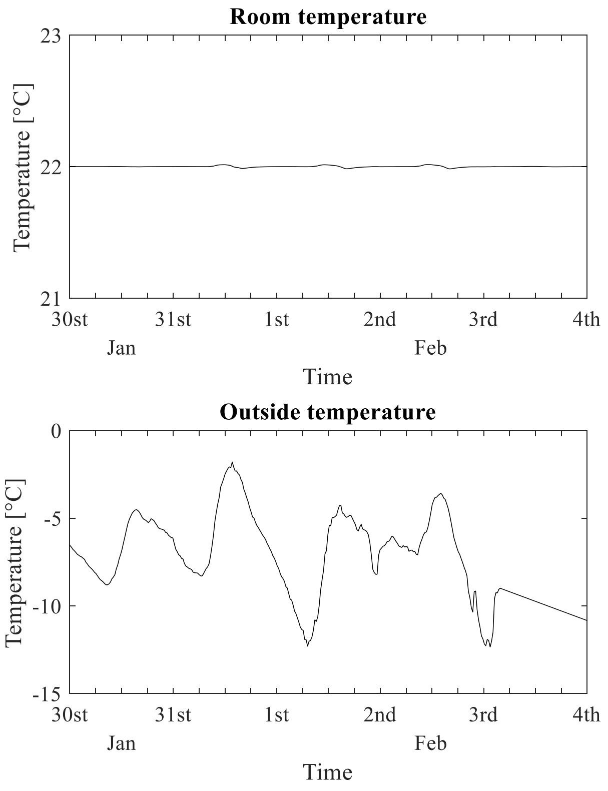


Figure 3. Room temperature and outside temperature without disturbance.

The first considered improvement (System 1) is to lower the house's heat losses by installing windows with a lower heat transfer coefficient (k_{win}=1.3 W/(m^{2}K)).

The second improvement (System 2) involves a change of the heating system by installing a hot water storage vessel (V_{stor} = 0.6 m³) as shown in Figure 4. This also leads to changes in the control of the heating system. Therefore, the heat flow of the boiler is regulated to obtain a temperature of 60 °C in the upper part of the hot water storage. The mass flow in the boiler circuit is regulated by a pump which adapts its power to obtain a maximum boiler outlet temperature of 60 °C. In the consumer circuit, a mixing valve is integrated to mix cool return water with the hot water of the storage in order to meet the supply temperature given by the heating curve. The control of the pump in the consumer circuit and the thermostat are left untouched.

In the following simulations, the systems are disturbed by a failure of the boiler or the supply pump (see also Figure 2 and Figure 4, indicated by the lightning symbol). Both failures occur in the first twelve hours of February, 1st.

3.2.1 Scenario Consumer Pump Failure

When the pump shuts down, no hot water flows through the heat exchanger. Hence, no heat can be transferred to the building and the room temperature of the reference systems drops drastically due to the cold outside temperature (Figure 5).

When looking at System 1 during the pump failure, one can observe that due to the improved insulation the temperature drops less than in the reference case. Furthermore, it becomes obvious, that the system recovers faster than in the reference case since the heat loss to the surroundings is lower, resulting in a shorter reheating phase.

For the failure of the consumer pump, System 2 shows a similar temperature drop as in the reference case since the house itself remains without heat supply and the heat losses to the surroundings are due to the same insulation as high as in the reference case. After the disturbance, the boiler inlet temperature is higher

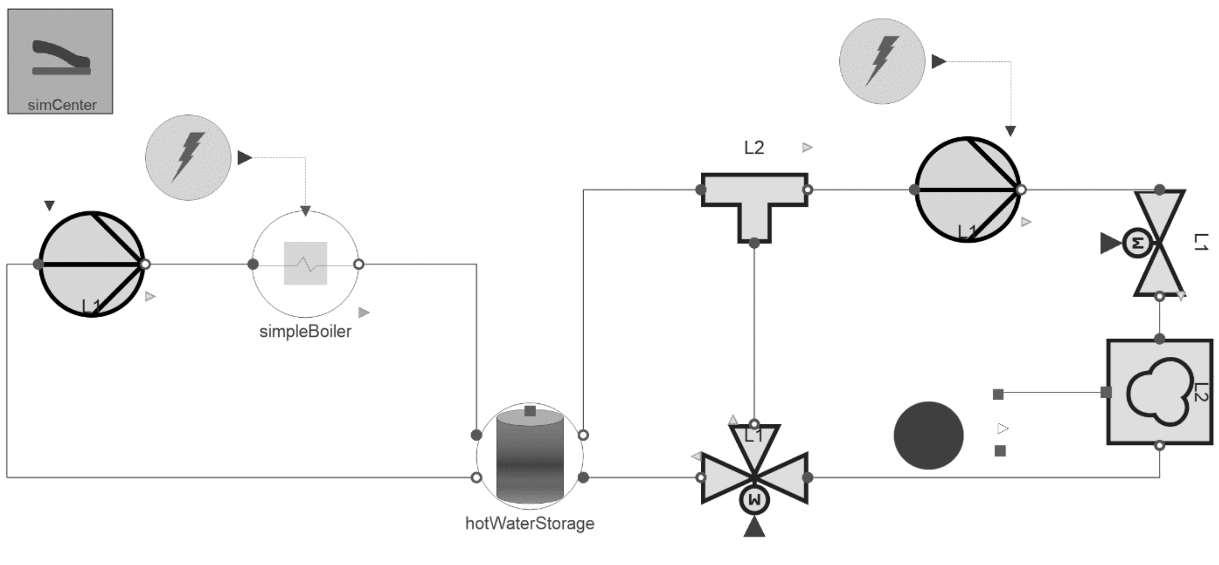


Figure 4. Model structure of System 2.

since the boiler is fed from the hot water storage in which the cold return water is mixed with the warmer water stored at the bottom of the tank. Additionally, the supply water is mixed with hotter water in the top of the storage vessel. Both effects lead to higher supply temperatures in System 2 than in the reference system which is why the set point of the room temperature is reached faster in this system.

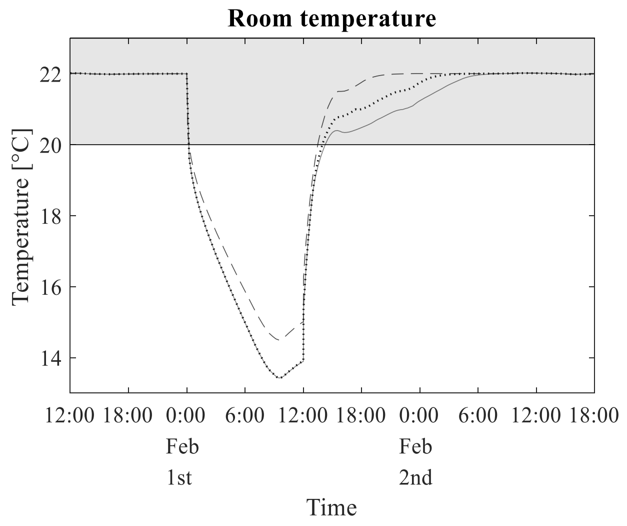


Figure 5. Temperature profiles during consumer pump failure (shaded: tolerance band, solid: Reference System, dashed: System 1, dotted: System 2).

3.2.2 Scenario Boiler Failure

The failure of the boiler in the reference system results in a less drastic temperature drop (Figure 6) since the heat exchanger is still supplied with warm water. However, after transferring heat to the building, the water is not reheated by the boiler which leads to gradually declining supply and room temperatures. Because of the lower supply water temperature, one can also observe that the heating system needs longer to recover after the disturbance since the boiler inlet temperature is lower than when the pump was shut down.

For System 1, the same effects occur as when the pump failed: due to the improved insulation the temperature drops less and the system recovers faster.

In System 2, when the boiler shuts down, the supply water stays hotter for a longer term than in the reference system since it is fed by the stored hot water. However, the temperature in the tank gradually declines as well since the cold return water is led into it and thus the room temperature also declines. When comparing this temperature behavior with System 1, one can notice that System 2 needs longer to recover. This fact also leads back to the higher heat transfer coefficient of the windows and the therefore higher heat losses in System 2. Furthermore, the hot boiler outlet water is mixed with the colder storage water which leads to a colder temperature at the radiator inlet.

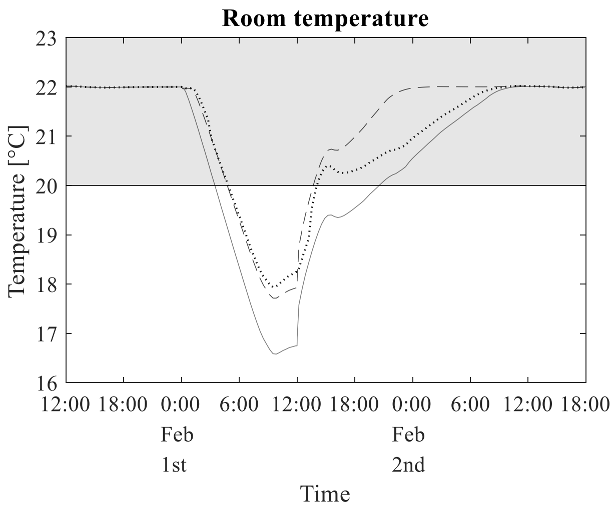


Figure 6. Temperature profiles during boiler failure (shaded: tolerance band, solid: Reference System, dashed: System 1, dotted: System 2).

3.2.3 Behavior of a Combined System

After focusing on single improvements in the considered system, a system combining both improvements was simulated with the presented disturbances. The temperature profiles are compared with those of System 1 and 2 in Figure 7 and Figure 8.

When considering the pump failure, it becomes once again obvious that the most important effect during this disturbance is the heat loss of the building. Accordingly, the temperature profile of the Combined System matches that of System 1. Installing a storage vessel in the system has no big impact when regarding this disturbance.

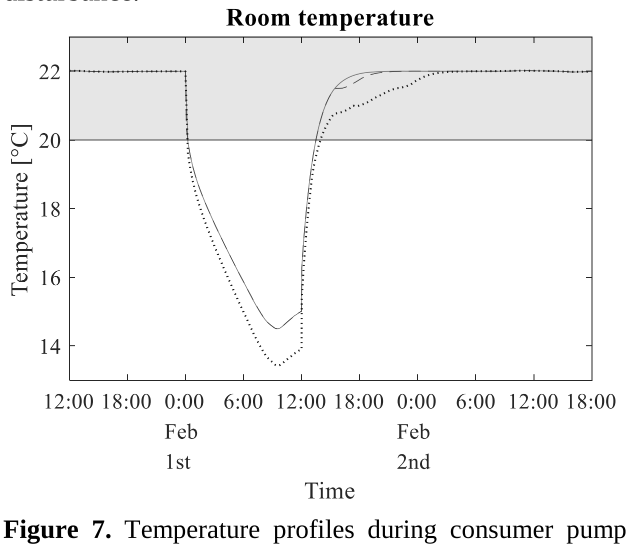


Figure 7. Temperature profiles during consumer pump failure (shaded: tolerance band, solid: Combined System, dashed: System 1, dotted: System 2).

When considering the boiler failure, the positive effects of lower heat losses due to better window insulation and a longer heat supply due to the installed heat storage add up to a significantly lower temperature drop and a faster system recovery.

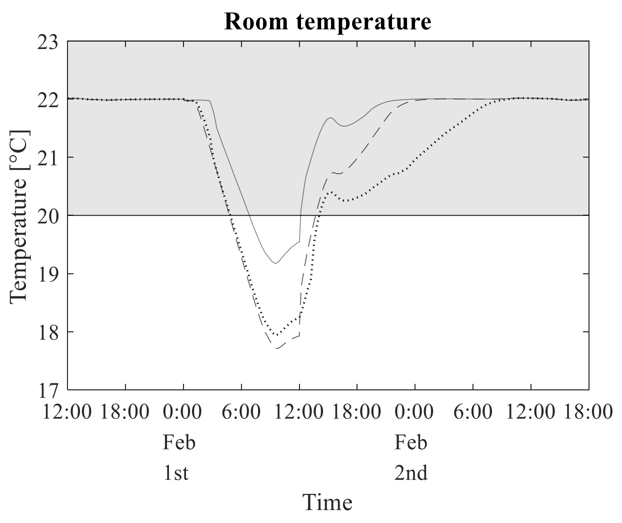


Figure 8. Temperature profiles during boiler failure (shaded: tolerance band, solid: Combined System, dashed: System 1, dotted: System 2).

3.3 Discussion of the Resilience Index

According to eq. (5), the resilience indices are computed for the presented systems under the influence of the two disturbance scenarios with and without considering the performance loss PL. The normalization values used are $\Delta x_{\mathrm{norm}} = 1$ K, $\Delta t_{\mathrm{norm}} = 6$ h and $A_{\mathrm{norm}} = \Delta x_{\mathrm{norm}} \cdot \Delta t_{\mathrm{norm}} = 6$ Kh. The results can be found in Table 2 and Table 3.

Overall, it is obvious that large temperature drops and recovery times lead to small resilience indices which shows that the resilience index reliably reflects the resistance and recovery ability of the system.

In general, the resilience indices are rather low which is plausible since the considered system does not feature common qualitative resilience aspects like redundancy, buffer capacity or variety (refer to (Fichter *et al.*, 2010) for more information). With regard to these low values, it should be noted that the absolute values of the indices are dependent on the choice of the normalization values. Therefore, it is important to only compare systems with the same normalization values since the results are otherwise distorted.

Table 2. Resilience indices for Reference System, System 1 and 2 and the Combined System during pump failure.

System		*Ref*	*1*	*2*	*Comb*
$\Delta x_{\max}$ [K]		6.6	5.5	6.6	5.5
Δt [h]		14.3	13.1	13.8	13.1
PL [Kh]		57.8	46.7	57.6	46.7
RI	w/ PL	0.01	0.02	0.01	0.02
	w/o PL	0.06	0.08	0.06	0.08

Table 3. Resilience indices for Reference System, System 1 and 2 and the Combined System during boiler failure.

System		*Ref*	*1*	*2*	*Comb*
$\Delta x_{\max}$ [K]		3.4	2.3	2.1	0.8
Δt [h]		16.8	9.1	9.1	5
PL [Kh]		26.5	12.7	12.3	2.9
RI	w/ PL	0.05	0.22	0.24	0.86
	w/o PL	0.09	0.21	0.24	0.6

Furthermore, it is apparent that in all cases the installation of windows leads to an increase of resilience since this improvement counteracts to the main reason of the system's vulnerability, the poor insulation of the building. Accordingly, implying this improvement is most effective in regards to resilience matters.

Another general aspect that becomes evident, is that the resilience indices vary for the same system in accordance to the disturbance it is exposed to. Hence, one can derive that there is no "absolute" resilience index, especially when keeping in mind that the concept of resilience also contains the system's capability to keep its functionality up when facing unknown disturbances. Therefore, when investigating a system's resilience, it is not sufficient to only look at one disturbance. In fact, the significance of a resilience analysis rises with its number of considered incidents.

Especially when looking at the results for the pump failure, one notices that all considered systems show very low resilience indices. This is in line with the observation that all systems experience significant temperature drops since the heat transfer to the building is directly cut off when the pump is not working. Thus, using the resilience index, is not only helpful when comparing different systems with each other but it can also reveal weak points of an energy system which consequently need to be protected or backed up more than others.

The multiplication with the performance loss leads to a weighting of the severity of the deviation. Since small deviation will lead to performance losses that are smaller than the normalization value, higher resilience indices will be calculated for these cases. This effect can be retraced by looking at the resilience index for the Combined System during the boiler shut down. Large deviation, however, lead to performance losses bigger than the normalization values and therefore even smaller resilience indices, as can be seen in the results of the Reference System and System 1 and 2.

4 Conclusion and Outlook

This paper introduces a definition of resilience for the assessment of energy supply systems by evaluating the maximum deviation, the recovery time and the performance loss of a system. Several approaches are presented and used to introduce a resilience index that can be applied to analyze dynamic simulation results as those produced by simulations in Modelica. This index was calculated for different heating system configurations of a single family dwelling.

It was shown that the use of the resilience index enables the comparison of two different system improvements. While System 1 focuses on the consumer side, System 2 changes the structure of the heating system.

To comprehend the development of the resilience index, a very simple example was chosen. However, the definition of the resilience index also allows the analysis of more complex systems and the efficient evaluation of proposed improvements.

The presented performance loss reflects a system’s recovery phase more than only focusing on the deviation’s amplitude and time outside the tolerance band. Thus, it is recommended to use this parameter when looking at systems that undergo pre-stable phases, to gain a more precise resilience evaluation. Additionally, the implementation of the performance loss leads to a weighting of the severity of deviations.

As shown, the presented resilience analysis enables further location of a system’s weak points which helps to choose and initiate system improvements that are the most efficient in regard to increasing the resilience.

Nevertheless, as in many evaluation methods, great caution needs to be taken when setting the evaluation conditions. This means that the absolute values of the resilience indices depend on the chosen normalization values. On the one hand, this constitutes the risk of comparing indices that are not comparable. On the other hand, it provides the flexibility to set the normalization according to the considered system. Therefore, an energy system which supplies sensitive infrastructure, for example a hospital, can be rated using smaller normalization values than a system supplying a residential area.

In addition, the quantification of the resilience allows a calculation of the “costs” of resilience – with regards to financial but also environmental aspects. As a result, a statement of how much more money or CO_2 emission lead to how much more resilience, can be made.

The introduced evaluation method deviates from methods in literature because it is not able to show a system’s adaptive capability since further social, economic and political aspects have to be considered that cannot be integrated in the physically based simulation environment of Modelica. The only way to approach this aspect is to perform several simulations that integrate system changes that are caused by a disturbance and influence the resilience in future scenarios. However, the fact remains that dynamic, technically-based simulations are not able to reflect the whole spectrum of resilience. For this reason, an additional qualitative assessment is recommended.

Further research should focus on using the resilience index on more complex systems including integrated energy systems and the evolutions that are necessary for these kinds of systems. Hence, it is proposed to allocate one resilience index for each integrated sector and combine them into one overall index which will make it possible to evaluate complex system changes, e.g. a rising share of renewables, with regards to resilience aspects.

Acknowledgements

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).

Reference list

Allen, M. *et al.* (2018) *Global Warming of 1.5 °C: An IPCC special report on the impacts of global warming of 1.5 °C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty*. Geneva (sr15). Available at: http://www.ipcc.ch/report/sr15/.

Andresen, L. *et al.* (2015) ‘Status of the TransiEnt Library: Transient Simulation of Coupled Energy Networks with High Share of Renewable Energy’, *Proceedings Modelica Conference 2015, The 11th International Modelica Conference,* September 21-23, 2015: Linköping University Electronic Press, pp. 695–705. doi: 10.3384/ecp15118695

Brunnemann, J. *et al.* (2012) ‘Status of ClaRaCCS: Modelling and Simulation of Coal-Fired Power Plants with CO2 Capture’, *Proceedings of the 9th International MODELICA Conference, September 3-5, 2012, Munich, Germany, 9th International MODELICA Conference, Munich, Germany,* Sept. 3-5, 2012: Linköping University Electronic Press, pp. 609–618. doi: 10.3384/ecp12076609

Bundesnetzagentur (2018) *Redispatch*. Available at: https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/Versorgungssicherheit/Engpassmanagement/Redispatch/redispatch-node.html.

Cimellaro, G.P. *et al.* (2009) *Quantification of Disaster Resilience of Health Care Facilities*. Buffalo.

Dassault Systèmes (2018) *Dymola®* (Version 2019). Vélizy-Villacoublay, France. Available at: https://www.3ds.com/de/produkte-und-services/catia/produkte/dymola/.

European Commitee for Standardization (2007) *EN 15251 2012-12, Indoor environmental input parameters for design and assessment of energy performance of buildings addressing indoor air quality, thermal environment, lighting and acoustics*. Brüssel.

Fichter, K. *et al.* (2010) *Theoretische Grundlagen für erfolgreiche Klimaanpassungsstrategien*. Bremen.

Francis, R. and Bekera, B. (2014) 'A metric and frameworks for resilience analysis of engineered and infrastructure systems', *Reliability Engineering & System Safety*, 121, pp. 90–103. doi: 10.1016/j.ress.2013.07.004

Gößling-Reisemann, S., Hellige, H.D. and Thier, P. (2018) *The Resilience Concept: From its historical roots to theoretical framework for critical infrastructure design*. Bremen (artec-paper 217). Available at: https://www.uni-bremen.de/fileadmin/user_upload/sites/artec/Publikationen/artec_Paper/217_paper.pdf.

Hamburg University of Technology (2017) *TransiEnt Library*. Hamburg. Available at: https://www.tuhh.de/transient-ee/en/.

Hamburg University of Technology, TLK-Thermo GmbH, XRG Simulation (2012) *ClaRa - Power Plant Simulation*. Hamburg, Braunschweig. Available at: https://www.claralib.com/.

Holling, C.S. (1973) 'Resilience and Stability of Ecological Systems', *Annual Review of Ecology and Systematics*, 4(1), pp. 1–23. doi: 10.1146/annurev.es.04.110173.000245

Lange, I. (2014) *Outdoor Temperatures, Hamburg Billwerder, 900s, 2012*. Available at: https://wettermast.uni-hamburg.de/.

Madni, A.M. and Jackson, S. (2009) 'Towards a Conceptual Framework for Resilience Engineering', *IEEE Systems Journal*, 3(2), pp. 181–191. doi: 10.1109/JSYST.2009.2017397

Modelica Association (2019) *Modelica*. Linköping. Available at: https://www.modelica.org/.

Molyneaux, L. *et al.* (2012) 'Resilience and electricity systems: A comparative analysis', *Energy Policy*, 47, pp. 188–201. doi: 10.1016/j.enpol.2012.04.057

Nan, C. and Sansavini, G. (2017) 'A quantitative method for assessing resilience of interdependent infrastructures', *Reliability Engineering & System Safety*, 157, pp. 35–53. doi: 10.1016/j.ress.2016.08.013

Roege, P.E. *et al.* (2014) 'Metrics for energy resilience', *Energy Policy*, 72, pp. 249–256. doi: 10.1016/j.enpol.2014.04.012

Senkel, A. (2017) *Vergleich verschiedener Arten der Wärmeverbrauchsmodellierung in Modelica*. Master Thesis. Hamburg University of Technology.

Thoma, K. (2014) *acatech STUDIE Resilien-Tech: „Resilience-by-Design": Strategie für die technologischen Zukunftsthemen*. Freiburg.

Wetter, M. *et al.* (2014) *Modelica Buildings Library*. Available at: https://simulationresearch.lbl.gov/modelica/download.html, doi: 10.1080/19401493.2013.765506.

Session 6B: Thermodynamic 2

Application of a Real Gas Model by van der Waals for a Hydrogen Tank Filling Process

Maximilian Kormann[1] Imke Lisa Krüger[2]

[1]Dassault Systèmes, Germany, maximilian.kormann@3ds.com
[2]Dassault Systèmes, Germany, imkelisa.krueger@3ds.com

Abstract

Hydrogen fuel tanks operate at high system pressure levels. In these regions effects occur, which cannot be handled by an ideal gas model. One of these is the Joule-Thomson effect. It describes an adiabatic throttling without change in enthalpy, but a change in temperature. The tank filling process can be simplified to a throttling valve, so the effect is of interest. In this investigation the van der Waals equations are implemented in a real gas model for the *Hydrogen Library* and the *Pneumatic Systems Library (PSL)* by Dassault Systèmes and the model is applied to a hydrogen tank filling process. Performance and accuracy are compared to the CoolProp fluid properties library, which is imported with the ExternalMedia library.

Keywords: Hydrogen, Pneumatic Systems Library, Tank Filling Process, Medium Models, Real Gas, van der Waals, Joule-Thomson-Effect, CoolProp

1 Introduction

The calculation of thermodynamic properties plays an important role in simulating thermohydraulic systems. The specific behaviour we want to study as well as the pressure and temperature range define the exactness needed of the medium properties model. The ideal gas approach is only valid for low pressures, complex approaches that cover the liquid phase as well slow down the simulation speed.

The van der Waals model gives a simplified analytical approach and it assumes the deviation to be acceptable for e.g. early stages of design or if only the qualitative behavior should be shown. The Joule-Thomson effect that is to be examined in this paper can be analytically modeled and studied with the van der Waals approach. It results in simple equations for e.g. the internal pressure.

The model has been implemented for the *Hydrogen* and *Pneumatic Systems Library* by Dassault Systèmes. New gases can be easily added, the only parameters needed are the molar mass of the gas, the critical pressure and temperature, the sutherland constants and the table-based specific internal energy.

A recent application where this effect occurs is the hydrogen tank filling processes for fuel cell vehicles. Hydrogen has to be stored at high pressures e.g. 200 bar due to the low energy density at low pressures. In order to be competitive to conventionally fueled cars, tank filling should take less than 200 s for 5 kilogram (US DOE et al., 2009). For safety reasons, the temperature inside the tank may not exceed 85 °C.

Due to its small molecules, hydrogen shows a different behavior when compressed or expanded than e.g. air. When air is expanded isenthalpic, temperature sinks where as for hydrogen the temperature rises. This is due to the different signs of the Joule-Thomson coefficient, that describes the temperature evolution during a pressure change.

To design the tank filling processes and identify cooling needs during the injection, system simulation is the best way. We will demonstrate in this paper that the van der Waals approach can be used for hydrogen to evaluate tank filling processes at an early stage of development.

2 Thermodynamic background

To describe the temperature change at an adiabatic throttling some thermodynamic basics have to be given.

2.1 Ideal Gas

The ideal gas model assumes the fluid to consist of dimensionless particles which can move freely in the given volume until they hit another particle or the wall. The thermal equation of state between the pressure p and temperature T is defined according to (Adkins, 1983; Tschoegl, 1983) with the specific gas constant R:

$$RT = pv \tag{1}$$

and the specific volume

$$v = \frac{V}{m} = \frac{1}{\rho} \tag{2}$$

2.2 Real gas equation by van der Waals

The ideal gas model lacks two effects: Interactive forces between the fluid molecules and their dimension. For the first effect, the pressure in the van der Waals model (der Waals, 1967) is increased by the internal pressure of the gas. The influence of the dimension of the molecules is considered by reducing the specific volume by the volume of the fluid molecules:

$$RT = \left(p + \frac{a}{v^2}\right)(v - b) \tag{3}$$

The two new parameters can be derived from the critical state of the fluid (Weigand et al., 2008). The internal pressure is defined using the cohension pressure parameter:

$$a = \frac{27\,(RT_c)^2}{64 p_c} \tag{4}$$

The volume occupied by the molecules can be written directly:

$$b = \frac{RT_c}{8 p_c} \tag{5}$$

2.3 Internal Pressure

The internal pressure of a fluid is defined as the partial derivation of the specific internal energy u by the specific volume at constant temperature (Moore, 1986):

$$\pi_T = \left(\frac{\delta u}{\delta v}\right)_T \tag{6}$$

It can be derived from the fundamental thermodynamic relation with the specific entropy s under assumption of a constant amount of mass (Callen, 1985) by deriving the internal energy partially by the volume at constant temperature:

$$du = T ds - p dv \tag{7}$$

$$\left(\frac{\delta u}{\delta v}\right)_T = T\left(\frac{\delta s}{\delta v}\right)_T - p \tag{8}$$

With the Maxwell-Relation (Weigand et al., 2008)

$$\left(\frac{\delta s}{\delta v}\right)_T = \left(\frac{\delta p}{\delta T}\right)_v \tag{9}$$

the expression for the internal pressure becomes:

$$\pi_T = \left(\frac{\delta u}{\delta v}\right)_T = T\left(\frac{\delta p}{\delta T}\right)_v - p \tag{10}$$

For an ideal gas we get from equation (1):

$$\left(\frac{\delta p}{\delta T}\right)_v = \frac{R}{v} \tag{11}$$

$$\pi_T = T\left(\frac{\delta p}{\delta T}\right)_v - p = T\frac{R}{v} - p = 0 \tag{12}$$

As the ideal gas equation does not model forces between the fluid molecules except elastic collisions, the internal pressure is zero. Thus the Joule-Thomson effect cannot be covered by an ideal gas law. In the real gas equation the van der Waals forces between the molecules are covered by the internal pressure term. From equation (3) we get:

$$\left(\frac{\delta p}{\delta T}\right)_v = \frac{R}{v-b} \tag{13}$$

$$\pi_T = T\left(\frac{\delta p}{\delta T}\right)_v - p = T\frac{R}{v-b} - p \tag{14}$$

$$= T\frac{R}{v-b} - \left(\frac{RT}{v-b} - \frac{a}{v^2}\right) = \frac{a}{v^2} \tag{15}$$

The internal pressure for a real gas modeled with the van der Waals equation is always positive.

2.4 Caloric equation of state

Assuming a constant amount of mass the total differential of the internal energy is (Weigand et al., 2008):

$$du = \left(\frac{\delta u}{\delta T}\right)_v dT + \left(\frac{\delta u}{\delta v}\right)_T dv = c_v dT + \pi_T dv \tag{16}$$

For an ideal gas, the internal pressure is zero and thus the internal energy only depends on the temperature with the specific heat capacity at constant volume c_v.

2.5 Specific Enthalpy

The enthalpy describes the energy of a system. The enthalpy can only change when energy is transferred over the system border. The specific enthalpy of a fluid is defined as (Weigand et al., 2008):

$$h = u + pv \tag{17}$$

For an ideal gas the expression simplifies to

$$h = u + RT \tag{18}$$

As described according to equation (16), for an ideal gas the internal energy only depends on the temperature. In this case, also the enthalpy only depends on the inner energy. This is not the case for a real gas.

2.6 The Joule-Thomson effect

When a gas is expanded from a high pressure over an isenthalpic valve, a change in temperature can be observed. This behaviour cannot be described by an ideal gas model as it does not model a pressure dependency of the specific enthalpy as mentioned in section 2.5. The effect is described by the Joule-Thomson coefficient (Greiner et al., 1993):

$$\mu_{JT} = \left(\frac{\delta T}{\delta p}\right)_h \tag{19}$$

To derive its equation, the total differential of the specific entropy under assumption of a constant amount of mass will be set into the fundamental thermodynamic relation from equation (7). It reduces the dependency of the change in enthalpy to pressure and temperature:

$$ds = \left(\frac{\delta s}{\delta T}\right)_p dT + \left(\frac{\delta s}{\delta p}\right)_T dp \tag{20}$$

$$dh = du + pdv + vdp \tag{21}$$

$$dh = T\left(\frac{\delta s}{\delta T}\right)_p dT + T\left(\frac{\delta s}{\delta p}\right)_T dp + V dp \tag{22}$$

With the definition of the specific heat capacity at constant pressure c_p and the Maxwell-Relation for the Gibbs energy we get:

$$T\left(\frac{\delta s}{\delta T}\right)_p = c_p \tag{23}$$

$$\left(\frac{\delta s}{\delta p}\right)_T = -\left(\frac{\delta v}{\delta T}\right)_p \tag{24}$$

$$dh = c_p dT - T\left(\frac{\delta v}{\delta T}\right)_p dp + V dp \tag{25}$$

Now we can set $dh = 0$ and rearrange:

$$\mu_{JT} = \left(\frac{\delta T}{\delta p}\right)_H = \frac{1}{c_p}\left(T\left(\frac{\delta v}{\delta T}\right)_p - v\right) \tag{26}$$

For $v >> b$ and $v >> a/RT$ the real gas equation (3) can be rewritten and solved by v (Greiner et al., 1993):

$$p = \frac{RT}{v}\left(1 + \frac{b}{v} - \frac{a}{vRT} + \frac{abp}{vR^2T^2}\right) \tag{27}$$

$$v = \frac{RT}{p} + b - \frac{a}{RT} + \frac{abp}{R^2T^2} \tag{28}$$

Then the inner differential from equation (26) can be formulated and put into the equation of the Joule-Thomson coefficient:

$$\left(\frac{\delta v}{\delta T}\right)_p = \frac{R}{p} + \frac{a}{RT^2} - 2\frac{abp}{R^2T^3} \tag{29}$$

$$\mu_{JT} = \frac{1}{c_p}\left(\frac{2a}{RT} - b - 3\frac{abp}{R^2T^2}\right) \tag{30}$$

3 Implementation as a real gas model

The equations from section 2.2 and 2.4 are introduced as an additional fluid model in the *Hydrogen Library* and the *Pneumatic Systems Library (PSL)*. The dynamic viscosity is estimated by the law of (Sutherland, 1893).

3.1 Density

The density is estimated according to the real gas model by van der Waals from equation (3).

3.2 Specific Enthalpy

To get a non-differential formulation of the specific enthalpy, equation (16) has to be integrated from a reference point indicated by a subscript 0 and put into equation (17):

$$h = \int_{T_0}^{T} c_v dT + \int_{v_0}^{v} \pi_T dv + pv \tag{31}$$

$$h = \underbrace{\int_{T_0}^{T} c_v dT - \frac{a}{v_0}}_{u_T(T)} + \frac{a}{v} + pv \tag{32}$$

As there is no easy-to-handle analytical expression for c_v and to avoid the use of polynomials, the temperature-dependent part of the specific internal energy and the internal pressure correction of the reference point (marked with an under-brace in equation (31)) is provided as a table-interpolation $u_T(T)$. Then we get the following equation for the specific enthalpy:

$$h = u_T(T) + pv + \frac{a}{v} \tag{33}$$

3.3 Parametrization

Compared to ideal gas models, the additional parameters for creating a new gas model are pressure and temperature in the critical state as well as the tables for the specific internal energy and heat capacity (see table 1). As the specific heat capacity is described by a look-up-table, the degrees of freedom are no longer necessary for the van der Waals Real Gas Model.

Table 1. Required parameters for the gas models.

Gas model	*Ideal*	*Real*
Molar mass	●	●
Dynamic viscosity at ref. point	●	●
Viscosity ref. point temperature	●	●
Sutherland constant	●	●
Degrees of freedom	●	
Critical pressure		●
Critical temperature		●
Temperature table		●
Specific internal energy table		●
Specific heat capacity table		●

4 Benchmark of the van der Waals model

To determine a range for the validity of the gas model, it will be compared to generated values by the CoolProp library (Bell et al., 2014) for pure Hydrogen gas. The open source C++ library CoolProp provides equations of state and transport properties for pure fluids and mixtures. The relative deviations of results for various fluid properties of hydrogen compared to the NIST Reference Fluid Thermodynamic and Transport Properties Database (Lemmon et al., 2010) are in the range of 1×10^{-4} to 1×10^{-14}. The CoolProp library is interfaced with the help of the ExternalMedia library (Casella and Richter, 2008). The ExternalMedia provides medium packages that can be used directly in the Pneumatic Systems Library and in the Hydrogen library.

4.1 Density of hydrogen

In the following, the influence of the gas model on the density will be outlined in the case of hydrogen.

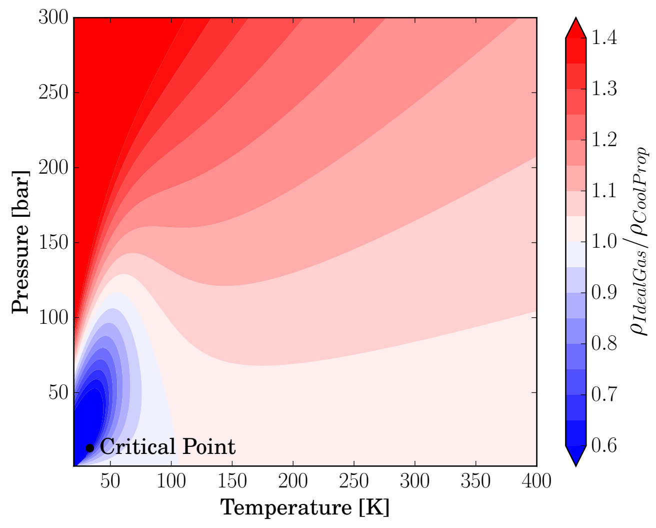


Figure 1. Comparison of the density between the ideal gas model and the CoolProp model for hydrogen.

The ideal gas model has a quite small range of validity: To keep the error in the density below 5%, the pressure has to be below 50 bar and the temperature has to be above 100 K (see Figure 1). As discussed in section 2.3, the Joule-Thomson effect can not be covered by the ideal gas law, as it does not consider intermolecular forces.

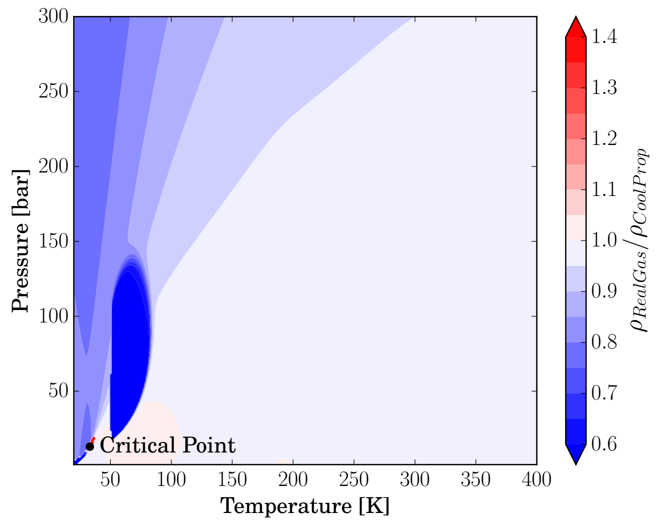


Figure 2. Comparison of the density between the van der Waals gas model and the CoolProp model for hydrogen.

The van der Waals model has a much wider range of validity (see Figure 2). Pressures up to 200 bar and temperatures above 100 K can be covered with an error in the density below 5%. For the same pressure and temperature the error is smaller than for the ideal gas model. Besides the wider range of application, the Joule-Thomson effect can be covered by the van der Waals air model, as discussed in section 2.3.

4.2 Density of air

Though air is a mixture of several gases, it still can be modeled by the van der Waals real gas approach. As stated in section 2.5, only the critical state of the fluid has to be known. This can be measured for air. Again, the density of the gas model will be compared to the CoolProp library.

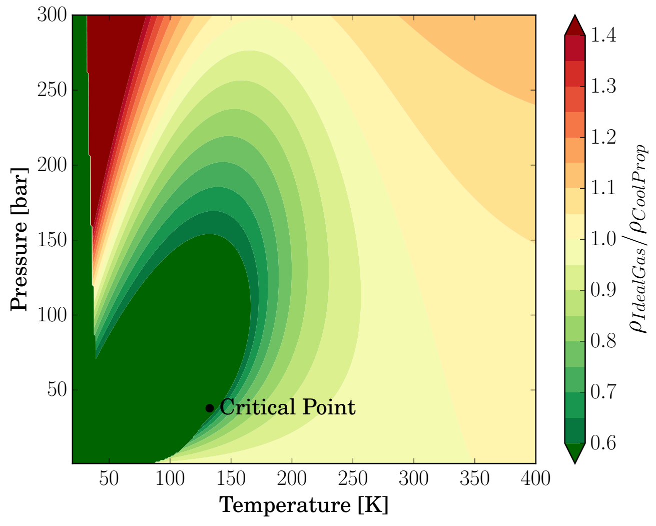


Figure 3. Comparison of the density between the ideal gas model and the CoolProp model for air.

The ideal gas model has quite small range of validity: To keep the error in the density below 5%, the pressure has to be below 150 bar and the temperature has to be above 250 K (see Figure 3).

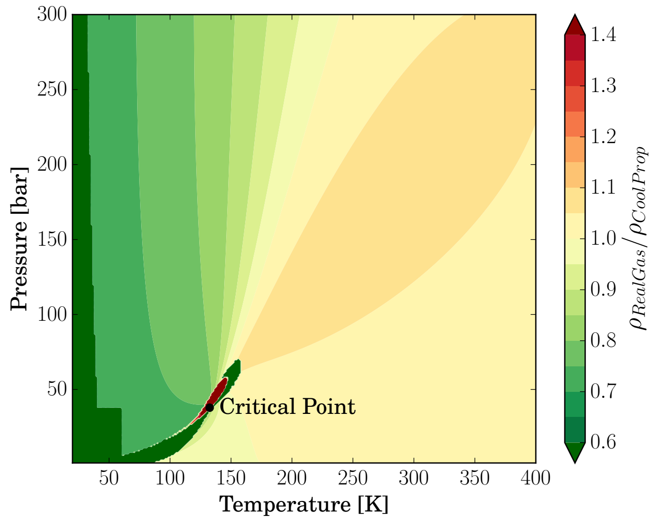


Figure 4. Comparison of the density between the van der Waals gas model and the CoolProp model for air.

The van der Waals air model has a slightly wider range of validity (see Figure 4). Pressures up to 200 bar and temperatures above 170 K can be covered with an error in the density below 5%. The benefit in application range by using the van der Waals model for air is smaller than for hydrogen. Still low temperatures can be handled with a higher precision and the Joule-Thomson effect is covered.

4.3 Joule-Thomson coefficient

In section 2.6 an analytic formulation for the Joule-Thomson coefficient has been derived in equation (30). It will be compared to the direct derivative of the enthalpy calculated in the *PSL* according to equation (19) for hydrogen:

```
temp_der_T = der(gas.temperature, p);
mu_jt_modelica = temp_der_T(p=p,h=h);
mu_jt_analytical = 1 /
    gas.specificHeatCapacityAtConstantPressure
    (p=p,T=T) * ( 2 * gas.a / gas.R / T -
    gas.b - 3 * a * b * p / R^2 / T^2 );
```

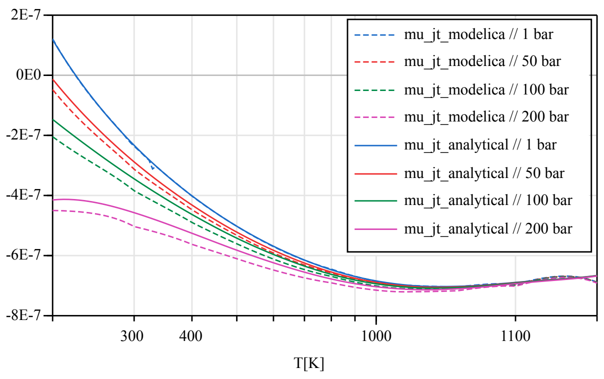


Figure 5. Analytical and Numerical Joule-Thomson coefficient for hydrogen.

The result for different pressures over the logarithmic scaled temperature is shown in Figure 5. For ambient pressure both definitions show the same behaviour. For higher pressures the assumption $v >> b$ from equation (27) cannot be met anymore and the analytical description of μ_{JT} recedes from the numerical derivative formulated in the *PSL*.

5 Applications

5.1 Hydrogen fuel tank

To simulate the filling process of a hydrogen tank a simplified model utilizing the *Hydrogen Library* as in Figure 6 is used. The fluid part consists of a pressure source at 200 bar, a variable orifice and a reservoir with a heat connection to the environment. The mass flow rate should be kept constant over the filling process, so a PI controller is used to set the valve opening. To restrict the maximum temperature in the tank during the filling process, the hydrogen gas from the pressure source is assumed to be precooled at $-10\,°C$.

5.2 Results

The temperature in the tank during filling process is plotted in Figure 7 for different hydrogen media models. The ideal gas model

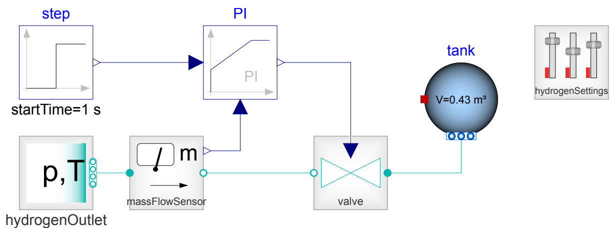


Figure 6. Dymola model to simulate a tank filling process for hydrogen.

(`Modelica.Media.IdealGases.SingleGases.H2`) strongly underestimates the maximum temperature as it only takes into account the temperature rise due to translational work from the pressure step at the inlet of the tank. Both the CoolProp and the van der Waals Real Gas Model consider the additional temperature rise due to the Joule-Thomson effect, while it is slightly overestimated by the van der Waals Real Gas Model. In the use-case of the tank filling process the safer approach is to use a model overestimating the temperatures, so the safety margin to the temperature limitation is increased.

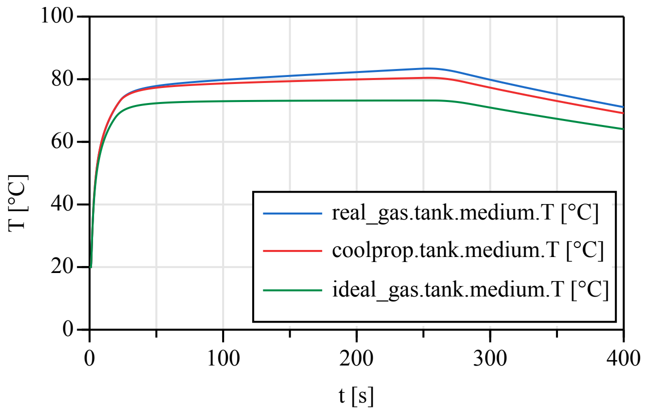


Figure 7. Results of the tank filling process simulation.

5.3 Performance

To compare the performance of the van der Waals gas model, the fuel tank experiment has been conducted using the *Hydrogen Library* ideal and van der Waals gas model as well as the CoolProp Library with the ExternalMedia interface (Casella and Richter, 2008). The results are summarized in table 2.

Table 2. Results of the performance test.

Fluid model	*Simulation time*
Ideal gas model / *Hydrogen*	0.2 s
van der Waals model / *Hydrogen*	1.06 s
CoolProp lib. / ExternalMedia	4.18 s

The van der Waals model has a significant performance advantage against the ExternalMedia approach at the cost of lower accuracy.

5.4 Excursus: Experiment of Gay-Lussac

The experiment of Gay-Lussac was conducted in 1807 and repeated by (Joule, 1845). A volume with a high pressure at 22 bar and environment temperature of 20 °C is connected to an evacuated volume and the gas flows until both pressures are equal. After reaching thermodynamic equilibrium, the same temperature as the start temperature was measured in both volumes. With the measurement accuracy of that time the ideal gas law was proven. The model shown in figure 8 is an example model of the *PSL*. The two volumes are modeled as spheric tanks with a heat transfer connection. Between the pneumatic ports of the volumes a V22 valve prevents air flow. After one second the valve opens and gas flows until the equilibrium is reached.

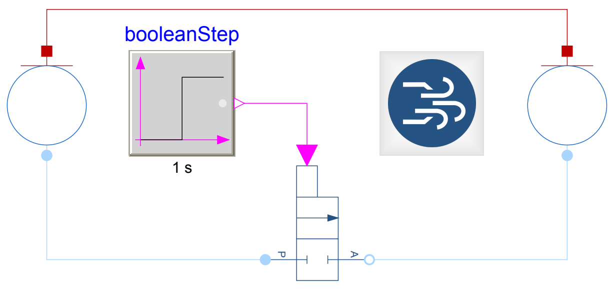


Figure 8. Dymola model to simulate the experiment of Gay-Lussac.

With more accurate measurement it was found that the temperature after reaching thermodynamic equilibrium was 3 K below the initial temperature. As the experiment did not allow any change in enthalpy, this behavior could only be explained by the real gas law. The results in table 3 show that the van der Waals model covers this real gas effect.

Table 3. Results of the experiment of Gay-Lussac.

Fluid model	*End temperature*
Ideal gas model / *PSL*	20.0 °C
van der Waals model / *PSL*	17.2 °C
CoolProp lib. / ExternalMedia	16.9 °C

6 Summary

The investigation shows that the van der Waals Real Gas Model is a good compromise between accuracy and simulation performance. A big advantage of the model is the possibility to describe the influence of the internal pressure on the specific enthalpy fully analytical. Thus it can be reduced to a single additional term in the formulation of the internal energy. A van der Waals model can be easily adapted to different gases as the formulation only requires pressure and temperature at the critical state, the molar mass and a table for the temperature-dependent part of the internal energy.

References

C. J. Adkins. *Equilibrium Thermodynamics (3rd ed.)*. Cambridge University Press, Cambridge, UK, 1983. ISBN 0-521-25445-0.

Ian H. Bell, Jorrit Wronski, Sylvain Quoilin, and Vincent Lemort. Pure and pseudo-pure fluid thermophysical property evaluation and the open-source thermophysical property library coolprop. *Industrial & Engineering Chemistry Research*, 53(6):2498–2508, 2014. doi:10.1021/ie4033999. URL `http://pubs.acs.org/doi/abs/10.1021/ie4033999`.

H. B. Callen. *Thermodynamics and an Introduction to Thermostatistics*. John Wiley & Sons, New York, 1985. ISBN 978-0471862567.

Francesco Casella and Christoph Richter. Externalmedia: A library for easy re-use of external fluid property code in modelica. *Proceedings of 6th International Modelica Conference*, pages 157–161, 2008.

J. D. Van der Waals. The equation of state for gases and liquids. *Nobel Lectures, Physics 1901-1921*, pages 254–265, 1967.

Walter Greiner, Ludwig Neise, and Horst Stöcker. *Thermodynamik und statistische Mechanik*. Verlag Harri Deutsch, Frankfurt am Main, 1993. ISBN 978-3808557082.

J.P. Joule. Liv. on the changes of temperature produced by the rarefaction and condensation of air. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 26(174):369–383, 1845. doi:10.1080/14786444508645153.

Eric W. Lemmon, M. L. Huber, and M. O. McLinden. *NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties - REFPROP*. National Institute of Standards and Technology, Standard Reference Data Program, Gaithersburg, 9.0 edition, 2010. URL `http://www.nist.gov/srd/nist23.cfm`.

W. Moore. *Grundlagen der Physikalischen Chemie*. De Gruyter, Berlin, 1986. ISBN 978-3110099416.

W. Sutherland. The viscosity of gases and molecular force. *Philosophical Magazine 5*, pages 507–531, 1893.

N. W. Tschoegl. *Fundamentals of Equilibrium and Steady-State Thermodynamics*. Elsevier, Amsterdam, 1983. ISBN 0-444-50426-5.

Office of energy efficiency US DOE, Renewable energy, the FreedomCAR, and fuel Partnership. Targets for onboard hydrogen storage systems for light-duty vehicles. Technical report, 2009.

B. Weigand, J. Köhler, and J. v. Wolfersdorf. *Thermodynamik kompakt*. Springer-Verlag, Berlin, 2008. ISBN 978-3-540-71865-9.

Modeling of the Flow Comparator Prototype as New Primary Standard for High Pressure Natural Gas Flow Metering

Sukhwinder Singh[1] Gerhard Schmitz[1] Bodo Mickan[2]

[1]Institute for Engineering Thermodynamics, Hamburg University of Technology, Germany, {sukhwinder.singh,schmitz}@tuhh.de
[2]Physikalisch-Technische Bundesanstalt, Braunschweig, Germany, bodo.mickan@ptb.de

Abstract

The German national metrological institute, Physikalisch-Technische Bundesanstalt, is developing a new concept for volumetric primary standard to calibrate high pressure gas flow meters. The TUHH is supporting these R&D activities with its competence to elaborate computational models for detailed analysis of complex mechanical systems including fluid flow aspects. The new primary standard is based on a actively driven piston prover to measure the gas flow rate using the time the piston needs to displace a defined enclosed volume of gas in a cylinder.

A computational model written in Modelica® is developed to investigate the Flow Comparator's dynamic behavior. Validation of the model shows good compliance of the piston velocity and differential pressure at the piston in the model with measured data. With this model the control voltage trajectory can be optimized to increase the available measuring time and it allows to gather detailed information about pressure and temperature development at arbitrary chosen locations in the system with high time resolution.

Keywords: modeling of multi-domain physical systems, flow comparator, high pressure natural gas flow metering, linear motor, optimization

1 Introduction

For the trade with natural gas the uncertainty of high pressure natural gas flow meters is of major importance. The calibration of the flow meters is done with transfer standards which are calibrated by the German national primary standard for high pressure natural gas flow. The current primary standard is a High Pressure Piston Prover (HPPP) (Schmitz and Aschenbrenner; PTB, 1991, 2009). It is owned and operated by the National Metrology Institute of Germany Physikalisch-Technische Bundesanstalt (PTB) and installed on the calibration facility for gas meters pigsar™ in Dorsten, Germany. The HPPP can be operated with inlet pressures up to 90 bar and flow rates up to 480 m^3/h (PTB, 1991).

Due to the increasing size and flow rates of the gas flow meters and the limited operation range of the current national standard, a new concept for calibrating gas flow meters is being developed, the Flow Comparator. A development prototype of the Flow Comparator is used for preliminary tests such as investigating the controllability and the usable flow rate at ambient conditions. A picture of the prototype is shown in Figure 1.

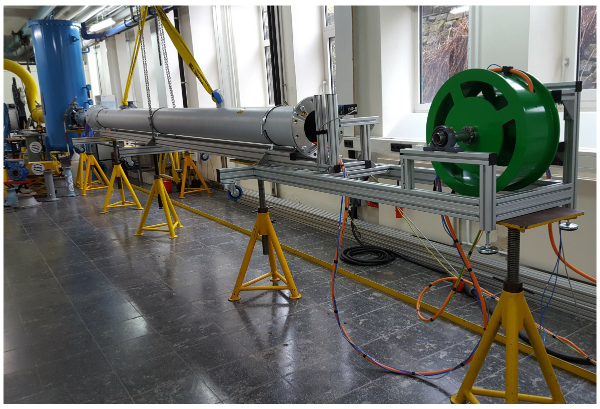


Figure 1. Picture of the Flow Comparator prototype

2 Experimental Setup

The key element of the Flow Comparator is a piston in a cylinder. Together they act as an asynchronous linear motor. For this, the cylinder has two layers, one with magnetic properties and the other one acts as an electrical conductor. The stator core with its windings is integrated into the piston. For the electrical power of the stator core a supply cable is connected to the piston. The velocity of the piston is controlled by using a frequency inverter to set the control voltage and frequency for the stator core.

The experimental setup is shown in Figure 2. The differential pressure over the piston is measured with a sensor in the piston. A specified leakage in the piston with a flow sensor measures the fluid flowing through it. With the two sensors, it is possible to compare the piston movement relative to the fluid flow. The piston has an integrated check valve to limit the pressure drop downstream of the piston.

The position of the piston is measured using a distance measuring equipment (DME). The ambient temperature and pressure as well as the temperature and pressure downstream of the cylinder are measured.

A Turbine Meter (TM) is used as transfer standard. The TM measures the volume flow rate using the rotational

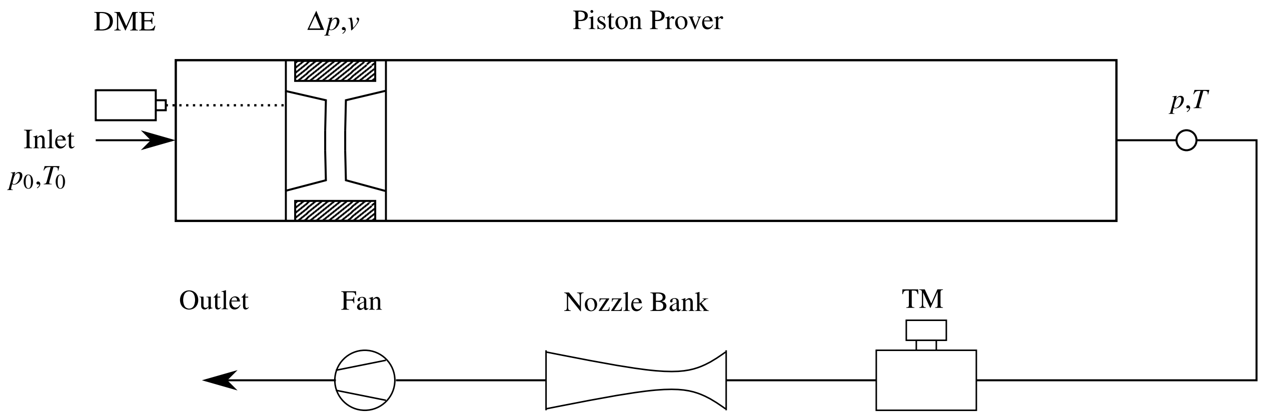


Figure 2. Scheme of the experimental setup with the Flow Comparator prototype

velocity of a turbine inserted into the fluid flow. The rotational speed is measured using a magnetically induced discrete signal of the turbine blades. The nozzle bank is used to set the flow rate and consists of calibrated nozzles with different volume flow rates in a parallel setup. The nozzle bank is not needed for the operation of the Flow Comparator, but provides the advantage of decoupling the experimental setup from pressure fluctuations created by the fan. The fan ensures that the pressure downstream of the nozzle bank is low enough to have a critical flow in the nozzles.

At the beginning of a measurement, the volume flow rate is set by the nozzle bank and the piston moves slowly upstream. At the starting point, the piston is accelerated downstream until the piston velocity is the same as the fluid velocity. The actual measurement phase starts when the piston reaches the defined velocity and moves past a certain point. The measurement phase ends at a defined point downstream where the piston is stopped. The volume flow rate can be calculated as stated in Equation 1, with the volume in between the starting and end point and the time span $\Delta_{FC}t$. Therefore, the Flow Comparator is traceable to the standards of length and time.

$$\dot{V}_{\mathrm{FC}} = \frac{V_{\mathrm{FC}}}{\Delta_{\mathrm{FC}}t} \tag{1}$$

In the same time span, the discrete pulses of the turbine meter are counted. The volume flow rate V_{TM} indicated by the turbine meter can be calculated using a relationship between volume flow rate and indicated signals per time period for the turbine, which is known from previous calibration (or from manufacturer specifications).

The calibration result is the relative deviation f at a certain volume flow rate and pressure. The relative deviation is calculated as stated in equation 2 with the corrected volume flow rate as indicated by the turbine and the corrected volume flow rate as indicated by the Flow Comparator.

$$f = \frac{\dot{V}^{\mathrm{c}}_{\mathrm{TM}} - \dot{V}^{\mathrm{c}}_{\mathrm{FC}}}{\dot{V}^{\mathrm{c}}_{\mathrm{FC}}} \tag{2}$$

To improve the calibration accuracy, some corrections are applied to the indicated volume flow rate by the turbine meter and the Flow Comparator. These are explained in the following:

1. With equation 3 it is possible to correct the error caused by the discrete nature of the turbine meter signals. As stated before, the time span $\Delta_{\mathrm{FC}}t$ is the duration of the measurement phase. The time span $\Delta_{\mathrm{TM}}t$ is the duration from the first signal of the turbine meter after the start of the measurement phase to the first signal of the turbine meter after the end of the measurement phase.

$$\dot{V}^{\mathrm{c}}_{\mathrm{TM}} = \dot{V}_{\mathrm{TM}} \frac{\Delta_{\mathrm{FC}}t}{\Delta_{\mathrm{TM}}t} \tag{3}$$

2. The following two corrections are applied due to small differences of piston velocity to the fluid velocity. With the differential pressure sensor and the fluid flow velocity sensor it is possible to compare the flow upstream and downstream of the piston.

   A non-zero differential pressure at the piston results in a leakage around the piston. The relationship between leakage and differential pressure is stated in Equation 4.

$$\dot{V}_{\mathrm{leak,\Delta p}} = a\sqrt{\Delta p} + b \tag{4}$$

   The coefficients a and b can be estimated by experiments.

   The volume flow correction with the differential pressure sensor is practical for relatively high leakage flows. For small leakages at the piston the fluid flow velocity sensor can be used. A non-zero velocity indicated by the fluid flow velocity sensor results in a leakage around the piston. The relationship is shown in equation 5 where the coefficients c and d are also experimentally determined.

$$\dot{V}_{\mathrm{leak,v}} = cv^2 + dv \tag{5}$$

3 Description of the model

Modelica® was used as modeling language to describe the physical and dynamic behavior of the Flow Comparator. As simulation environment Dymola is used. A graphical representation of the developed model is shown in Figure 3.

The assumptions used in the model are (von der Heyde et al., 2015):

- pressure losses are proportional to the dynamic pressure,
- the gas flow is one dimensional,
- the system is adiabatic,
- potential energy of the gas is neglected,
- the heat transfer in the gas can be neglected in comparison to convective energy transport.

The air used in the Flow Comparator is sucked out of the experimental hall. Therefore, a constant ambient temperature and pressure can be assumed. This is modeled using a supply volume of infinite size from the Modelica Standard Library (MSL). These boundary conditions are set by equation 6 and equation 7. p_{In} is the inlet pressure and T_{In} is the inlet temperature

$$p_{\mathrm{In}} = const. \tag{6}$$

$$T_{\mathrm{In}} = const. \tag{7}$$

The air properties are calculated using an air model of the MSL.

Another boundary condition is set by the nozzle bank. As aforementioned the fan ensures that the pressure downstream of the nozzle bank is low enough to have critical flow in the nozzle. The critical volume flow rate $\dot{V}_{\mathrm{N}}$ in the nozzle is set by Equation 8.

$$\dot{V}_{\mathrm{N}} = const. \tag{8}$$

The physical behavior of several nozzles is the same to one larger nozzle with equivalent diameter. Therefore, the nozzle bank is modeled as one nozzle with larger diameter based on equations from International Standard DIN EN ISO 9300 (International Organization for Standardization, 2005). The mass flow rate in the nozzle is calculated in Equation 9 using the critical volume flow rate $\dot{V}_{\mathrm{N}}$ and the upstream density ρ.

$$\dot{m}_{\mathrm{N}} = \dot{V}_{\mathrm{N}}\rho \tag{9}$$

For the model, the measuring cylinder is divided into one volume upstream of the piston and one volume downstream of the piston. The enclosed gas volumes depend on the position of the piston and change volume with piston movement. They can store mass m, internal energy mu and momentum mv as described in Equation 10, 11 and 12.

$$\frac{dm}{dt} = \dot{m}_i + \dot{m}_{i+1} \tag{10}$$

$$\begin{aligned}\frac{d}{dt}mu = \dot{m}_i\left(h_i + \frac{v_i^2}{2}\right) + \dot{m}_{i+1}\left(h_{i+1} + \frac{v_{i+1}^2}{2}\right)\\ + \left(\frac{p_{i+1} - p_i + p_{\mathrm{f},i+1} - p_{\mathrm{f},i}}{2}\right)\dot{V}_i + \dot{Q}\end{aligned} \tag{11}$$

$$\begin{aligned}\frac{d}{dt}mv = \dot{m}_i|v_i| + \dot{m}_{i+1}|v_{i+1}| - A\left(p_{i+1} - p_i\right)\\ - A\left(p_{\mathrm{f},i+1} - p_{\mathrm{f},i}\right)\end{aligned} \tag{12}$$

In direction of fluid flow a spatial discretization is applied which leads to a number of finite volumes in the enclosed gas volume. For the discretization the finite volume method with a staggered grid approach is used. Figure 4 shows the placement of variables on a 1D staggered mesh. The scalar variables (e.g. pressure, density etc.) are located in the control volume cell center while the velocity and momentum variables are stored on the cell faces.

Equations 10-12 are applied for each finite volume in the enclosed gas volume. $\dot{m}$ is the mass flow rate, h the specific enthalpy, v the mean velocity in the cross area, p the static pressure, p_{f} the pressure loss due to friction, V the volume and $\dot{Q}$ is the heat flow.

The pressure loss is calculated using the detailed characteristic wall friction model from the MSL. The model calculates the pipe friction coefficient depending on the Reynolds number and the relative roughness. A heat port is included in the model of the enclosed gas volume and can be connected to another heat port, e.g. the ambient or the piston. The heat flow in the model is calculated using a heat transfer model from the MSL.

The position and motion of the piston is determined by the equation of motion as stated in Equation 13.

$$m_{\mathrm{P}}\ddot{s}_{\mathrm{P}} = p_1 A_{\mathrm{P}} - p_2 A_{\mathrm{P}} - F_{\mathrm{R,P}} - F_{\mathrm{R,C}} + F_{\mathrm{LM}} \tag{13}$$

p_1 and p_2 are the pressures of the fluid upstream and downstream of the piston, $F_{\mathrm{R,P}}$ is the roll resistance of the piston, $F_{\mathrm{R,C}}$ the resistance of the connection cable and F_{LM} is the force of the linear motor to drive the piston. The roll resistance of the piston is modeled using a constant rolling resistance coefficient as stated in equation 14 with c_{R} being the roll resistance coefficient and F_{N} being the normal force of the piston.

$$F_{\mathrm{R,P}} = c_{\mathrm{R}} F_{\mathrm{N}} \tag{14}$$

The resistance due to the weight of the connection cable $F_{\mathrm{R,C}}$ is modeled as shown in Equation 15 with g being the gravitational force, m_{C} the total weight of the connection cable, s the current position of the piston and l the total length of the connection cable.

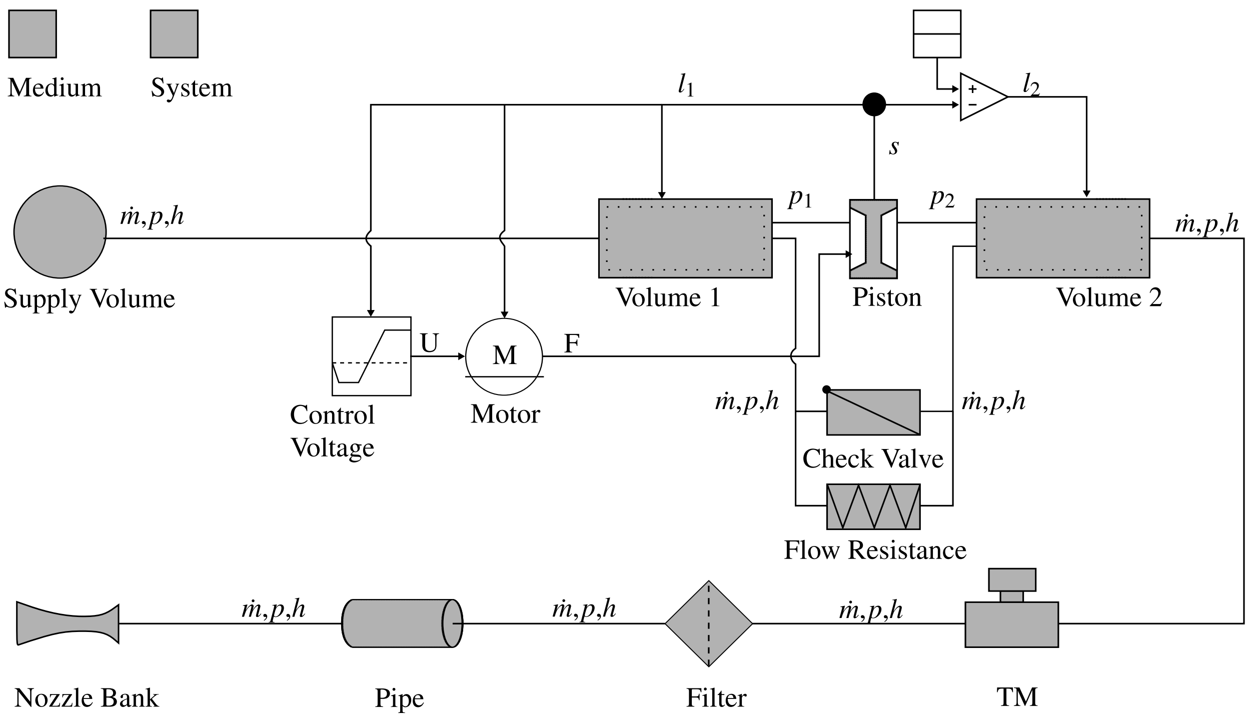


Figure 3. Graphical representation of the computational model

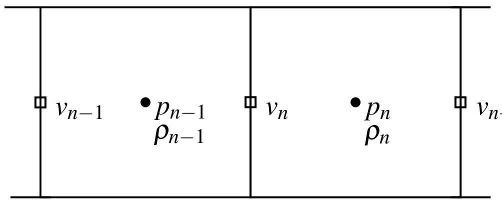


Figure 4. Placement of variables using the finite volume method with a staggered grid approach

$$F_{\mathrm{R,C}} = g \cdot m_{\mathrm{C}} \frac{s}{l} \tag{15}$$

The movement of the piston can be controlled with the linear motor integrated into the piston. As a first approach to model the force of the linear motor, a function depending on control voltage input and velocity of the piston is used. The function is derived by measuring the velocity of the piston for several control voltages and different flow resistances. The fitting function used is shown in Equation 16.

$$F_{\mathrm{LM}} = \alpha(I - I_{\mathrm{S}}) \cdot \left(\frac{v}{U_{\mathrm{control}}} - v_{\mathrm{S}}\right) + F_{\mathrm{F}} \tag{16}$$

α is a proportional constant, I is the electric current of the linear motor, I_{S} is the magnetizing current for the magnetic field, v is the velocity of the piston, v_{S} is the normalized synchronous velocity of the linear motor and F_{F} is the friction force of the piston. The experimental data and the result of the fitting is shown in Figure 5 where each line represents a different control voltage. The experimental data and fitting function are close-fitting overall. The output of the linear motor model is the force accelerating the piston. The motor model needs the current velocity of the piston and control voltage of the frequency inverter as input.

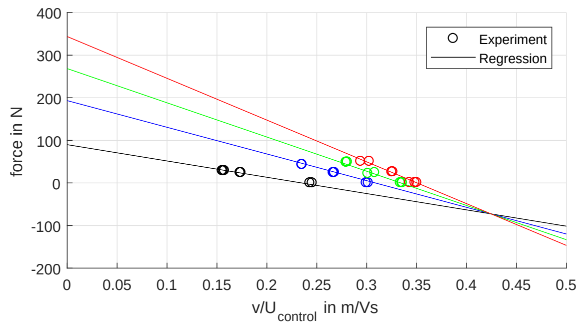


Figure 5. Experimental results and fitting function for the force-velocity relation of the linear motor

The control voltage model depends on the position of the piston. Initially, it increases linearly to a predefined negative value. When the starting position of the piston is reached, the control voltage increases to a defined positive value.

The check valve in the piston is modeled as a check valve between the gas volumes of the measuring cylinder. It opens at a specific differential pressure and enables a fluid flow from volume 1 to volume 2. The check valve is modeled with a hysteresis to avoid chattering. The mass flow rate through the check valve is proportional to the pressure drop over the check valve. For this a pressure loss coefficient ζ_{CV} as specified by the manufacturer is used.

The flow resistance model describes the leakage between measuring cylinder and piston as there is a small diameter difference between the two. To describe the relation of mass flow rate and pressure drop a function is derived from experiments.

The turbine meter model uses a constant pressure loss coefficient ζ_{TM} to model the pressure loss in the turbine meter as shown in Equation 17. ρ is the density and v_A the mean velocity in the cross area A of the turbine meter.

$$\Delta p = \zeta_{\mathrm{TM}} \frac{\rho}{2} v_{\mathrm{A}}^2 \tag{17}$$

The pressure loss coefficient ζ_{TM} is taken from measurement data. The relationship between indicated volume flow rate and real volume flow rate of the turbine is modeled as stated in Equation 18 which is a further development of the equation described in (Mickan et al., 2010). $v_{\mathrm{i,rel}}$ is the relative indicated volume flow rate in relation to the maximal possible volume flow rate for the specific turbine meter, v the real volume flow rate, ρ is the density and v_{i} the indicated volume flow rate. The coefficients a, b, A and B are results of different experiments.

$$\dot{v}_{\mathrm{i,rel}} - (a + b v_{\mathrm{i,rel}}) = A\rho v^2 - B\rho v v_{\mathrm{i}} \tag{18}$$

The filter is modeled as simple flow resistance and the pressure loss is determined as shown in equation 17. The DynamicPipe model of the MSL is used as the pipe model. It uses the balance equations for mass m, internal energy mu and momentum mv shown in Equation 19, 20 and 21 on a number of finite volumes in the pipe. $\dot{m}$ is the mass flow, h the specific enthalpy, v the velocity, A the cross-sectional area of the pipe, p the pressure and F_{F} the friction force in the pipe (Mickan et al., 2010).

$$\frac{dm}{dt} = \dot{m}_i + \dot{m}_{i+1} \tag{19}$$

$$\begin{aligned} \frac{d}{dt} mu &= \dot{m}_i h_i + \dot{m}_{i+1} h_{i+1} \\ &+ \frac{1}{2}\left(vA\left(p_{i+1} + p_i\right) + vF_{\mathrm{F}}\right) \end{aligned} \tag{20}$$

$$\frac{d}{dt} mv = \dot{m}_i |v_i| + \dot{m}_{i+1} |v_{i+1}| - A\left(p_{i+1} - p_i\right) - F_{\mathrm{F}} \tag{21}$$

4 Verification

The verification of a model shows the correct physical implementation of a model and the accurate solution of the equation system. For this, different parameter of the model are varied and piston velocity and pressure difference over the piston are used as measure for verification.

For the solution of the equation system, the solver Dassl with a relative tolerance of 10^{-6} is used. A further decrease in relative tolerance as well as using other solvers does not change the model trajectory. For the volumes of the measuring pipe, 8 discrete volumes are used and for the pipe, 4 discrete volumes are used.

In Figure 6, the piston velocity for different control voltages of the frequency inverter is shown. The acceleration of the piston is the same as it primarily depends on the power ramp of the frequency inverter. Higher control voltages have a greater maximum peak velocity as well as end velocity.

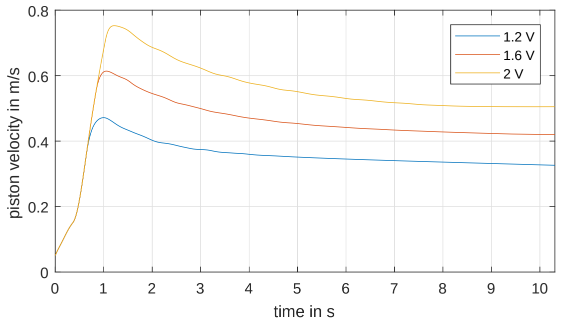


Figure 6. Piston velocity over time for different frequency inverter control voltages

The piston velocity for different rising time for the ramp of the frequency inverter is shown in Figure 7. As expected, a lower rising time results in a higher acceleration of the piston due to the faster increasing current in the linear motor and therefore a higher induced force on the piston. After the control voltage has increased to the defined value, the difference in piston velocity decreases and the piston velocity is almost the same at the end of the simulation.

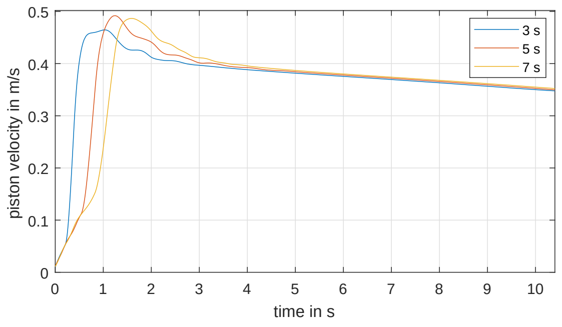


Figure 7. Piston velocity over time for different frequency inverter rising time

In Figure 8, the differential pressure at the piston for different volume flow rates of incoming air flow is shown. For a volume flow rate of 65 m^3/h, the piston velocity is close to the velocity of the air flow and the differential pressure is almost zero after the acceleration process. As the piston velocity is almost independent of the air flow rate, a lower volume flow rate results in a negative differential pressure at the piston. A negative differential pressure means that the downstream pressure is higher than the upstream pressure due to the faster movement of the piston in comparison to the air flow. Accordingly, a higher air flow rate results in a positive differential pressure around the piston.

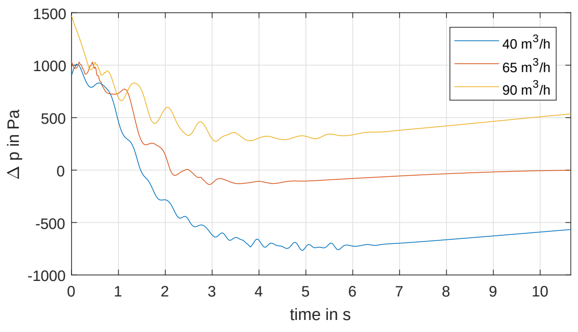


Figure 8. Differential pressure at the piston over time for different air flow rates

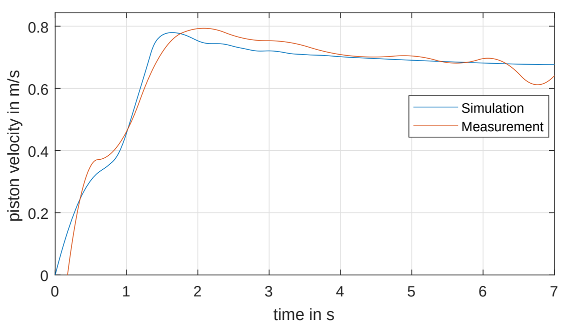


Figure 9. Comparison of the piston velocity over the time in the model and the measured data for a volume flow rate of 116 m^3/h and a control voltage of 1.95 V

5 Validation

To use the model for further predictions of the dynamic behavior of the Flow Comparator, the accuracy of the model is highly relevant. The accuracy is affected by the aforementioned mentioned general assumptions, the accuracy of the empirical correlation for the linear motor and frequency inverter, the assumptions for the friction force, and further simplifications.

For the model's validation, the prototype's measurement data is used. The experiments are conducted as described in Section 2. The position of the piston is measured using the laser distance measuring equipment.

The simulations are carried out with the same control voltage for the frequency inverter as the experiment for a given air flow rate to validate the empirical approach used for frequency inverter and linear motor. The moment when the piston starts moving forward is set as $t = 0$ s and the validation is only done for that part of the experiment as this is the important part of the measurement.

In Figure 9, the piston velocity in simulation and experiment is shown over time for a volume flow rate of 116 m^3/h and a control voltage of 1.95 V. In the first 0.5 s, the linear motor is not active due to the rising ramp of the frequency inverter and the piston is accelerated by the differential pressure at the piston. In this time frame, the simulation shows a lower acceleration for the piston than in the measurement. When the linear motor is active, the simulated piston acceleration is higher than the measured acceleration of the piston. In simulation and measurement, the maximum piston velocity is similar. The decrease in piston velocity due to the increasing resistance force of the connection cable shows a similar behavior in simulation and measurement. Overall, a relatively good accordance of the simulated and measured piston velocity for a volume flow rate of 116 m^3/h is achieved.

The differential pressure at the piston over the time for the simulation and the measurement is shown in Figure 10 for a volume flow rate of 116 m^3/h and a control voltage of 1.95 V. The differential pressure around the piston between simulation and measurement shows a time offset. The simulation has an immediate differential pressure drop when the piston accelerates while the measurement data shows a delayed decrease in differential pressure. The slope of differential pressure decrease is the same for simulation and measurement. Except for the time offset, the simulation and measurement show a good agreement in behavior of the differential pressure at the piston.

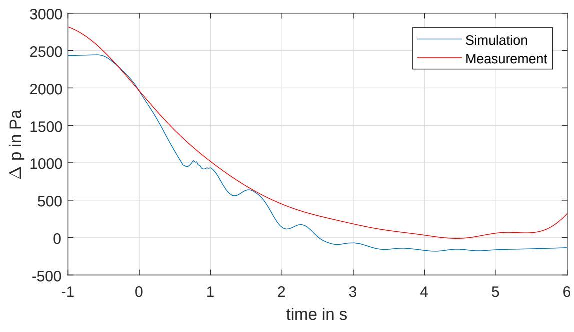


Figure 10. Comparison of the differential pressure at the piston over the time in the model and the measured data with a time offset $t_{\mathrm{off}} = -1.9$ s for a volume flow rate of 116 m^3/h and a control voltage of 1.95 V

For validation of lower volume flow rates, the simulation and measurement data for $V = 65\,\mathrm{m}^3/\mathrm{h}$ is shown in Figure 11. The simulation again has a lower piston acceleration at the beginning and a higher piston acceleration when the linear motor is active. After reaching the maximum piston velocity simulation and experimental data have an similar decrease in piston velocity. Therefore, a good accordance between measurement and simulation is also achieved for lower volume flow rates.

The difference in acceleration between simulation and measurement data due to the air flow is caused partly by the modeling of the pistons resistance force and the approach of using a constant roll resistance coefficient. This may lead to the described difference. The accuracy of the linear motor model may be increased by using an empirical approach which describes the dynamic part of the acceleration in more detail.

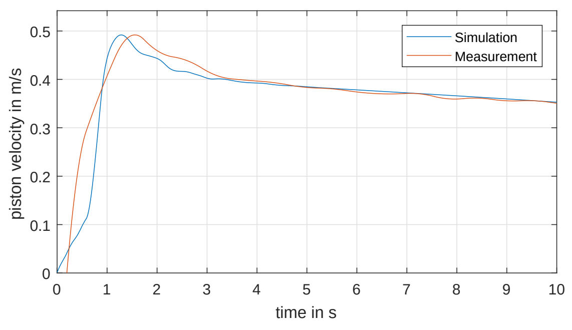


Figure 11. Comparison of the piston velocity over the time in the model and the measured data for a volume flow rate of $65\,m^3/h$ and a control voltage of $1.25\,V$

6 Optimized control voltage trajectory

The model is used to optimize the control voltage trajectory of the frequency inverter to increase the available measuring time. The differential pressure at the piston is used as measure for the available measuring time.

There are different aspects to consider when optimizing the control voltage. The pressure drop at the piston results in a lower density downstream of the piston and accordingly less mass in the measuring cylinder. In order to increase the pressure and density downstream of the piston an overshoot in piston velocity is necessary. Accordingly, an overshoot in control voltage needs to be applied. The piston's resistance increases while it moves downstream due to the connection cable. Therefore, an increase in driving force is necessary.

In Figure 12 the trajectory of the optimized and non-optimized control voltage over time for a volume flow rate of $116\,m^3/h$ is shown. The control voltage in the optimized case has an maximum value of $2.8\,V$ and thus has an overshoot of $0.85\,V$. It decreases to a value close to the non-optimized control voltage and then increases with a constant slope.

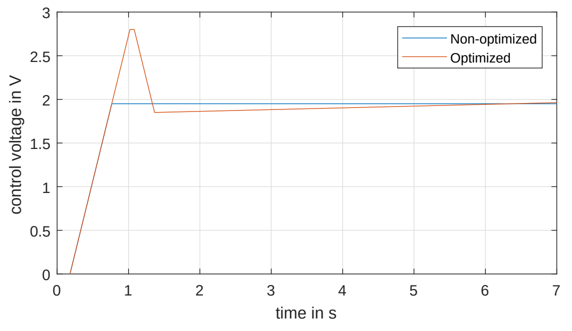


Figure 12. Comparison of the optimized and non-optimized frequency inverter control voltage over time for a volume flow rate of $116\,m^3/h$

The piston velocity over time is shown in Figure 13 for both regarded cases. The piston velocity is the same as long as the control voltage is increasing and has the same value. At the control voltage overshoot the piston velocity for the optimized case overshoots, too. After the overshoot the piston velocity in the optimized case reaches the air flow velocity much earlier than in the non-optimized case. Furthermore, the piston velocity in the optimized case stays close to the flow velocity while the piston velocity in the non-optimized case decreases continuously.

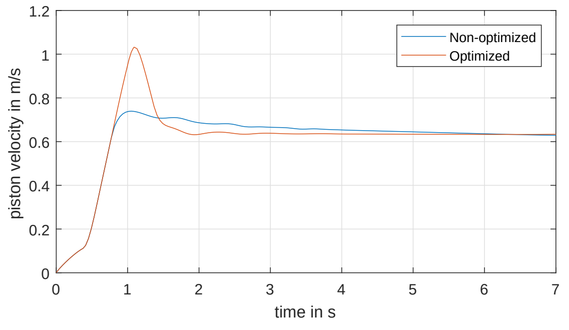


Figure 13. Comparison of the piston velocity over time for the optimized and non-optimized frequency inverter control voltage for a volume flow rate of $116\,m^3/h$

The differential pressure at the piston over time for the non-optimized and optimized case is shown in Figure 14. At the beginning of the measurement the differential pressure at the piston is the same for both regarded cases. The differential pressure at the piston decreases faster to the desired value of $\Delta p = 0\,Pa$ than in the non-optimized case. Furthermore, in the optimized case the differential pressure stays in a close range around $\Delta p = 0\,Pa$ during the measurement. In comparison, the differential pressure at the piston in the non-optimized case is increasing.

The maximum permitted differential pressure at the piston during the measuring time is set to $\Delta p = 50\,Pa$. With this restriction the measuring time of the optimized case is $4.5\,s$ long while in the non-optimized case the measuring time is about $2.5\,s$. The measuring time can be increased by $80\,\%$ using the optimized control voltage trajectory.

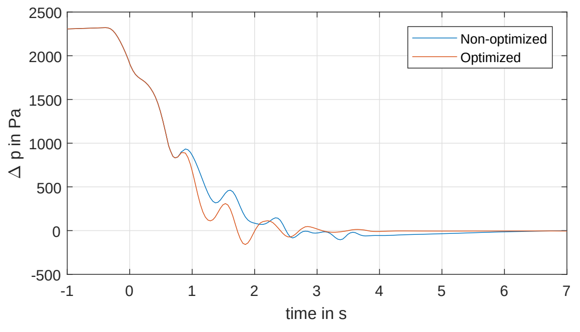


Figure 14. Comparison of the differential pressure at the piston over time for the optimized and non-optimized frequency inverter control voltage for a volume flow rate of $116\,m^3/h$

7 Conclusion and Outlook

A model of the Flow Comparator is implemented in Modelica®. The model is successfully verified and validated against measurement data. With the model the frequency inverter control voltage trajectory is optimized to maximize the available measuring time. With this simple optimization, the measuring time could be increased by 80 % in the model. This result of optimization will allow to extend the upper limits of flow rate usable for calibrations. Furthermore, the possibility to gather detailed information about pressure and temperature development at arbitrary chosen locations in the system with high time resolution enables much better and more reliable statements about the accuracy of flow rate measurement with this system.

The model uses an empirical approach to model the linear motor's force. In future work, the linear motor should be modeled using physically based equations. Additionally, it will be essential to extend the model by heat transfer from the motor components to the gas to complete the modeling of the overall thermodynamic performance of the piston prover.

Furthermore, the optimization of the frequency inverter control voltage should be done using a more detailed approach. For this the position depending resistance as well as the increase of resistance due to the connection cable weight needs to be measured exactly.

In future work the check valve model and the model of the leakage between piston and cylinder can be improved by using a greater number of measurements to describe their behavior.

References

International Organization for Standardization. DIN EN ISO 9300: Durchflussmessung von Gasen mit Venturidüsen bei kritischer Strömung. 2005.

B. Mickan, R. Kramer, and V. Strunck. Transient response of turbine flow meters during the application at a high pressure piston prover. In *15th Flow Measurement Conference (FLOMEKO)*. Linköping University Electronic Press, 2010.

Physikalisch-Technische Bundesanstalt PTB. Prüfschein der Rohrprüfstrecke. 1991.

Physikalisch-Technische Bundesanstalt PTB. PTB Mitteilungen. *Special Issue Volume 119 no.1*, 2009.

G. Schmitz and A. Aschenbrenner. Experience with a Piston Prover as the New Primary Standard of the Federal Republic of Germany in High-Pressure Gas Metering. In *Proceedings of the 18th World Gas Conference, Berlin, 8.-12.7.1991.*

M. von der Heyde, G. Schmitz, and B. Mickan. Modeling of the German National Standard for High Pressure Natural Gas Flow Metering in Modelica. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linköping University Electronic Press, sep 2015. doi:10.3384/ecp15118663. URL `https://doi.org/10.3384/ecp15118663`.

Transient modelling and simulation of a double-stage Organic Rankine Cycle

Tim Eller[1] Florian Heberle[1] Dieter Brüggemann[1]

[1]Institute of Engineering Thermodynamics and Transport Processes (LTTT), Center of Energy Technology (ZET), University of Bayreuth, 95440 Bayreuth, Germany; {tim.eller, florian.heberle, brueggemann}@uni-bayreuth.de

Abstract

Geothermal energy is a renewable resource for power and heat production. For low enthalpy reservoirs the geothermal energy is usually converted to electricity by an Organic Rankine Cycle (ORC). The efficiency and profitability of these power plants can be increased by combined heat and power production. In this study, a dynamic model of a double-stage ORC power plant is developed to investigate and evaluate geothermal combined heat and power plant concepts. The simulation model is validated by operational data of a real geothermal power plant in the South of Germany. For the validation, the relative root mean squared error (RRMSE) is used. In addition, the coefficient of correlation is calculated to evaluate the dynamic behavior. The results show that the electrical power output of the power plant can be predicted by an RRMSE of 3.9 %. The coefficient of correlation is 0.99 and shows that the model is capable to predict the dynamic behavior of the power plant.

Keywords: transient simulation, Organic Rankine Cycle, geothermal heat and power production

1 Introduction

Geothermal energy is a renewable resource for low carbon heat and power production. In binary systems, the thermal power of the brine is usually converted to electricity by Organic Rankine Cycles (ORC). A previous study (Heberle et al., 2016) shows that the efficiency and profitability of these power plants can be increased by an additional heat supply. Because of the fluctuating heat demand, the power plant is driven more often in part load conditions. For that reason, a dynamic model of a double-stage ORC power plant is developed to evaluate different power plant concepts for combined geothermal heat and power production.

In the literature, several dynamic ORC models are presented for waste heat recovery (WHR) from engines. Huster et al. (2018) modelled a one-stage ORC for WHR in a diesel truck. For the simulation, the software gPROMS is used and the model is validated against measurement data. The results show that the initialization process of the model is a challenging task and that the dynamics in the heat exchangers are dominated by the pressure level. Jiaxin Ni et al. (2017) developed also an ORC for waste heat recovery (WHR) from diesel engines using the software Dymola. The study shows that the dynamics of the system can be damped by the integration of an intermediate oil cycle. Bin Xu et al. (2017) developed a model for WHR from diesel engines in MATLAB/Simulink and showed that the vapor temperature and the evaporation pressure can be predicted with 2 % and 3 %, respectively.

Next to WHR dynamic ORC models are also developed for solar and geothermal applications. Baccioli et al. (2017) built up a dynamic model for an ORC with compound parabolic solar collectors and developed a control strategy to drive the system without a thermal energy storage. Proctor et al. (2016) developed a one-stage ORC simulation model for geothermal power production and validated the model against measurement data with a standard deviation of 1.4 %. The model will be used to test potential improvements to the control system of the power plant. To sum up, so far only small scale dynamic ORC models are developed.

In this study, a dynamic model of a double-stage ORC is developed with the software Dymola to investigate potential plant concepts for geothermal combined heat and power production. The simulation model is described and in chapter 3 the results of the validation are presented.

2 Methodology

In this section the double-stage ORC concept is introduced and the dynamic modelling of the cycle components is presented. For modelling and simulation the software Dymola (Dassault Systèmes, 1992-2004) is combined with the Modelica based library ThermoCycle (Quoilin et al., 2014). For the calculation of the fluid properties, the software CoolProp (Bell et al., 2014) is used.

2.1 Double-stage ORC

In this study, a double-stage ORC is considered based on a real operating power plant in the German Molasse

Basin. A scheme of the power plant and its components is shown in Figure 1.

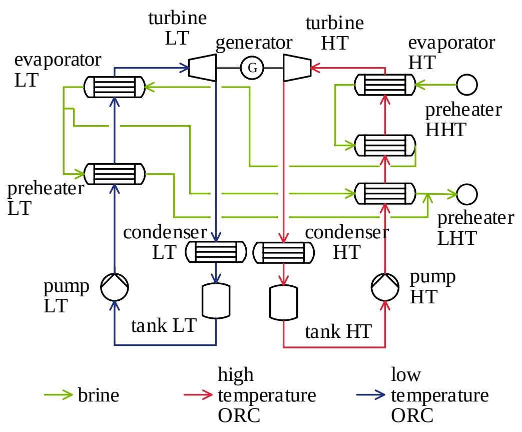


Figure 1. Scheme of the considered double-stage Organic Rankine Cycle.

The double-stage ORC consists of two modules, a high temperature (HT) and a low temperature (LT) ORC. In both modules, R245fa is used as working fluid. Regarding the cycle components, each ORC contains a pump, at least one preheater, an evaporator, a turbine and a condenser. The thermal water enters the HT ORC with a temperature of 138 °C und and a mass flow rate of 120 kg/s. Firstly, heat is supplied to the HT-ORC and then to the LT-ORC. A detailed *T*-$\dot{H}$-diagram of the power plant is shown in Figure 2. Table 1 summarizes some characteristic data for the heat exchangers and the rotating equipment at the design point.

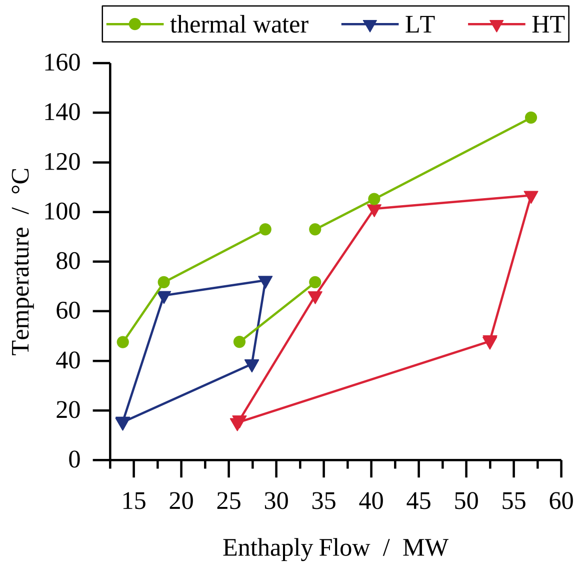


Figure 2. *T*-$\dot{H}$-diagram of the considered double-stage Organic Rankine Cycle.

Table 1. Characteristic data of the considered power plant. (Heberle et al., 2015)

parameter	value
rotating equipment	
LT pump isentropic efficiency	78.4 %
HT pump isentropic efficiency	76.7 %
HT turbine isentropic efficiency	82.7 %
HT turbine isentropic efficiency	88.3 %
generator efficiency	98.0 %
heat exchanger areas	
LT-preheater	201.0 m²
LT-evaporator	741.4 m²
LHT-preheater	270.4 m²
HHT-preheater	277.6 m²
HT-evaporator	741.4 m²
LT-condenser	7512.0 m²
HT-condenser	3756.0 m²

2.2 Modelling

In principle, two types of components have to be modelled to simulate a double-stage ORC system: turbomachines (pumps, turbines) and heat exchangers (preheaters, evaporators, condensers).

Turbomachines

Several previous investigations of dynamic ORC models show that the time constants in the turbomachines are relatively low compared to those of the heat exchangers. Therefore, the pump and the turbine can be modelled as quasi-stationary components (Quoilin, 2011; van Putten and Colonna, 2007; Wei et al., 2008).

For the pump the exhaust enthalpy is calculated according to the following equation:

$$h_{out} = h_{in} + \frac{p_{out} - p_{in}}{\eta_s \rho_{in}} \qquad (1)$$

The isentropic efficiency of the pump η_s depends on the pumped volume flow rate. For the simulation, the characteristic curve of the manufacturer data sheet is implemented for the LT- and the HT-ORC pump, respectively. The normalized curves are shown in Figure 3.

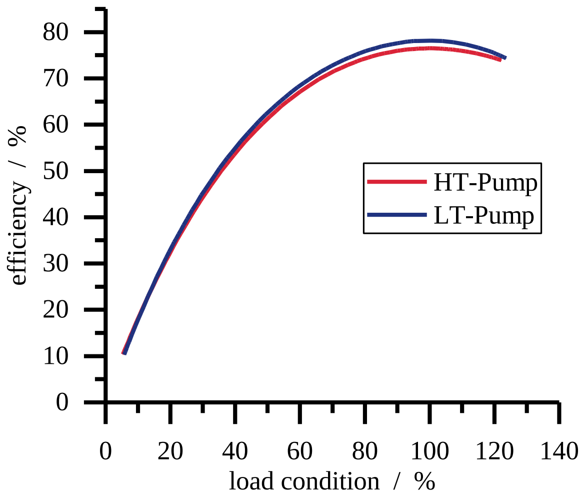


Figure 3. Characteristic curve of the pump isentropic efficiency for the HT- and LT-ORC.

The turbine is modelled based on Stodola's law:

$$\dot{m}_{in} = K\sqrt{\rho_{in} p_{in}\left[1-\left(\frac{p_{out}}{p_{in}}\right)^2\right]} \tag{2}$$

The coefficient *K* is calculated for the HT and the LT ORC by the respective nominal turbine inlet and outlet conditions.

The isentropic efficiency of the turbine depends on the volume flow rate at the turbine outlet and on the enthalpy difference utilized in the turbine (Milora and Tester, 1977). Both parameters are influenced by part load operation. In addition, in the considered power plant air-cooled condensers are used. Therefore, the condensing pressure varies during the day according to the ambient temperature. This affects the turbine outlet pressure and thereby the utilized enthalpy difference. For that reason, a quasi-stationary model of the isentropic efficiency is implemented in the turbine based on the approach of Ghasemi et al. (2014):

$$\eta_s = \eta_{s,nom} r_h r_v \tag{3}$$

The nominal isentropic efficiency $\eta_{s,nom}$ is corrected by two factors: r_h takes into account the off-design enthalpy difference and is calculated by

$$r_h = \left\{\left[\left(1.398 r_T - 5.425\right) r_T + 6.274\right] r_T - 1.866\right\} r_T + 0.619 \tag{4}$$

with

$$r_T = \sqrt{\frac{\left(h_{turbine,in} - h_{turbine,out}\right)}{\left(h_{turbine,in} - h_{turbine,out}\right)_{nom}}}\,. \tag{5}$$

r_V takes into account the off-design volume flow rate at the turbine outlet and is given by

$$r_v = \left\{\left[\left(-0.21 r_{VT} - 1.117\right) r_{VT} - 2.533\right] r_{VT} - 2.588\right\} r_{VT} + 0.038 \tag{6}$$

with

$$r_{VT} = \frac{\dot{V}}{\dot{V}_{nom}}\,. \tag{7}$$

Heat exchangers

The dynamic models for the heat exchangers are built up in three steps according to Quoilin et al. (2011):

1. At first, a stationary model is built up with detailed correlations for the heat transfer and the pressure drop depending on the geometry.
2. In the next step, the stationary model is simulated at the design point and nominal values for the heat transfer coefficient α_{nom} and the pressure drop Δp_{nom} are calculated.
3. In the third step, the detailed heat and pressure drop correlations are simplified and a dynamic heat transfer coefficient and pressure drop is calculated based on the nominal values. Exemplarily, the dynamic heat transfer coefficient α of the working fluid in the preheaters can be calculated according to equation (8):

$$\alpha = \alpha_{nom}\left(\frac{\dot{m}}{\dot{m}_{nom}}\right)^n \tag{8}$$

The preheaters are shell and tube heat exchangers with double-segmental baffles and two passes for the hot and the cold side, respectively. The thermal water flows in the tubes and the working fluid in the shell. For modelling of heat exchangers, the most common approaches are the finite volume approach and the moving boundary approach (Jensen, 2003). For the simulation of the preheaters, the finite volume approach is used.

The heat transfer coefficient of the thermal water in the tubes is calculated by Gnielinski (2013). For the working fluid on the shell side an adapted version of the correlation of Bell-Delaware is used (Milcheva et al., 2017). For the pressure drop on the hot side a correlation of Kast and Nirschl (2013) is implemented and for the working fluid the pressure drop is calculated by Taborek et al. (Hewitt, 2008).

The evaporators in the considered power plant are designed as kettle boilers. The working fluid on the shell side is heated by the thermal water in the tube bundle. On the tube side there are four passes.

Since the dynamics on the shell side can not be covered accurately by the finite volume or moving boundary approach, a two-volume model for the evaporator is developed in ThermoCycle according to Pili et al. (2017). For the tube side the finite volume approach is used.

The heat transfer coefficient for pool boiling on tube bundles is calculated by Macchi and Astolfi (2017)

$$\alpha_{bundle} = \alpha_{nb,1} F_b + \alpha_{nc} \,. \tag{9}$$

F_b takes into account the effect of convective boiling and is calculated by Taborek et al. (Hewitt, 2008):

$$F_b = 1.0 + 0.1\left[\frac{0.785 D_b}{0.866 (p_t / D_o)^2 D_o} - 1.0\right]^{0.75} \tag{10}$$

According to Welzl et al. (2018) for the pool boiling heat transfer coefficient $\alpha_{nb,1}$ of R245fa the correlation of Cooper (1984) is used. The convective boiling heat transfer coefficient α_{nc} is assumed to be 250 W/m²K according to Macchi and Astolfi (2017). For the heat transfer in the vapor region the Bromley equation (Bromley, 1950) is implemented. The pressure drop is calculated by the static pressure drop since the pressure drop in kettle boilers is dominated by the static part (Thome, 2004). For the thermal water, the same correlations for the heat transfer coefficient and pressure drop are used as for the preheaters.

As mentioned above, in the considered power plant air-cooled condensers are implemented. The working fluid flows in tube bundles with two passes and the air in cross-flow over the tube bundles. On the working fluid side the heat transfer for one-phase regions (liquid and vapor) is calculated by Gnielinski (2013). For the two-phase region Cavallini et al. (2006) is used. For the pressure drop Kast and Nirschl (2013) is implemented for one phase regions and for the two-phase region Friedel is used. The heat transfer coefficient on the shell side is calculated by Haaf et al. (Steimle and Plank, 1988).

2.3 Validation parameters

The developed dynamic models are validated against operational data of a real geothermal heat plant. The validation takes place in two steps.
At first, the relative root mean squared error (RRMSE) (Despotovic et al., 2016) is calculated as an indicator for the quantitative quality of the simulation results by

$$RRMSE = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(R_i)^2}}{\bar{x}} \,, \tag{11}$$

where N is the number of intervals and R the difference between the measured value x and the simulated value y.

The RRMSE, however, is not able to take into account the dynamic behavior of the model. Therefore, in a second step the coefficient of correlation ρ is calculated:

$$\rho = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2 \sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{12}$$

The coefficient of correlation is used for time series analysis and can range from -1 to +1. A value of +1 means that both time histories are identical in shape. (Sarin et al., 2010) Therefore, the coefficient of correlation evaluates the dynamic behavior of the model compared to the operational data.

3 Results

For the validation of the double-stage ORC a period of 24 hours in steps of one minute is simulated. The validation results are presented in Table 2.

Table 2. Validation results of the cycle components.

parameter	RRMSE [%]
HT-ORC	
pump outlet pressure	3.5
pump outlet volume flow rate	17.0
turbine inlet pressure	4.8
turbine outlet pressure	3.6
HHT preheater inlet temperature	0.4
HHT preheater outlet temperature	0.9
HHT preheater temperature difference	6.2
evaporator outlet temperature	0.4
tank temperature	1.8
LT-ORC	
pump outlet pressure	4.2
pump outlet volume flow rate	5.3
turbine inlet pressure	1.4
turbine outlet pressure	4.7
LT preheater outlet temperature	0.5
evaporator outlet temperature	0.1
tank temperature	1.8
Thermal water	
HT evaporator temperature difference	11.0
HHT preheater inlet temperature	0.9
HHT preheater outlet temperature	0.7
HHT preheater temperature difference	5.1
LT evaporator inlet temperature	0.7
LT evaporator outlet temperature	0.5
LT evaporator temperature difference	5.0
LHT preheater inlet temperature	0.5
LHT preheater outlet temperature	1.2
LHT preheater temperature difference	12.8
LT preheater outlet temperature	0.8
LT preheater temperature difference	6.3
injection temperature	1.1

The volume flow rate and the temperature of the geothermal fluid, the electrical power consumption of the fans and the ambient temperature are used as inputs for the simulation.

The RRMSE is on average 3.6 %. For the pressure and the temperatures, the RRMSE is lower than 5 %. Except for the temperature differences and the volume flow rates in LT- and HT-ORC module the RRMSE is higher than 5 %. According to the manufacturer for the volume flow rates in the ORC-modules the uncertainties of the integrated flow rate sensors are responsible for the deviations (Heberle et al., 2015). Regarding the temperature differences, the reason for the deviation is the uncertainty in the measurement of the volume flow rate of the geothermal fluid.

In Figure 4 the results for the evaporating pressure of the HT-cycle are presented. The deviation between simulation and operational data is quantified by an RRMSE of 4.8 %. Regarding the coefficient of correlation is 0.97 and shows that the simulation model can reproduce the dynamic behavior.

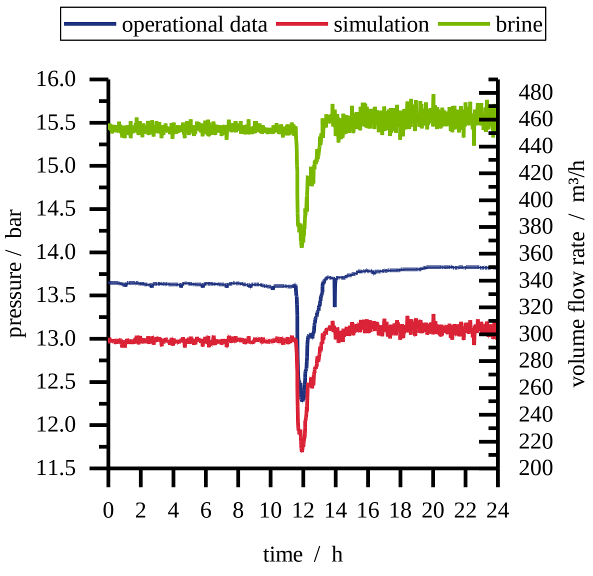


Figure 4. Validation results for the evaporating pressure of the HT-ORC.

Figure 5 shows the validation results for the reinjection temperature of the geothermal fluid, which is an indicator for the heat supplied to the ORC. The RRMSE is 1.1 %. In addition, the dynamic behavior is evaluated by a coefficient of correlation of 0.99 and shows that the simulation and the operational data are almost identical in shape.

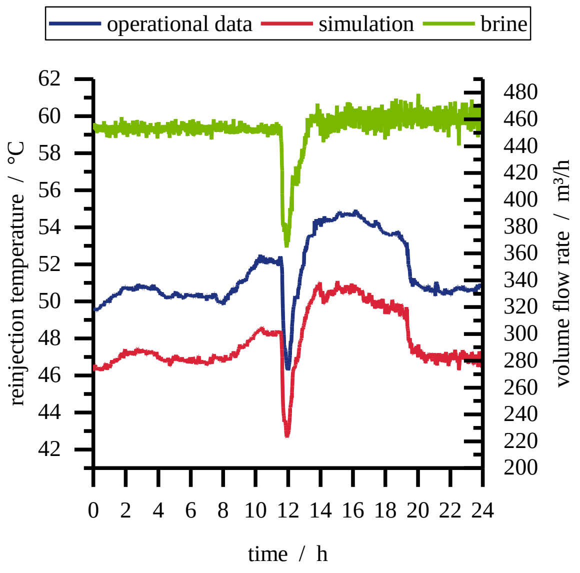


Figure 5. Validation results for the reinjection temperature of the geothermal fluid.

For the evaluation of the whole system, the electrical power output of the generator is used as the validation parameter. The results are shown in Figure 6. The RRMSE is 3.9 % and the coefficient of correlation is 0.99. Therefore, the simulation model can predict the dynamic behavior.

As mentioned, the measurement of the volume flow rate of the thermal water is connected to high uncertainties. For that reason, the simulated electrical power output of the generator is lower than the real power output even though no heat losses to the ambient are considered in the simulation model.

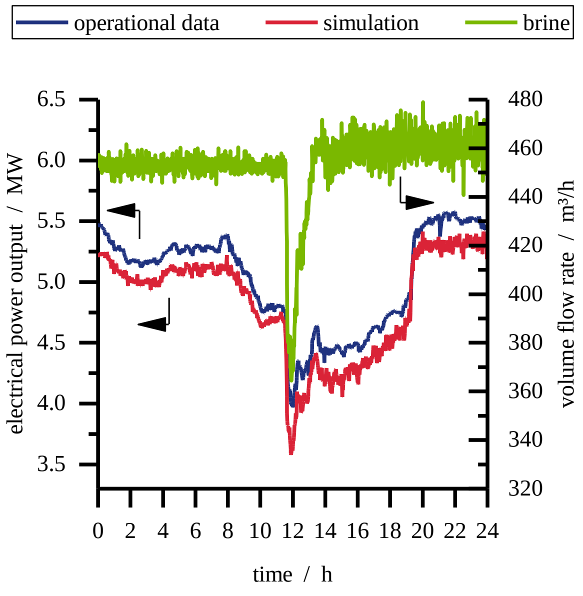


Figure 6. Validation results of the double-stage Organic Rankine Cycle.

The dynamic model is developed to investigate different concepts for geothermal heat and power production. Geothermal reservoirs usually provide a fixed volume flow rate and temperature. Therefore, the parameters of the thermal water can be defined in the model so that the results are not affected by measurement uncertainties of the thermal water volume flow rate. For that reason, the deviation regarding the generator output is acceptable. Due to the fluctuating heat demand of district heating networks, the power plant is driven more often in part load conditions. Therefore, it is necessary to predict the dynamic behavior of the power plant accurately. As the results show, the dynamics can be reproduced by the developed simulation model.

4 Conclusion

In this study, a transient simulation model of a double-stage ORC is developed and validated by operational data of a real power plant in the German Molasse Basin. The model is built up in Dymola based on the ThermoCycle library.

The results show a RRMSE of 3.6 % on average The pressure and temperatures can be predicted by an RRMSE lower than 5 %. For the whole double-stage ORC power plant the electrical output of the generator can be predicted by 3.9 %. The coefficient of correlation is 0.99 and shows that the simulation model can reproduce the dynamics of the real power plant.

In future work, based on the dynamic simulation model different geothermal combined heat and power plant concepts are investigated and evaluated by annual return simulations. For the district heating network heat demand profiles based on real heat plant data will be implemented. In addition, different peak loads as well as supply and return temperatures are investigated.

Acknowledgements

The funding from the Bavarian State Ministry for Education, Science and the Arts in the framework of the project “Geothermie-Allianz Bayern” is gratefully acknowledged.

References

Baccioli, A., Antonelli, M., and Desideri, U.: Dynamic modeling of a solar ORC with compound parabolic collectors: Annual production and comparison with steady-state simulation, Energy Conversion and Management, 148, 708–723, doi:10.1016/j.enconman.2017.06.025, 2017.

Bell, I. H., Wronski, J., Quoilin, S., and Lemort, V.: Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp, Industrial & engineering chemistry research, 53, 2498–2508, doi:10.1021/ie4033999, 2014.

Bromley, L. A.: Heat Transfer in Stable Film Boiling, Chemical engineering progress, 221–227, 1950.

Cavallini, A., Col, D. D., Doretti, L., Matkovic, M., Rossetto, L., Zilio, C., and Censi, G.: Condensation in Horizontal Smooth Tubes: A New Heat Transfer Model for Heat Exchanger Design, Heat Transfer Engineering, 27, 31–38, doi:10.1080/01457630600793970, 2006.

Cooper, M. G.: Heat Flow Rates in Saturated Nucleate Pool Boiling-A Wide-Ranging Examination Using Reduced Properties, in: Advances in heat transfer, Hartnett, J. P., and Irvine, T. F. (Eds.), Advances in Heat Transfer, 16, Academic Press, Orlando, London, 157–239, 1984.

Dassault Systèmes: Dymola - Dynamic Modeling Laboratory, 1992-2004.

Despotovic, M., Nedic, V., Despotovic, D., and Cvetanovic, S.: Evaluation of empirical models for predicting monthly mean horizontal diffuse solar radiation, Renewable and Sustainable Energy Reviews, 56, 246–260, doi:10.1016/j.rser.2015.11.058, 2016.

Friedel, L.: Improved friction pressure drop correlations for horizontal and vertical two-phase pipe flow, in: European Two-phase Flow Group Meeting 1979.

Ghasemi, H., Sheu, E., Tizzanini, A., Paci, M., and Mitsos, A.: Hybrid solar–geothermal power generation: Optimal retrofitting, Applied Energy, 131, 158–170, doi:10.1016/j.apenergy.2014.06.010, 2014.

Gnielinski, V.: Durchströmte Rohre, in: VDI-Wärmeatlas. (in German), 11., bearb. und erw. Aufl., VDI-Buch, Springer Vieweg, Berlin, 785–792, 2013.

Heberle, F., Eller, T., and Brüggemann, D.: Thermoeconomic evaluation of one-and double-stage ORC for geothermal combined heat and power production., in: Proceedings of European Geothermal Congress 2016, Paper-ID T-PO-148, Strasbourg (France), September 2016, Strasbourg (France), September 2016, 2016.

Heberle, F., Jahrfeld, T., and Brüggemann, D.: Thermodynamic Analysis of Double-Stage Organic Rankine Cycles for Low-Enthalpy Sources Based on a Case Study for 5.5 MWe Power Plant Kirchstockach (Germany), in: Proceedings World Geothermal Congress 2015, Melbourne, Australia, 19-25 April, 2015.

Hewitt, G. F.: Thermal and hydraulic design of heat exchangers, Heat exchanger design handbook (HEDH), 3, 2008.

Huster, W. R., Vaupel, Y., Mhamdi, A., and Mitsos, A.: Validated dynamic model of an organic Rankine cycle (ORC) for waste heat recovery in a diesel truck, Energy, 151, 647–661, doi:10.1016/j.energy.2018.03.058, 2018.

Jensen, J. M.: Dynamic modeling of thermo-fluid systems: With focus on evaporators for refrigeration, MEK-ET-PHD, 2003-01, Department of Mechanical Engineering, Technical University of Denmark, Lyngby, 152 pp., 2003.

Kast, W. and Nirschl, H.: Druckverlust in durchströmten Rohren, in: VDI-Wärmeatlas. (in German), 11., bearb. und erw. Aufl., VDI-Buch, Springer Vieweg, Berlin, 2013.

Macchi, E. and Astolfi, M. (Eds.): Organic rankine cycle (ORC) power systems: Technologies and applications, Woodhead Publishing series in energy, number 107, Woodhead Publishing, Duxford, UK, 2017.

Milcheva, I., Heberle, F., and Brüggemann, D.: Modeling and simulation of a shell-and-tube heat exchanger for Organic Rankine Cycle systems with double-segmental baffles by adapting the Bell-Delaware method, Applied Thermal Engineering, 126, 507–517, doi:10.1016/j.applthermaleng.2017.07.020, 2017.

Milora, S. L. and Tester, J. W.: Geothermal energy as a source of electric power. Thermodynamic and economic design criteria, 2. printing, MIT Press, Cambridge Massachusetts, 186 pp., 1977.

Ni, J., Wang, Z., Zhao, L., Zhang, Y., Zhang, Z., Ma, M., and lin, S.: Dynamic simulation and analysis of Organic Rankine Cycle system for waste recovery from diesel engine, Energy Procedia, 142, 1274–1281, doi:10.1016/j.egypro.2017.12.485, 2017.

Pili, R., Spliethoff, H., and Wieland, C.: Dynamic Simulation of an Organic Rankine Cycle—Detailed Model of a Kettle Boiler, Energies, 10, 548, doi:10.3390/en10040548, 2017.

Proctor, M. J., Yu, W., Kirkpatrick, R. D., and Young, B. R.: Dynamic modelling and validation of a commercial scale geothermal organic rankine cycle power plant, Geothermics, 61, 63–74, doi:10.1016/j.geothermics.2016.01.007, 2016.

Quoilin, S.: Sustainable Energy Conversion Through the Use of Organic Rankine Cycles for Waste Heat Recovery and Solar Applications., Dissertation, Faculty of Applied Science, University of Liège, Liège, 2011.

Quoilin, S., Desideri, A., Wronski, J., Bell, I., and Lemort, V.: ThermoCycle: A Modelica library for the simulation of thermodynamic systems, in: the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden, March 10-12, 2014, Linköping Electronic Conference Proceedings, Linköping University Electronic Press, 683–692, 2014.

Sarin, H., Kokkolaras, M., Hulbert, G., Papalambros, P., Barbat, S., and Yang, R.-J.: Comparing Time Histories for Validation of Simulation Models: Error Measures and Metrics, Journal of Dynamic Systems, Measurement and Control, 132, 1–10, doi:10.1115/1.4002478, 2010.

Steimle, F. and Plank, R. (Eds.): Handbuch der Kältetechnik, (in German), Springer, Berlin, 730 pp., 1988.

Thome, J. R.: Engineering Data Book III, Wolverine Tube, Inc., Lausanne, 2004.

van Putten, H. and Colonna, P.: Dynamic modeling of steam power cycles: Part II – Simulation of a small simple Rankine cycle system, Applied Thermal Engineering, 27, 2566–2582, doi:10.1016/j.applthermaleng.2007.01.035, 2007.

VDI-Wärmeatlas, (in German), 11., bearb. und erw. Aufl., VDI-Buch, Springer Vieweg, Berlin, 1760 pp., 2013.

Wei, D., Lu, X., Lu, Z., and Gu, J.: Dynamic modeling and simulation of an Organic Rankine Cycle (ORC) system for waste heat recovery, Applied Thermal Engineering, 28, 1216–1224, doi:10.1016/j.applthermaleng.2007.07.019, 2008.

Welzl, M., Heberle, F., and Brüggemann, D.: Experimental evaluation of nucleate pool boiling heat transfer correlations for R245fa and R1233zd(E) in ORC applications, Renewable Energy, doi:10.1016/j.renene.2018.09.093, 2018.

Xu, B., Rathod, D., Kulkarni, S., Yebi, A., Filipi, Z., Onori, S., and Hoffman, M.: Transient dynamic modeling and validation of an organic Rankine cycle waste heat recovery system for heavy duty diesel engine applications, Applied Energy, 205, 260–279, doi:10.1016/j.apenergy.2017.07.038, 2017.

Nomenclature

Symbols

D	*diameter*
h	specific enthalpy
m	mass flow rate
N	number of intervals
p	pressure
p_t	tube pitch
R	residual
x	measured value
y	simulated value

Greek symbols

α	heat transfer coefficient
η	efficiency
ρ	density

Subscripts

b	bundle
in	inlet
nom	nominal
o	outer
out	outlet
s	isentropic

Superscript

$\bar{}$	mean value

SESSION 6C: TOOLS

A New OpenModelica Compiler High Performance Frontend

Adrian Pop[1] Per Östlund[1] Francesco Casella[2] Martin Sjölund[1] Rüdiger Franke[3]

[1]PELAB - Programming Environments Lab, Dept. of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden, {adrian.pop,per.ostlund,martin.sjolund}@liu.se
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy, francesco.casella@polimi.it
[3]ABB, IAPG-A26, Kallstadter Str. 1, 68309 Mannheim, Germany, ruediger.franke@de.abb.com

Abstract

The equation-based object-oriented Modelica language allows easy composition of models from components. It is very easy to create very large parametrized models using component arrays of models. Current open-source and commercial Modelica tools can with ease handle models with a hundred thousand equations and a thousand states. However, when the system size goes above half a million (or more) equations the tools begin to have problems with scalability. This paper presents the new frontend of the OpenModelica compiler, designed with scalability in mind. The new OpenModelica frontend can handle much larger systems than the current one with better time and memory performance. The new frontend was validated against large models from the ScalableTestSuite library and Modelica Standard Library, with good results.

Keywords: OpenModelica, compiler, flattening, frontend, modelling, simulation, equation-based, scalability

1 Introduction and Motivation

System-level dynamic modelling and simulation is a key activity in modern system engineering design. In parallel to the detailed component design, which is performed using advanced 3D CAD, CFD and FEM software tools, system-level modelling, usually including systems of systems and large numbers of interacting components, allows predicting the dynamic performance of complex systems, which emerges from the interaction of its components.

The Modelica language (Modelica Association, 2017; Fritzson, 2015) is a standardized tool-independent non-proprietary equation-based object-oriented modeling language, which was introduced 20 years ago by the non-profit Modelica Association, with strong links to industry and academia. This language, and the related eco-system of tools, model libraries and the FMI standard (Blochwitz et al., 2011), is ideally suited to system-level modeling of complex, heterogenous and multi-domain cyber-physical systems. It has become a de-facto standard in many industries, most notably the automotive one. The Modelica language is currently supported by about 10 different modeling and simulation software tools; one of them, in particular, the open-source OpenModelica software suite (Fritzson et al., 2018), is the only Modelica tool owned and maintained by a non-profit organization – the Open Source Modelica Consortium (OSMC).

The main applications of Modelica tools so far have been the study of individual systems, such as a car's drivetrain and active suspension and steering control system, a single industrial robot, a single power plant, a single HVDC power link, the air conditioning system of a car, etc. Existing Modelica tools employ strategies and algorithms that are optimized for such system models, whose typical complexity lies in the range of 1000-50000 equations and up to a few thousand state variables. The advent of the internet-of-things paradigm is now fostering the development of innovative very large-scale cyber-physical systems, for example smart grids, or fleets of autonomous vehicles. It is also sparking a renewed interest at the modernization of traditional large-scale systems. A first example is continental-size high-voltage power generation and transmission, which is facing increasing challenges due to the introduction of power electronics equipment and to the increased penetration of intermittent renewable energy sources. A second example is district heating, possibly integrated with heat pumps and distributed power generation in an integrated electrical and thermal smart grid. See (Casella, 2015) for further examples and motivation.

Unfortunately, when Modelica is used to tackle the modelling of large-scale systems with sizes exceeding the ones mentioned above, currently available simulation software that support Modelica fall short at providing adequate performance. The time required to compile the models vastly exceeds what end users typically expect for system level studies, i.e., a few minutes at most. The size of the generated code and the memory requirements for compilers vastly exceed what is normally available on laptops and workstations used for daily work (8-16 GB).

In the last couple of years there have been some pioneering attempts at pushing the boundary of the size of Modelica models that can be handled with reasonable time and effort. In particular, some of our published papers have demonstrated the feasibility of Modelica models of high-voltage power generation and transmission systems (Braun et al., 2017; Casella et al., 2017) and of detailed models of key system components of future nuclear fusion reactors, see (Froio et al., 2017). The size of the largest models handled so far is about 750000 equations, which is about one order of magnitude bigger than the typical size mentioned earlier.

The results were indeed very interesting and sparked a lot of interest in the Modelica community. On the other hand, they clearly showed the limits of current Modelica tool technology, which is rather strained in terms of time and memory requirements at that scale, and that cannot in practice handle models of a size larger than one million equations. Breaking that barrier and achieving the 10 million equations model size goal requires fundamental methodological breakthroughs.

To summarize, the Modelica language has a lot of potential to support the system-level modelling of innovative engineering systems that require large-scale models. However, current Modelica tools have serious limitations as the system size grows.

1.1 Overcoming the Size Barrier with New Efficient Flattening Approach

An important goal of this work is overcoming the size barrier of current Modelica simulation tools, making it possible to efficiently generate fast simulation code for systems of up to 10 million equations, enabling new important applications such as those mentioned earlier, including large-scale networked system models. Overcoming the size barrier to 10 million equations means handling one to two orders of magnitude larger models than what is currently possible with state-of-the-art tools. To keep the total simulation time within reasonable bounds, the time needed for the model compiler to generate executable code should be in the order of minutes in the worst case, and the memory requirements should be fulfilled by the standard memory size available on laptop computers (16 GB), or possibly on engineering workstations for the largest problems (64 GB). The size of the executable code should also be much smaller than what can be achieved today, otherwise most of the simulation time risks to be spent waiting for data to be shuffled back and forth between RAM and CPU cache.

The availability of such a tool will allow to use the Modelica language, its high-level declarative modelling paradigm, to support a wide range of large-scale system design activities, as discussed in the previous sections. To realize these goals, a large new tool development within the OpenModelica tool suite was initiated about two years ago, in particular the development of a new highly performing compiler frontend, reported in this paper. More than half of the OpenModelica model compiler has been re-written and extended, and the software tool architecture significantly enhanced.

Traditionally, when a Modelica compiler is generating the simulation code, the system model is first flattened (expanded), i.e., reduced to a large system of scalar equations, before performing structural analysis and code generation. Although this process allows to combine components belonging to different domains in a straightforward way, this approach is obviously inefficient when there are many similar components in a system model, that only differ by their parameters, because the structural analysis and the generated code will be highly redundant.

Arrays of models, or even multiple instances of the same model, which only differ by the values of their parameters, should not be flattened to their scalar equations, but rather handled in an efficient way throughout the code generation process. Structural analysis and symbolic simplification of models which are instantiated multiple times should be performed only once instead of many times. At the system level, structural analysis of the overall system of equations should use algorithms and methods that consider arrays as symbolic entities instead of breaking them down to individual components. The efficiency of the final code generation process should also be improved so that ideally, if there are 1000 instances of the same component in a model, code should be generated for the equations of one of them only, and then called 1000 times, so as to drastically reduce the code generation time and memory consumption.

Achieving this goal requires fundamental changes to the structure of the Modelica tool with respect to the current state-of-the-art, which is to perform flattening to scalar equations before starting the code generation phase.

2 Related Work

Instantiation and flattening of Modelica is quite complex. Even as of the time of this writing there are open discussion on the Modelica issue tracker about unclear parts of the Modelica specification with regards to flattening. Furthermore, there is no available information on the instantiation and flattening process in the commercial Modelica tools – this is only available for the two open-source Modelica tools available: OpenModelica and JModelica.org.

JModelica.org is based on JastAdd (Hedin and Magnusson, 2003), a Java based meta-compilation system that supports Reference Attribute Grammars (RAGs). The instantiation and flattening in JModelica.org is detailed in (Åkesson et al., 2010). The process is similar to the one in OpenModelica. The instance ASTs (abstract syntax trees) are created from source ASTs and data is referenced using inter-AST references. From the instance ASTs trees the Flat ASTs are generated. The difference between OpenModelica and JModelica.org comes down to the fundamental differences between JastAdd and MetaModelica (Pop and Fritzson, 2006). The JastAdd framework computes attributes in the ASTs based on user-defined equations that relate to existing or circular attributes. In MetaModelica we use functional programming via functions and pattern matching to compute these attributes. JastAdd translates to Java, MetaModelica translates to C code. Both frameworks have automatic garbage collectors.

Interested readers can read more about compilers in (Aho et al., 1986). More on functional programming is available in (Hudak, 2000; Milner et al., 1997). Our previous work on boostrapping the OpenModelica compiler can be found in (Sjölund et al., 2014).

3 OpenModelica Compiler New Frontend Architecture

This section details the architecture and design of the new frontend. The new frontend is implemented in modern MetaModelica 3.0 which combines Modelica features with functional languages features. The implementation consists of 65 MetaModelica packages or uniontypes defining encapsulated data structures and functions that operate on the defined data.

3.1 New Frontend Typical File Structure

The new frontend uses the full capabilities of MetaModelica 3.0 which simplifies the code, control flow and architecture.

Data structures are defined using uniontypes and records. For example the flat model obtained after instantiation and flattening was performed is defined as below.

```
encapsulated uniontype NFFlatModel
  import Equation = NFEquation;
  import Algorithm = NFAlgorithm;
  import Variable = NFVariable;

  record FLAT_MODEL
    list<Variable> variables;
    list<Equation> equations;
    list<Equation> initialEquations;
    list<Algorithm> algorithms;
    list<Algorithm> initialAlgorithms;
    Option<SCode.Comment> comment;
  end FLAT_MODEL;

end NFFlatModel;
```

Encapsulation of data definition and functions that work on the defined data is similar to Modelica. Below is a partial definition of a binding in the new frontend together with functions to access or query it.

```
encapsulated package NFBinding
  public
    import Expression = NFExpression;
    import NFInstNode.InstNode;
    import SCode;
    import Type = NFType;
    import NFPrefixes.Variability;
    import Error;
  protected
    import Dump;
  public
    constant Binding EMPTY_BINDING
      = Binding.UNBOUND();

uniontype Binding
  record UNBOUND
  end UNBOUND;

  record UNTYPED_BINDING
    Expression bindingExp;
    // ...
  end UNTYPED_BINDING;

  record TYPED_BINDING
    Expression bindingExp;
    // ...
  end TYPED_BINDING;
```

```
public
  function isBound
    input Binding binding;
    output Boolean isBound;
  algorithm
    isBound := match binding
      case UNBOUND() then false;
      else true;
    end match;
  end isBound;

  function untypedExp
    input Binding binding;
    output Option<Expression> exp;
  algorithm
    exp := match binding
      case UNTYPED_BINDING()
        then SOME(binding.bindingExp);
      else NONE();
    end match;
  end untypedExp;

  function typedExp
    input Binding binding;
    output Option<Expression> exp;
  algorithm
    exp := match binding
      case TYPED_BINDING()
        then SOME(binding.bindingExp);
      else NONE();
    end match;
  end typedExp;

end Binding;
end NFBinding;
```

One can note some of the new features in MetaModelica 3.0:

- does not require verbose listing of all components (or named component access) of the record in the pattern matching (`UNTYPED_BINDING()`)
- accesses record components via the dot notation inside the case (`binding.bindingExp`).
- allows definitions of functions inside uniontypes
- allows definitions and the use of generic datatypes such as trees using redeclare/replaceable types

3.2 Features Relevant to High Performance

The new frontend was carefully designed with performance and scalability in mind.

References (pointers) are used to link component references to their definition scope via lookup and usage scope via application.

Constant evaluation and expression simplification are more restricted compared to the old frontend.

Both arrays of basic types and arrays of models are not expanded until the Scalarization phase (see next section).

Expansion of arrays is currently needed because the backend cannot handle all the cases of non-expanded arrays. See Section 4 on preliminary handling of non-expanded arrays of models in the backend and runtime.

3.3 New Frontend Design

The old OpenModelica frontend builds a DAE structure (flattened Modelica code) directly from the SCode structure (simplified parsed abstract syntax tree) for a model.

This means that it takes one component and flattens it to list of variables and equations for that single component before continuing with the next component. This flattening process involves doing instantiation, scalarization of arrays, typing, and so on.

Components in Modelica models often have dependencies on other components though, and the approach taken by the old frontend means that components sometimes need to be partially or fully flattened out of order. This has made it hard to implement certain features, such as redeclares, and has also led to a lot of superfluous flattening where parts of the model are flattened multiple times.

One of the driving forces in the design of the new frontend has therefore been to find ways to break dependencies between the various frontend phases. Instead of being component-focused like the old frontend it has instead been designed to be model-focused, meaning that each frontend phase processes the whole model before the model is passed on to the next phase. The result is the design seen in Figure 1, which shows the flow of the model through the different phases of the new frontend. The following sections will describe each phase in more detail.

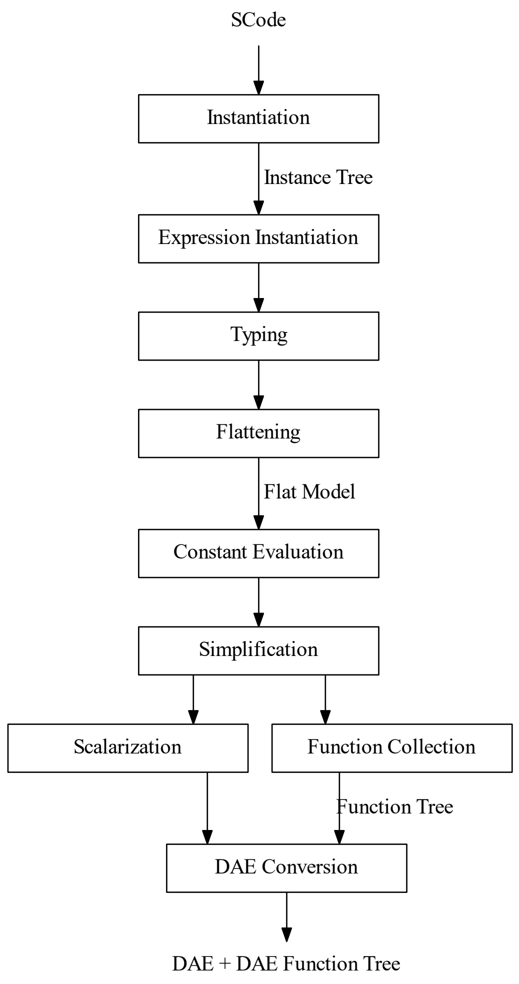


Figure 1. Frontend phases.

3.3.1 Instantiation

The instantiation phase takes all the libraries and models that have been loaded by the compiler in the form of an SCode structure as well as the name of the model that should be instantiated, and builds an instance tree for that model. The instance tree consists of the class instance corresponding to the model as the root node, with the component instances of the class as child nodes that themselves have component instances (as seen in Figure 2).

Because the SCode structure is not suitable for name lookup, as it only contains lists of elements, the instance tree is instead used for this purpose by the new frontend since each node contains a lookup tree. The first task of the instantiation is thus to partially instantiate the conceptual root class that contains all top-level classes, which mainly involves constructing a lookup tree. The first part of the model name can then be looked up in the root class, and the rest of the name is looked up recursively using the same process.

Once the SCode element of the model's class has been found, it will then be instantiated, which involves three stages: partial instantiation, expansion, and full instantiation. Partial instantiation will, as mentioned, construct a lookup tree, but only local classes and imported names are added in this stage. This is needed to be able to look up the names of base classes, since Modelica allows classes to inherit from local and imported classes but not from inherited classes.

The next stage, expansion, uses this partial lookup tree to resolve any base classes of the class. All the inherited names as well as the names of local components are then added to the lookup tree. The reason why local components are not added until this stage is because the order in which the local and inherited components are declared needs to be preserved, since this is important for e.g. functions where the order of the function parameters matter. The class elements are therefore stored in declaration order in arrays, with the lookup tree only referencing elements in those arrays, and the inherited components need to be known before all components can be added in the correct order.

The final stage is full instantiation in which the components of the class are instantiated, which involves looking up the type of each component and instantiating it. In this stage modifiers are also associated with the elements they modify, and redeclares are applied. The names of any base classes are also looked up again in this stage, to make sure that the same classes are found as in the earlier expansion stage since inheriting from an inherited class is illegal in Modelica (see Figure 3). This conveniently also allows the frontend to also check that no extends is referencing a component, since those are, as mentioned earlier, added to the lookup tree after resolving base class names.

Because modifiers are only applied in the full instantiation stage, it is possible for the new frontend to cache the

```
model A
  Real x;
  Real y;
end A;

model B
  A a;
  Real z;
end B;

model M
  B b;
end M;
```

$\Rightarrow$

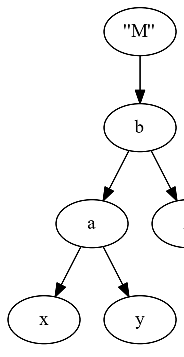


Figure 2. Example of a model and its instance tree.

```
model A
  model B
    ...
  end B;
end A;

model M
  extends A;
  // Illegal, B is inherited from A.
  extends B;
end M;
```

Figure 3. Example of illegal inheritance in a Modelica model.

work done during partial instantiation and expansion for each class. This means that e.g. the lookup tree for a class is only constructed once and then reused for all instances of that particular class, unlike the old frontend where a new lookup tree is constructed for each instance.

3.3.2 Expression Instantiation

Expressions in the compiler are things such as numbers, strings, unary and binary operator expressions, and named references to elements such as `a.b[2].c[4]`. They are used to represent things such as equations and algorithms, modifiers, and array dimensions.

The old frontend represents names used in expressions as a nested structure where each node contains the referenced element's name and type, the subscripts used, and for qualified names a reference to the next part of the name. The new frontend uses a similar representation, but instead of storing the element's name it stores a reference to the element's instance tree node (which itself contains the name).

This small difference in representation has a large impact on the design of the new frontend, because unlike the old frontend it always has direct access to the referenced elements. The old frontend is instead forced to look up names whenever it requires additional information about a name used in an expression, which due to how the old frontend is designed might require additional instantiation and other performance issues. Rewriting the old frontend to use a similar representation would have required a complete redesign, since it has no instance tree in the same way that the new frontend has.

For the new frontend this means that it needs to find the correct instance node during name lookup, which can be tricky to do during the instantiation since names can refer to elements that have not yet been instantiated. Expressions are not needed to build the instance tree though, so the instantiation is therefore separated into two phases: the instantiation phase described in the previous section and the expression instantiation phase described in this section.

The instantiation phase builds the instance tree and constructs all the nodes, and the expression instantiation phase instantiates all expressions in that instance tree. This involves looking up the names used in expressions and associating them with the correct nodes in the instance tree. In the case of function calls this also triggers instantiation of the called functions, which mostly involves instantiating the function as a normal class.

3.3.3 Typing

The typing phase traverses the instance tree and determines the type of all components and expressions. Similar to the instantiation this is again done in two stages: typing of components and typing of expressions.

The typing of components involves determining the type of each component in the instance tree. For components of basic types, such as `Real` or `Integer`, this is trivial by virtue of them being instances of said types. For composite types, such as instances of models, blocks or records, it means typing each child of the instance tree node and constructing a type from them.

The most complex part of the typing of components is typing dimensions for array components and classes. This is partly because dimensions can be expressions that need to be evaluated, and partly because they can be declared as `:` which means that the dimension size must be deduced from the component's binding equation.

This can require typing components that have not yet been typed, and must be done with care to avoid introducing unnecessary dependencies between dimensions. Having dimensions whose size depend on each other could result in a typing loop that would cause the compiler to hang or crash, but the new frontend detects such loops and gives an appropriate error message instead. In many cases such loops can be avoided by typing as little as possible when determining the size of a dimension, such as only typing the first dimension of `a` when typing a dimension defined as `size(a, 1)`.

The next stage of the typing phase is typing of expressions, which involves typing binding equations, equations, and algorithms. This also includes checking that expressions are type compatible, for example checking that binding equations are type compatible with the components they belong to. Having a separate stage for typing of expressions is not strictly necessary in the same way as during the instantiation, but it means that expressions can be typed with the assumption that all components are already typed and acyclic. The typing of expressions therefore becomes less complicated and more optimized than it would be if all the typing were to be done in a single combined stage.

Most of the typing of expressions is fairly straightforward, but the typing of binding equations becomes somewhat non-trivial in the new frontend due to the way array components are handled. Take for example this model:

```
model A
  Real x;
end A;

model M
  A a[3](x = {1, 2, 3});
end M;
```

The old frontend would instantiate and type each element of the array component `a` and the modifier on `a` would be split, so the component would thus be instantiated and typed as `a[1].x = 1`, `a[2].x = 2`, and `a[3].x = 3` (where `[1]`, `[2]`, and `[3]` are subscripts).

The new frontend instead treats this as one component, `a[3].x = {1, 2, 3}` (where `[3]` is a dimension), which is achieved by keeping track of where a modifier comes from and adding the appropriate dimensions to the component's type when type checking the binding equations. In this particular case it would thus add the dimension `[3]` to the type of `x` when checking that the binding equation is type compatible, in other words type checking one `Real[3] == Real[3]` relation whereas the old frontend would type check three separate `Real == Real` relations. The superior efficiency of this approach in the case arrays with thousands of elements need to be instantiated is pretty obvious.

3.3.4 Flattening

The flattening phase of the new frontend traverses the instance tree and flattens the tree into a flat model that consists of a list of variables, a list of equations and a list of algorithms:

```
model A
  Real x;
  Real y;
equation
  y = der(x);
end A;                        model M
                                Real b.a.x;
model B                         Real b.a.y;
  A a;                  ⇒     equation
equation                        b.a.y = der(b.a.x);
  a.x = time;                   b.a.x = time;
end B;                        end M;

model M
  B b;
end M;
```

The flattening involves prefixing component names and element name references in expressions with the names of their parents in the instance tree, to make sure all variables in the flat model have unique names. It also collects all the connect-equations in the model and inserts the required equations generated from the connections into the flat model.

Another task done by the flattening phase is unrolling for-equations into scalar equations:

```
for i in 1:3 loop                x[1] = 1;
  x[i] = i;               ⇒      x[2] = 2;
end for;                         x[3] = 3;
```

This might be considered more appropriately done by the later scalarization phase, or preferably not done at all, even though the connection handling requires for-equations containing connect-equations to be unrolled. The current backend additionally requires all for-equations to be unrolled, so at the moment the flattening unrolls all for-equations by default, regardless of whether they contain connect-equations or not. However, it is possible to disable the loop unrolling (as well as other scalarization features discussed later on in the paper) with the `-d=-nfScalarize` debug flag, which allows to experiment with extensions of the backend, code generation, and runtime phases that can handle arrays directly.

3.3.5 Constant Evaluation

Some parts of the frontend evaluate expressions when needed, for example when typing dimensions consisting of arbitrary expressions where the actual size needs to be known in a model context (unlike in a function context). Constants that are not used in such places should still be evaluated though, which is done in the constant evaluation phase. This phase traverses the flat model and replaces references to constants with the values bound to those constants:

```
model M
  constant Real x = 1.0;            model M
  Real y;                             Real y;
equation                      ⇒     equation
  y = x;                              y = 1.0;
end M;                              end M;
```

Models can also contain so called structural parameters, which are parameters used in places where they affect the structure of the model. One example is array dimensions which must as mentioned be known in a model context, but are allowed to be defined by parameters. Once such a parameter has been evaluated it should no longer be considered changeable, since changing its value after the model has been compiled could result in parts of the model using the old value and other parts the new value. The earlier parts of the new frontend therefore mark such parameters as structural, and the constant evaluation phase makes sure all occurrences in the model are replaced with the parameter's value.

3.3.6 Simplification

The simplification phase traverses the flat model and simplifies expressions, equations and algorithms. This includes doing trivial simplifications such as evaluating unary and binary operations involving numerical literals, e.g. `1 + 1` ⇒ `2`, but also structural changes such as removing for-loops with zero-sized iteration ranges.

3.3.7 Scalarization

The scalarization phase expands array variables and equations into separate scalar variables and equations:

```
model M
  Real x[3];
equation
  x = {1, 2, 3};
end M;
```

⇒

```
model M
  Real x_1;
  Real x_2;
  Real x_3;
equation
  x_1 = 1;
  x_2 = 2;
  x_3 = 3;
end M;
```

This is not necessary for the operation of the frontend itself, but is done because the old frontend does it and the backend expects it to be done. A long term goal is to improve the handling of arrays in the backend though, partially or completely removing the need for this phase (see Section 4).

Since the scalarization is a separate phase in the new frontend it can also easily be disabled, unlike in the old frontend where the scalarization is an integral part of the flattening process that is hard to isolate. This is currently possible by means of the already mentioned `-d=-nfScalarize` debug flag.

3.3.8 Function Collection

The flat model contains only the variables, equations and algorithms of the model. The functions used in the model are stored in the instance tree nodes corresponding to the functions' classes, but the backend expects to get a binary search tree containing the functions that are used in the model.

The new frontend therefore has a phase that goes through the model and collects all functions that are used in the model into a function tree. Besides explicitly called functions, this also includes, e.g., record constructors for all record instances, which might be needed by the backend even if they are not explicitly called in the model.

3.3.9 DAE Conversion

The old frontend produces a DAE structure that's used as an immediate representation of the flat model, and the OpenModelica backend expects the model to be given in this format.

The flattening phase of the new frontend uses its own representation of a flat model though, since using the old DAE structure would cause many of the advantages of the new frontend to be lost (such as name references in expressions having direct access to the instance tree nodes).

The new frontend therefore contains a final phase that converts the flat model and the function tree to the DAE structure expected by the backend. This serves as an interface between the new frontend and the backend, and is relatively straightforward since the DAE structure is mostly a subset of the data structures used by the new frontend.

4 Compilation of Vectorized Models for New Digital Applications

Many emerging applications require the individual control of vast numbers of similar devices that share a common system infrastructure. Such applications include distributed renewable power generation and charging of electric vehicles that share the same power grid. Other applications arise from autonomously driving cars that share the same roads. The Internet of Things opens the possibility to connect such devices to digital twins and to implement supervisory control applications in the cloud.

Unfortunately today's Modelica tools typically suffer from bad performance for the translation of resulting large models. This is caused by the Modelica DAE representation defined for a flat model. Even though the Modelica syntax supports arrays of model objects to express repeated structures, the expansion of arrays during flattening results in large numbers of scalar variables and equations that slow down the translation. This is particularly bad if the computational effort of the used algorithms grows more than linearly with increasing model size. Moreover, the resulting executable model code becomes unnecessarily large.

The new frontend offers the feature to convert arrays of component models to array equations and to keep arrays during flattening. Additionally, the current backend has been extended prototypically, by exploiting previous work by (Schuchart et al., 2015) to treat for-equations during model translation and by (Franke et al., 2015) to treat unexpanded arrays and array slices in the generated code.

Consider the following example. It instantiates a large number of solar plants as array of component models and connects them to a collector grid.

```
package Vectorized
  import SI = Modelica.SIunits;

  connector Terminal
    SI.Voltage v;
    flow SI.Current i;
  end Terminal;

  model SolarPlant
    input Boolean on "Plant status";
    input SI.Power P_solar "Solar power";
    parameter Real eta = 0.9 "Efficiency";
    Terminal term;
  equation
    term.v * term.i =
      if on then eta * P_solar else 0;
  end SolarPlant;

  model Collector
    parameter Integer n;
    parameter SI.Voltage V = 1000;
    output SI.Power P_grid;
    Terminal terms[n];
  equation
    for i in 1:n loop
      terms[i].v = V;
    end for;
    0 = P_grid + terms.v * terms.i;
  end Collector;
  model SolarSystem
    parameter Integer n = 1000;
    SolarPlant plant[n](
    each on = true,
    P_solar = 100:100:n*100);
    Collector grid(n = n);
  equation
    connect(plant.term, grid.terms);
  end SolarSystem;

end Vectorized;
```

With a number of n=1000 solar plants, the flat model would have 6001 variables and 6001 equations. The number of variables reduces to 7 when preserving arrays. Then the whole dependency analysis, equation sorting and code generation only treat 7 equations. This is possible as each array variable is defined with one array equation, resulting in a balanced array model.

The current implementation in OpenModelica still considers the dimension parameter *n* as structural and fixes its value during model instantiation. This is not needed though. The actual value of *n* could be left undefined until model execution.

The flat array model reads:

```
class SolarSystem
  parameter Integer n = 1000;
  Real[1000] plant.term.i;
  Real[1000] plant.term.v;
  parameter Real[1000] plant.eta = 0.9;
  Real[1000] plant.P_solar =
    (100:100:100000);
  Boolean[1000] plant.on = true;
  parameter Integer grid.n = 1000;
  parameter Real grid.V = 1000.0;
  Real grid.P_grid;
  Real[1000] grid.terms.i;
  Real[1000] grid.terms.v;
equation
  plant.term.v = grid.terms.v;
  plant.term.i + grid.terms.i = 0.0;
  for $i in 1:1000 loop
    plant[$i].term.v * plant[$i].term.i =
      if plant[$i].on then
        plant[$i].eta * plant[$i].P_solar
      else
        0.0;
  end for;
  for i in 1:1000 loop
    grid.terms[i].v = grid.V;
  end for;
  0.0 = grid.P_grid +
    grid.terms.v * grid.terms.i;
end SolarSystem;
```

The array of connectors between plant and grid results in array equations. The flow equation, as well as some variable bindings, relate arrays to scalars. This simplifies the further treatment up to code generation, implicitly assuming an "each" qualifier. Note that this only happens after the check of types and dimensions. It is crucial to avoid unnecessary expansions of large literal arrays.

Many Modelica expressions cannot be vectorized easily. For instance the condition of an if-expression must be a scalar boolean in an array equation as well. This is why the vectorization of component models converts non-trivial equations to for-equations. See the equation with P_solar as an example.

The early prototype implementation of vectorized models presented here already proved useful in first production uses (see next section for benchmarks). The model given in this section can be compiled and simulated without array expansion via flags `-d=newInst,-nfScalarize --simCodeTarget=Cpp`. Future work needs to focus on enhanced preservation of arrays during symbolic transformations in the backend. The frontend might expand arrays selectively, e.g. expand two or three dimensional arrays in electrical multi-phase models, while preserving large arrays of component models along with dimension parameters.

5 Status and Benchmarks

The new frontend is still work in progress at the time of this writing (January 2019). It is currently able to process about 75% of the 7884 models with an `experiment(StopTime)` annotation in the set of 55 tested open-source Modelica libraries that are included in the extended testsuite of the OpenModelica continuous integration system. The development effort so far has been focused towards achieving full coverage of the Modelica Standard Library (MSL) 3.2.3, for which the fraction of successfully simulating models is currently 92%, including non-trivial models such as the 6 d.o.f. robot model of `Modelica.Mechanics.Multibody` and models using the IF97 water model of `Modelica.Media`.

The updated status of the coverage is available online (New FrontEnd - Modelica Library Coverage). The development of the new frontend can be followed on (New FrontEnd - Ticket 4138); in particular, the progress of the coverage of the development version of MSL 3.2.3 is shown in Fig. 4. The new frontend is currently able to process all models except one, though there are still some issues that are revealed later in the code generation process, either because of incorrectly flattened models, or because the model is flattened in a different way than the old frontend, which the back-end cannot handle correctly. Note that the verification indicator is not reliable, due to many false negatives and to the lack of reference results for all the models introduced in version 3.2.3.

During the last six months, the number of successfully simulating MSL models has steadily increased at a rate of about 8%/month, so it is expected that full coverage for the MSL will be achieved by the end of Q2 2019 at the latest; the coverage of the 55 open-source library testsuit should approach 100% before the end of 2019.

All the benchmarks have been realized on a portable computer: HP ZBook Studio G3 I7 QuadCore 6820HQ @ 2.7Ghz with 16Gb of RAM.

To validate performance and scalability we have benchmarked the new OpenModelica frontend against the state-of-the-art commercial tool Dymola 2019 (2018-04-11) (Dassault Systèmes) on some large models from the ScalableTestSuite library (Casella, 2015).

An example of the Dymola and OpenModelica scripts

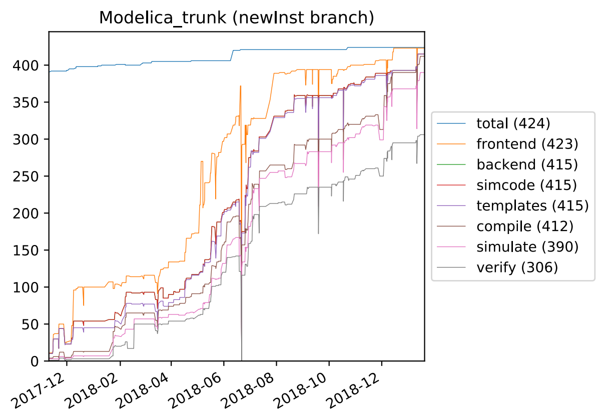


Figure 4. Modelica Standard Library version 3.2.3 coverage

used are given below. Note that this include parsing of the Modelica Standard Library, the ScalableTestSuite and running `checkModel` on the given model where ModelPath is the full path to the model in the library.

Dymola script:

```
openModel("ScalableTestSuite/package.mo");
checkModel("ModelPath");
exit();
```

OpenModelica script:

```
loadFile("ScalableTestSuite/package.mo");
getErrorString();
checkModel(ModelPath);
getErrorString();
```

The benchmarking was performed from command line using running adaptations of the scripts above. The results for selected ScalableTestSuite (STS) models and the Vectorized.SolarSystem from section 4 are given below in Table 1.

One can see that the new OpenModelica frontend performs very well in comparison to Dymola, in some cases faster, in some cases slower. The comparision between the current frontend (CF) and the new frontend (NF) is also included where possible. From these benchmarks one can also see that investigation is needed to find out why parameter arrays are scaling poorly in the new frontend (models 6, 7, 8). For models 10 and 11 the number in the parentheses is for the new frontend not expanding arrays at all during the flattening. The performance improvement in this case is extreme.

In Table 2 we compare the current frontend (CF) with the new frontend (NF) when instantiating and flattening models from `Modelica.Mechanics.MultiBody` and evaluating their graphical annotation. The OpenModelica compiler API function that is called to evaluate the graphical annoations is `getComponentAnnotations()`. The new frontend performs 20 to 200 times better than the current OpenModelica frontend, allowing to obtain a nearly immediate response time of the OMEdit GUI, which relies on this API.

6 Conclusions and Future Work

In this paper, the new high-performance frontend of the OpenModelica compiler is presented. The frontend has been completely redesigned, with the main objective of achieving dramatically improved performance on large models, as well as of resolving many corner-cases that the old frontend could not handle without the need of excessive ad-hoc work.

The architecture of the new design is presented in detail, particularly concerning the new approach that avoids the full expansion and scalarization of components one at a time, thus allowing significant optimizations when large numbers of instances of the same class, and/or large arrays, are present in the model. Many of these optimization would also require a substantial redesign of the compiler backend, code generation, and runtime system. For the time being the new non-scalarization approach has been experimented with a prototype implementation in the backend, which works in some specific cases, for which very promising results are reported.

Future developments involve first and foremost the finalization of the new frontend, with the aim of achieving 100% coverage of most open-source Modelica libraries, particularly the MSL. This goal is planned to be achieved during the first half of 2019. In the long term, the plan is to use the new frontend to achieve full support of non-expanded arrays of equations and models in the entire compiler toolchain, including also the backend, code generation, and runtime system.

Another research direction is to improve the compilation speed by using the LLVM framework to perform function evaluation in the new frontend.

7 Acknowledgements

This work has been supported by Vinnova in the ITEA OPENCPS and EMPHYSIS projects and in the Vinnova RTISIM and EMISYS projects. This work was supported in parts by the German Federal Ministry of Education and Research (BMBF) in the PARADOM project. Support from the Swedish Government has also been received from the ELLIIT project. The OpenModelica development is supported by the Open Source Modelica Consortium. Many students, researchers, engineers have contributed to the OpenModelica system. There is not room here to mention all these people, but we gratefully acknowledge their contributions.

References

Alfred V. Aho, Ravi Sethi, and Jeff D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley Publishing Company, 1986.

Torsten Blochwitz et al. The Functional Mockup Interface for tool independent exchange of simulation models. In Christoph Clauß, editor, *Proceedings of the 8th International Modelica Conference*. Linköping University Electronic Press, March 2011. doi:10.3384/ecp11063105.

Willi Braun, Francesco Casella, and Bernhard Bachmann. Solving large-scale Modelica models: new approaches and experimental results us-

No	Model	Equations	Dym (s)	OMC NF/CF (s)
1	Electrical.DSystemAC.SE.DistributionSystemLinear_N_40_M_40	99776	15.53	06.32 / 91.33
2	Electrical.DSystemAC.SE.DistributionSystemLinear_N_80_M_80	397936	40.50	17.76 / 435.32
3	Electrical.DSystemAC.SE.DistributionSystemLinear_N_112_M_112	779312	74.21	32.31 / 1076.54
4	Electrical.DSystemDC.SE.DistributionSystemModelicaActiveLoads_N_80_M_80	129929	18.04	08.33 / 159.28
5	Electrical.TransmissionLine.SE.TransmissionLineModelica_N_1280	26915	09.84	04.45 / 47.77
6	Elementary.ParameterArrays.SE.Table_N_100_M_100	0	06.59	05.09 / 06.21
7	Elementary.ParameterArrays.SE.Table_N_400_M_400	0	10.25	12.19 / 18.03
8	Elementary.ParameterArrays.SE.Table_N_1600_M_100	0	09.77	19.04 / 28.17
9	Power.ConceptualPowerSystem.SE.PowerSystemStepLoad_N_64_M_16	11907	17.29	03.99 / 28.57
10	Vectorized.SolarSystem(n=10000) from section 4	60001	146.30	34.12 / 314.8 (**02.95**)
11	Vectorized.SolarSystem(n=100000) from section 4	600001	14458.68	2450.57 / 19760.42 (**02.95**)

Table 1. Flattening performance comparison Dymola vs. OpenModelica (NF vs CF included). Bold numbers in parentheses are with Scalarization disabled `-d=-nfScalarize`. Shortened names: SE=ScaledExperiments, DSystem=DistributionSystem.

Model	CF (s)	NF (s)	Factor
World	9.53	0.28	33.9
Joints.FreeMotionScalarInit	28.90	0.14	199.4
Joints.Planar	3.56	0.13	25.6
Joints.UniversalSpherical	6.99	0.22	30.5
Joints.SphericalSpherical	4.64	0.11	39.5
Joints.Universal	2.31	0.12	18.4

Table 2. Flattening performance comparison of the current (old) vs the new frontend in OpenModelica (OMEdit GUI impact).

ing OpenModelica. In *Proc. 12th International Modelica Conference*, pages 557–563, Prague, Czech Republic, May 15–17 2017. doi:10.3384/ecp17132557.

Francesco Casella. Simulation of large-scale models in Modelica: State of the art and future perspectives. In Peter Fritzson and Hilding Elmqvist, editors, *Proceedings 11th International Modelica Conference*, pages 459–468, Versailles, France, Sep 21–23 2015. The Modelica Association. ISBN 978-91-7685-955-1. doi:10.3384/ecp15118459.

Francesco Casella, Alberto Leva, and Andrea Bartolini. Simulation of large grids in OpenModelica: reflections and perspectives. In *Proc. 12th International Modelica Conference*, pages 227–233, Prague, Czech Republic, 2017. doi:10.3384/ecp17132227.

Dassault Systèmes. Dymola version 2019, 2018. URL `http://dymola.com`.

Rüdiger Franke, Marcus Walther, Niklas Worschech, Willi Braun, and Bernhard Bachmann. Model-based control with FMI and a C++ runtime for Modelica. In *Proceedings of the 11th International Modelica Conference*. Modelica Association, Paris, France, 2015.

Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley-IEEE Press, 2 edition, April 2015. ISBN 978-1-118-85912-4.

Peter Fritzson, Adrian Pop, Adeel Asghar, Bernhard Bachmann, Willi Braun, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Alejandro Danós, Rüdiger Franke, Mahder Gebremedhin, Bernt Lie, Alachew Mengist, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, Waurich Volker, and Per Östlund. The OpenModelica Integrated Modeling, Simulation and Optimization Environment. In Michael Tiller and Luigi Vanfretti, editors, *Proceedings of the 1st American Modelica Conference*. Linköping University Electronic Press, October 2018. URL `http://www.ep.liu.se/`.

Antonio Froio, Francesco Casella, Fabio Cismondi, Alessandro Del Nevo, Laura Savoldi, and Roberto Zanino. Dynamic thermal-hydraulic modelling of the EU DEMO WCLL breeding blanket cooling loops. *Fusion Engineering and Design*, 124:887–891, 2017. doi:10.1016/j.fusengdes.2017.01.062.

Görel Hedin and Eva Magnusson. JastAdd: An aspect-oriented compiler construction system. *Sci. Comput. Program.*, 47(1):37–58, April 2003. ISSN 0167-6423. doi:10.1016/S0167-6423(02)00109-0. URL `http://dx.doi.org/10.1016/S0167-6423(02)00109-0`.

Paul Hudak. *The Haskell School of Expression: Learning Functional Programming Through Multimedia*. Cambridge University Press, New York, NY, USA, 2000. ISBN 0-521-64408-9.

Robin Milner, Mads Tofte, and David Macqueen. *The Definition of Standard ML*. MIT Press, Cambridge, MA, USA, 1997. ISBN 0262631814.

Modelica Association. Modelica: A unified object-oriented language for physical systems modeling, language specification version 3.4, 2017. URL `http://www.modelica.org/`.

New FrontEnd - Modelica Library Coverage. New FrontEnd - Modelica Library Coverage, 2018. URL `https://libraries.openmodelica.org/branches/overview-newinst.html`.

New FrontEnd - Ticket 4138. New FrontEnd - Ticket 4138, 2018. URL `https://trac.openmodelica.org/OpenModelica/ticket/4138`.

OSMC. Open Source Modelica Consortium, 2007. URL `https://openmodelica.org/home/consortium`.

Adrian Pop and Peter Fritzson. MetaModelica: A unified equation-based semantical and mathematical modeling language. In *7th Joint Modular Languages Conference, JMLC 2006 Oxford, UK, September 13-15, 2006 Proceedings*, pages 211–229. Springer Berlin Heidelberg, 2006. doi:10.1007/11860990_14.

Johan Åkesson, Torbjörn Ekman, and Görel Hedin. Implementation of a Modelica compiler using JastAdd attribute grammars. *Sci. Comput. Program.*, 75(1-2):21–38, January 2010. ISSN 0167-6423. doi:10.1016/j.scico.2009.07.003. URL `http://dx.doi.org/10.1016/j.scico.2009.07.003`.

Joseph Schuchart, Volker Waurich, Martin Flehmig, Marcus Walther, Wolfgang E. Nagel, and Ines Gubsch. Exploiting repeated structures and vectorization in Modelica. In *Proceedings of 11th International Modelica Conference*. Modelica Association, Paris, France, 2015.

Martin Sjölund, Peter Fritzson, and Adrian Pop. Bootstrapping a Compiler for an Equation-Based Object-Oriented Language. *Modeling, Identification and Control*, 35(1):1–19, 2014. doi:10.4173/mic.2014.1.1.

OMJulia: An OpenModelica API for Julia-Modelica Interaction

Bernt Lie[1], Arunkumar Palanisamy[2], Alachew Mengist[2], Lena Buffoni[2], Martin Sjölund[2], Adeel Asghar[2], Adrian Pop[2], Peter Fritzson[2]

[1]University of South-Eastern Norway, Porsgrunn, Norway, Bernt.Lie@usn.no;
[2]Linköping University, Linköping, Sweden, Peter.Fritzson@liu.se

Abstract

Modelica is an object oriented, acausal equation-based language for describing complex, hybrid dynamic models. About ten Modelica implementations exist, of which most are commercial and two are open source; the implementations have varying levels of tool functionality. Many Modelica implementations have limited support for model analysis. It is therefore of interest to integrate Modelica tools with a powerful scripting and programming language, such as Julia. Julia is a modern and free language for scientific computing. Such integration would facilitate the needed analysis possibilities and can speed up the development of effient simulation models. A number of design choices for interaction between Julia and Modelica tools are discussed. Next, Julia package OMJulia is introduced with an API for interaction between OpenModelica and Julia. Some discussion of the reasoning behind the OMJulia design is given. The API is based on a new class *ModelicaSystem* within package OMJulia, with systematic methods which operate on instantiated models. OMJulia supports handling of FMU and Modelica models, setting and getting model values, as well as some model operations. Results are available in Julia for further analysis. OMJulia is a further development of a previous OMPython package; a key advantage of Julia over Python is that Julia has better support for control engineering packages. OMJulia represents a first effort to interface a relatively complete Modelica tool to Julia, giving access to an open source set-up for modeling and analysis, including control synthesis, easily installable from a unified package manager. Some possibilities of OMJulia are illustrated by application to a few simple, yet industrially relevant problems within control design. *Keywords*: Modelica, FMI, FMU, OpenModelica, Julia, Julia API, OMJulia

1 Introduction

Julia is a modern, rich script language, (Bezanson et al., 2017), with excellent support for efficient and fast differential equation solvers (Rackauckas and Nie, 2017), including DAEs (Sund et al., 2018), as well as a number of other packages for plotting, control engineering, optimization, statistics, machine learning, etc., (JuliaLang, 2018).

Modelica is a modern, equation based, acausal language for encoding models of dynamic systems in the form of differential algebraic equations (DAEs), see, e.g., (Modelica Association, 2016), (Modelica Association, 2017), (Fritzson, 2015) on Modelica, and, e.g., (Brenan et al., 1989) on DAEs. The Functional Mock-up Interface (FMI) is a common standard format to support both model exchange and co-simulation of dynamic models in the form of Functional Mock-up Units (FMU) between many modelling and simulation environments, (FMI Consortium, 2018).

OpenModelica[1] (Fritzson et al., 2018) is a mature, freely available tool set that includes *OpenModelica Connection Editor* (flow sheeting, textual editor with debugging facilities, and simulation environment), the *OMShell* (command line/script based execution), and a number of extensions. OpenModelica Shell supports commands for simulation of Modelica models, for use of the Modelica extension Optimica, for carrying out analytic linearization via the Modelica package *Modelica_LinearSystem2*, and for converting Modelica models into Functional Mock-Up Units (FMUs) as well as for converting FMUs back to Modelica models. However, the OMShell is relatively limited wrt. other, advanced analysis possibilities such as availability of random number generator, control tools, etc.

Based on *OMPython* (Ganeson, 2012; Ganeson et al., 2012), an API was developed for simple operation on Modelica models from within Python (Lie et al., 2016). Both Modelica (Baur et al., 2009) and Python[2] have limited support for control tools, and it is of interest to explore connecting OpenModelica to other scripting tools with richer eco-systems for control engineering — two possibilities are MATLAB and Julia. To ease the maintenance of interfacing Modelica with 3 different script languages, it is necessary to compromise on the specific style of each language. This paper discusses the API adapted to Julia, and illustrates how OMJulia can be used for analysis of Modelica models, exemplified by a simple water tank model, and then for more advanced analysis of a nonlinear reactor model[3]. The paper is organized as follows. In Section 2, an overview of the API is given. In Section 3, use of the API is applied to analysis of a simple, process oriented model. In Section 4, a somewhat more complex chemical engineering type process is used to illustrate possibilities

[1]`www.openmodelica.org`

[2]https://sourceforge.net/p/python-control/wiki/Home/

[3]The nonlinear reactor case will be added in the final paper.

with combining OpenModelica with Julia. In Section 5, some discussion of the API is provided with conclusions.

2 Overview of Julia API

2.1 Goal

Julia is a modern, rich script language, while Modelica, offers mature, equation based encoding of physically based models, with system (input-output), and library support. It is of interest to consider the use of Modelica with Julia for a wide range of engineering disciplines. The computer science threshold of using Modelica with Julia should be low. The OMJulia extension should be installed via the standard Julia packet manager (Git-based), and support the same platforms as Julia does. Results should be returned as standard Julia structures.

OMJulia can be installed as described at `https://github.com/OpenModelica/OMJulia.jl`.

2.2 Design Choices

Essentially, four paths to Modelica–Julia interaction are realistic[4].

1. Sending Modelica script commands as text strings from Julia to the Modelica tool via the ZMQ communication protocol[5] (Hintjens, 2013), and retrieving results. This is similar to the original idea of OMPython[6]. Advantage: simple solution. Disadvantage: requires detailed knowledge of Modelica tool script commands; possibly relatively slow if the interaction time is a large fraction of the computation time.

2. Julia API with commands native to Julia, which are translated to Modelica script commands "behind-the-scene", interacts with Modelica via ZMQ, and with results returned to Julia in Julia objects. Advantage: simple to use within Julia. Disadvantage: limited to existing possibilities in Modelica tool; possibly relatively slow.

3. Translate Modelica code to Julia code. Currently, OpenModelica code is translated to C code. It is possible to alternatively translate the code to Julia code. Advantage: utilize specialized syntax (Modelica) for describing models, and with full integration with Julia, fast. Disadvantage: the user must handle two languages.

4. Extend Julia with Modelica-like structures, such as the Modia initiative (Elmqvist et al., 2017). Advantage: the user operates in one language, fast. Disadvantage: limitations in Julia syntax and slightly different language semantics may make the extensions more complex for the user than Modelica is.

[4]The same paths are possible with other script languages such as Python and MATLAB

[5]http://zeromq.org/

[6]Originally, OMPython used CORBA technology instead of ZMQ

Ideal integration for speed and use of Julia tools would be achieved by either design choices 3 or 4. `Sims.jl` represents an early exploration of choice 4, while `Modia.jl` represents a newer, more extensive work within choice 4.[7] Here, we describe the OMJulia API, which belongs to design choice 2. A longer term plan is to improve on the previous OMPython API (Lie et al., 2016), and offer a suite for Python, Julia, and MATLAB.

Based on experience with the OMPython API, the syntax of the OMJulia API is updated/improved for easier use. To be future proof, the tool developer should "own" the API. Ease of maintenance of such a suite is essential, which implies that the syntax should be similar across script languages. Thus, some compromises must be made wrt. syntax. As an example, the key paradigm in Python is objects, and applying method `simulate` to object `mod` would have the syntax `mod.simulate()`. The key paradigms in Julia are types and multiple dispatch ("function overloading"), and the natural syntax in Julia would be `simulate(mod)` where the type of `mod` decides which method/function implementation is used ("dispatching"). Still, Julia allows for the same syntax as Python, and the Pythonian syntax is therefore chosen — for ease of maintenance. Ease of maintenance also dictates that OMJulia should depend on as few packages as possible, and take advantage of existing packages in Julia for plotting, etc.

2.3 Description of the API

The API is described in the subsections below.

2.3.1 Julia Class and Constructor

The first step to using the OMJulia API is to introduce it in the Julia session using the `using` command:[8]

```
julia> using OMJulia
```

Next, an empty Julia model object is constructed which communicates with OpenModelica:[9]

```
julia> mod = OMJulia.OMCSession()
```

We are now ready to fill the model object with content. The OMJulia method which is used to populate the model object with a Modelica model is the model constructor `ModelicaSystem()`. This constructor requires two arguments, with an optional third argument:

1. The first argument is a string containing the name of the Modelica file which holds the model, if necessary with full directory path.

2. The second argument is a string containing the name of the main Modelica model within the file.

[7]See `www.julialang.org` under `Explore packages`.

[8]The Julia prompt `julia>` is not typed, and does not appear in script files, nor in IJulia/Jupyter notebooks.

[9]Any valid Julia identifier can be used as the model object name.

3. If the main Modelica model uses some libraries (e.g., the Modelica Standard Library), these are listed as strings in a Julia vector (= 1D array) in a third argument. If a single library is used, the vector of a single string can be replaced by the string.

Example 1. Use of Model Constructor

Suppose that we have establised a Julia object `mod` which communicates with OpenModelica, see above. Suppose next that we have a Modelica model with name CSTR, wrapped in a Modelica package *Reactors* — stored in file `Reactors.mo`:

```
package Reactors
  // ...
  model CSTR
    /// ...
  end CSTR;
  //
end Reactors;
```

Assuming that no external Modelica code is used, the following Julia code populates the Julia object `mod` with the Modelica model:

```
julia>  mod.ModelicaSystem("Reactors.mo", "
    Reactors.CSTR")
```

▲

2.3.2 Methods, Arguments, and Return Values

In the Julia language, it is in general recommended *not* to use class functions ("methods") in the way we have done in OMJulia. Instead of using get and set methods (as in Python), one could operate directly on the object attributes[10]. And instead of using methods that transform the object, e.g., simulate, linearize, etc., one could define general functions combined with type dispatching. However, because OMJulia is part of a family of script language interfaces for OpenModelica, some compromise has been made in order to simplify maintenance. To this end, in OMJulia, "methods" in the sense of object oriented languages a la Python are appended to the object after a dot.[11]

Methods in OMJulia have zero or one argument. In the case of one argument, this is either a Julia string or a vector (= 1D array) of strings.[12] The following Julia syntax is useful in this context:

1. String *concatenation* is achieved by symbol `*`, thus strings `"K"`, `"="`, and `"5"` can be concatenated by `"K"*"="*"5"` to become `"K=5"`.

2. String *substitution* (referred to as string *interpolation* in the Julia community) is achieved by the reserved symbol `$`, e.g., `"T=\$(25+273)"` is interpreted as `"T=298"`, while `T0=298` followed by `"T=\$T0"` or `"T=$(T0)"` gives the same result.[13]

Some methods return a single string `s` holding a numerical value, or a vector `v` holding strings each with a numerical value. Such a string `s` can be trivially converted to a floating point number by `parse(Float64,s)`; such a vector `v` can be converted to a vector of floating point numbers by `[parse(Float64,s) for s in v]`.

In the subsequent overview of methods, object name `mod` is used for illustration — in real use, any valid Julia identifier can be used as object name. Methods may or may not return results — if the methods do not return results, the results are stored within the object.

2.3.3 Utility Routines, Converting Modelica ↔ FMU

Two utility methods convert files between Modelica files with file extension `.mo` and Functional Mock-up Unit (FMU) files with file extension `.fmu`.

1. `mod.convertMo2Fmu()` — method for converting the *Modelica model* of the object into an FMU file.
   - Required arguments: *none*, operates on the Modelica file associated with the object.
   - Optional input arguments:
     - `version`: string with FMU version, `"1.0"` or `"2.0"`; the default is `"1.0"`.
     - `fmuType`: string with FMU type, `"me"` (model exchange) or `"cs"` (co-simulation); the default is `"me"`.
     - `fileNamePrefix`: string; the default is `"className"`.
   - Return argument:
     - `generatedFileName`: string, returns the full path + filename of the generated FMU (`.fmu`).

2. `mod.convertFmu2Mo(s)` — method for converting an FMU file into a Modelica file.
   - Required input arguments: string `s`, where `s` holds the name of the FMU file, including extension `.fmu`.

[10]In Julia, operating directly on the object attributes is safe because Julia is a strongly typed language, contrary to, e.g., Python. Safe, assuming that strong type definition has been used.

[11]In Julia, the word *method* as a different meaning than in general object oriented languages. Here, the word "method" is used as in object oriented languages such as Python.

[12]In the OMPython initiative, (Lie et al., 2016), Python's keyword assignment syntax was used. Keyword assignment is, however, troublesome, since possible Modelica identifiers such as `mod.K` and `der(x)` are invalid as identifiers/keywords in Python, Julia, etc.

[13]With `$` being a reserved symbol in Julia, it is necessary to use the escape character `\`, i.e., `\$` to achieve the effect of character `$` in strings, e.g., to specify LaTeX typesetting. Alternatively, by using Julia package `LaTeXString`, syntax `L"..."` replaces `$` with `\$` in the string without user intervention.

- Optional input arguments: a number of optional input arguments, e.g., the possibility to change working directory for the imported FMU files.
- Return argument:
  - `generatedFileName`: string, returns the full path + filename of the generated Modelica file (`.mo`).

2.3.4 Get and Set Information

Several methods are dedicated to getting and setting information about objects. With two exceptions — `getQuantities()` and `getSolutions()` — the get methods have identical use of arguments and results, while all the set methods have identical use of input arguments, with results stored in the object.

Get Quantity Information. Show Quantity Information Method `mod.getQuantities()` has no input arguments, and returns a vector[14] of dictionaries, one dictionary for each quantity. Each dictionary has the following keys (strings) — with values being strings, too.

- `"name"` — the name of the quantity, e.g., `"T"`, `"der(T)"`, `"n[1]"`, `"mod1.T"`, etc.,
- `"aliasvariable"` — typically `nothing`,
- `"variability"` — typically `"continuous"`, `"parameter"`, etc.,
- `"changeable"` — value `"true"` or `"false"`,
- `"causality"` — value `"internal"` or `"external"` (for inputs),
- `"value"` — string of number `"50"`, text string, or `"None"`,
- `"description"` — string copied from Modelica: description of the quantity, e.g. `"Mass in tank, kg"`, or `nothing`.
- `"alias"` — typically `"noAlias"`.

Modelica *constants* are not included in the returned vector of dictionaries.[15]

A Julia specific utility function `mod.showQuantities()` is included with the same syntax as `mod.getQuantities()`, taking advantage of Julia `DataFrames` to present the quantities in a table.[16]

[14]In Julia, a *vector* is a 1D array.

[15]In Modelica, **`constant`** is used for values which require recompilation when changed. **`parameter`** values, on the other hand, can be changed without recompilation.

[16]In Python, `mod.showQuantities()` is redundant because the return object directly produces a table with Python `pandas`.

Get Solutions We consider method `getSolutions()` — which assumes that the `simulate()` method has been applied (see below). Three calling possibilities are accepted.

- `mod.getSolutions()`, i.e., without input arguments, returns a *vector* of strings of *names* of quantities for which there is a solution.[17]
- `mod.getSolutions(s)`, where `s` is a *string* of a name, returns a single time series (= *vector* of floating point numbers) for the corresponding name.
- `mod.getSolutions(v)`, where `v` is a *vector* of strings of names, returns a *vector* of time series (= *vectors* of floating point numbers) for the corresponding names.

It follows that a vector of all time series can be returned by the construct `mod.getSolutions(mod.getSolutions())`.

Standard Get Methods We consider methods `getXXX()`, where `XXX` is either of `{ Continuous, Parameters, Inputs, Outputs, SimulationOptions, LinearizationOptions }`. Thus, methods `mod.getContinuous()`, `mod.getParameters()`, etc. Three calling possibilities are accepted.

- `mod.getXXX()`, i.e., without input argument, returns a *dictionary* with names (strings) as keys and values given in strings.
- `mod.getXXX(s)`, where `s` is a *string* of a name, returns a single string with value of the corresponding name.
- `mod.getXXX(v)`, where `v` is a *vector* of strings of names, returns a *vector* of strings of values for the corresponding names.

Set Methods The information that can be set is a subset of the information that can be get. Thus, we consider methods `setXXX()`, where `XXX` is either of `{Parameters, Inputs, SimulationOptions, LinearizationOptions}`, thus methods `mod.setParameters()`, `mod.setInputs()`, etc. Two calling possibilities are accepted.

- `mod.setXXX(s)`, with `s` being a string of keyword assignments of type quantity `"name = value"`. Here, the quantity `name` could be a parameter name, an input name, etc.
  - For parameters and simulation/linearization options, the value should be a single value such as a numerical value or a name of a solver, etc., e.g., `s` is `"R=8.31"` or `"solver=dassl"`.

[17]The reason why a dictionary with every name as key and time series as value is not returned, is that the amount of data might be exhaustive.

  - For inputs, the value could be a numerical value if the input is constant in the time range of the simulation, e.g., `"u = 1.0"`, or
  - For inputs, the value could alternatively be a vector of tuples `(t_{j},u_{j})`, i.e., `[(t1,u1),(t2,u2),...,(tN,uN)]` where the input varies linearly between `(t_{j},u_{j})` and `(t_{j+1},u_{j+1})`, where `t_{j}<= t_{j+1}`, and where at most two subsequent time instances `t_{j},t_{j+1}` can have the same value. As an example, `"u=[..., (1,10), (1,20), ...]"` describes a perfect jump in input value from value `10` to value `20` at time instance `1`.

- `mod.setXXX(v)`, with `v` being a vector of strings as described for `mod.setXXX(s)`. An example could be could be `["R=8.31","cp=4.18"]`.

2.3.5 Operating on Julia Object: Simulation

The following method operates on the object, and has no input arguments. The method has no return values; instead the results are stored within the object.

- `mod.simulate()` — simulates the system with the given simulation options

To retrieve the results, method `mod.getSolutions()` is used as described previously.

2.3.6 Operating on Julia Object: Linearization

The following methods are used for linearization:

- `mod.linearize()` — with no input argument, returns a tuple of 2D arrays (matrices) A, B, C, D.

- `mod.getLinearInputs()` — with no input argument, returns a vector of strings of names of inputs used when forming matrices B and D.

- `mod.getLinearOutputs()` — with no input argument, returns a vector of strings of names of outputs used when forming matrices C and D.

- `mod.getLinearStates()` — with no input argument, returns a vector of strings of names of states used when forming matrices A, B, C, D.

Observe that linearization is carried out at the `stopTime` specified in `LinearizationOptions`. The reason why linearization is not carried out at initial time, is that to handle DAEs, OpenModelica needs to initialize the model at initial time — before linearization can be carried out. For normal use, `stopTime` should be given a small value if linearization at the current operating value is intended.

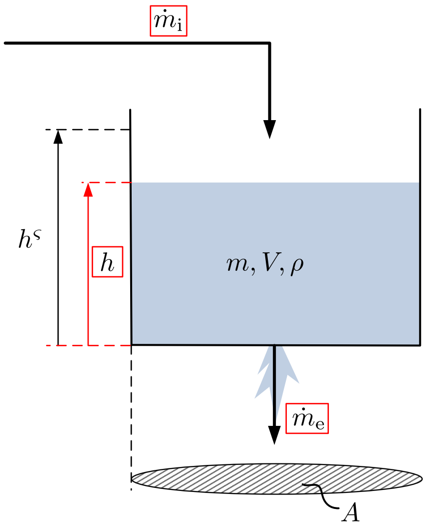


Figure 1. Driven water tank, with externally available quantities framed in red: initial mass is emptied through bottom at rate $\dot{m}_{\mathrm{e}}$, while at the same time water enters the tank at rate $\dot{m}_{\mathrm{i}}$.

2.3.7 Operating on Julia Object: Sensitivity

Sensitivy is related to $\frac{\partial y(t)}{\partial \theta}$, i.e., how an infinitesimal change in a parameter θ leads to an infinitesimal change in the solution of variable y; both θ and y can in principle be vectors. Sensitivity is very important in connection with model fitting and identifiability analysis. The following method is implemented on the Julia side, and provides *numeric* sensitivities. The method has 2 or 3 input arguments, and returns a tuple of 2 return arguments.

- `mod.sensitivity(a1,a2[,a3])` — computes sensitivity $\frac{\partial y(t)}{\partial \theta}$. Input arguments must be vectors: `a1` holds strings of the name of model parameters (θ), `a2` holds strings of the name of system variables (y), while the optional third argument `a3` holds floating point values for fractional parameter perturbation. The return tuple holds two vectors, `r1` and `r2`. The first vector, `r1`, holds strings of the name of the sensitivities that have been computed, while vector `r2` holds the corresponding time series (vector of solution values) — computed at the time instances given by the simulation options.

3 Basic Use of API for Model Analysis

3.1 Case: Simple Tank Filled with Liquid

We consider the tank in Figure 1 filled with water.

Water with initial mass $m(0)$ is emptied by gravity through a hole in the bottom at effluent mass flow rate $\dot{m}_{\mathrm{e}}$, while at the same time water is filled into the tank at influent mass flow rate $\dot{m}_{\mathrm{i}}$. Our modeling objective is to find the liquid level h. Here, the input variable is the influent mass flow rate $\dot{m}_{\mathrm{i}}$, while the output variable is the quantity we are interested in, h.

Table 1. Parameters for driven tank with constant cross sectional area.

Parameter	*Value*	*Comment*
ρ	$1\,\mathrm{kg/L}$	Density of liquid
A	$5\,\mathrm{dm}^2$	Constant cross sectional area
K	$5\,\mathrm{kg/s}$	Valve constant
h^{ς}	$3\,\mathrm{dm}$	Level scaling

Table 2. Operating condition for driven tank with constant cross sectional area.

Quantity	*Value*	*Comment*
$h(0)$	$1.5\,\mathrm{dm}$	Initial level
$m(0)$	$\rho h(0) A$	Initial mass
$\dot{m}_\mathrm{i}(t)$	$2\,\mathrm{kg/s}$	Nominal influent mass flow rate; may be varied

3.2 Model Summary

The model can be summarized in a form suitable for implementation in Modelica as

$$\frac{dm}{dt} = \dot{m}_\mathrm{i} - \dot{m}_\mathrm{e} \quad (1)$$

$$m = \rho V \quad (2)$$

$$V = Ah \quad (3)$$

$$\dot{m}_\mathrm{e} = K\sqrt{\frac{h}{h^{\varsigma}}} \quad (4)$$

To complete the model description, we need to specify model parameters and operating conditions. Model parameters (constants) are given in Table 1.

The operating conditions are given in Table 2.

3.3 Modelica Encoding of Model

The Modelica code describes the core model of the tank, `ModWaterTank`, and consists of a *first section* where constants and variables are specified, and a *second section* where the model equations are specified (compactified Modelica code is shown below).

```
model ModWaterTank
   constant Real rho = 1 "Density";
   parameter Real A=5, K=5, h_s=3;
   parameter Real h_0=1.5, m_0=rho*h_0*A;
   Real m(start=m_0, fixed=true);
   Real V, md_e;
   input Real md_i;
   output Real h;
equation
   der(m)=md_i-md_e;
   m=rho*V, V=A*h, md_e=K*sqrt(h/h_s);
end ModWaterTank;
```

As seen from the *first section* of model `ModWaterTank`, the model has 4 essential parameters (`rho`–`h_s`) of which one is a Modelica constant (`rho`) while other 3 are design parameters, compare this to Table 1. Furthermore, the model contains 2 "initial state" parameters, where 1 of them can be chosen at liberty, `h_0`, while the other one, `m_0`, is computed automatically from `h_0`, see Table 2. The purpose of the "free parameter" `h_0` is that it is easier for the user to specify level than mass. Also, free "initial state" parameters makes it possible for the user to change the initial states from outside of model `ModWaterTank`, e.g., from Julia.

Next, one variable is given with initial value — the state `m` — is initialized with the "initial state" parameter `m_0`. Then, 2 variables are defined as auxiliary variables (algebraic variables), `V` and `md_e`.[18]

One input variable is defined — `md_i` — this is the influent mass flow rate $\dot{m}_\mathrm{i}$, see Table 2. Inputs are characterized by that their values are not specified in the core model — here `ModWaterTank`. Instead, their values must be given in an external model/code — we will specify this input in Julia. Finally, 1 output is given — `h`.

In the *second section* of model `ModWaterTank`, the Model equations exactly map the mathematical model given in Eqs. 1–4. For illustrative purposes, the core model `ModWaterTank` is wrapped within a package named `WaterTank` and stored in file `WaterTank.mo`,

```
package WaterTank
// Package for simulating
   // driven water tank
   model ModWaterTank
      // Main driven water tank model
      // ...
      ...
   end ModWaterTank;
// End package
end WaterTank;
```

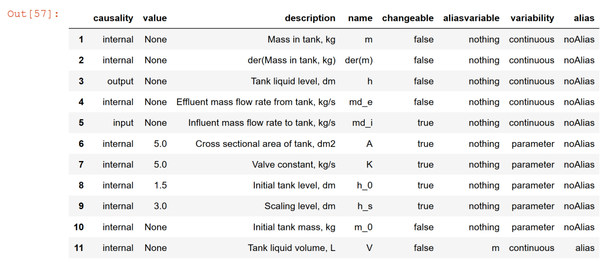
3.4 Use of Julia API

First, the following Julia statements are executed — we did this in Jupyter notebook (IJulia).

```
using Plots; pyplot()
using LaTeXStrings
using DataFrames
using OMJulia
# Linewidth
LW1 = 1.5
LW2 = 1
# Colors - core
usn_red = colorant"#D64349"
usn_blue = colorant"#27B2DO"
usn_green = colorant"#3BAFA2"
usn_purple = colorant"#4646A5"
usn_gold = colorant"#FFD240"
```

Here, package `Plots` is the plotting meta package of Julia; we use `pyplot` as back-end. Package `LaTeXStrings` makes it possible to automate insertion of escape symbol \ in LaTeX code to produce proper Julia strings. Package `DataFrames` is used to present `quantities` in Jupyter notebook tables. Two line widths

[18] `md` is notation for `m` with a dot, $\dot{m}$, i.e., a mass flow rate.


Out[57]:

	causality	value	description	name	changeable	aliasvariable	variability	alias
1	internal	None	Mass in tank, kg	m	false	nothing	continuous	noAlias
2	internal	None	der(Mass in tank, kg)	der(m)	false	nothing	continuous	noAlias
3	output	None	Tank liquid level, dm	h	false	nothing	continuous	noAlias
4	internal	None	Effluent mass flow rate from tank, kg/s	md_e	false	nothing	continuous	noAlias
5	input	None	Influent mass flow rate to tank, kg/s	md_i	true	nothing	continuous	noAlias
6	internal	5.0	Cross sectional area of tank, dm2	A	true	nothing	parameter	noAlias
7	internal	5.0	Valve constant, kg/s	K	true	nothing	parameter	noAlias
8	internal	1.5	Initial tank level, dm	h_0	true	nothing	parameter	noAlias
9	internal	3.0	Scaling level, dm	h_s	true	nothing	parameter	noAlias
10	internal	None	Initial tank mass, kg	m_0	false	nothing	parameter	noAlias
11	internal	None	Tank liquid volume, L	V	false	m	continuous	alias

Figure 2. Typesetting of quantity vector of dictionaries as a table in a Jupyter notebook.

are assigned, to variables `LW1` and `LW2`, to obtain uniform line width.

Colors are taken from the graphical profile of the employer of first author are used to illustrate how one can define colors using `hex` code. Alternatively, the CSS color names are available[19] as case insensitive symbols, e.g., `:red`, `:cornflowerblue`, etc.

3.5 Basic Simulation of Model

We instantiate object tank with the following command:

```
tnk = OMJulia.OMCSession()
tnk.ModelicaSystem("WaterTank.mo","WaterTank.
    ModWaterTank")
```

In the sequel, Julia prompt `julia>` is used when Jupyter[20] notebook actually uses `In[*]` — where `*` is some number, while the response in Jupyter notebook is prepended with `Out[*]` .

```
julia> q = tnk.getQuantities()
julia> typeof(q)
Array{Any,1}
julia> length(q)
11
julia> q[1]
Dict{Any,Any} with 8 entries:
  "name"          => "m"
  "aliasvariable" => nothing
  "variability"   => "continuous"
  "changeable"    => "false"
  "causality"     => "internal"
  "value"         => "None"
  "description"   => "Mass in tank, kg"
  "alias"         => "noAlias"
julia> tnk.showQuantities()
```

Method `tnk.showQuantities()` produces a table overview, Fig. 2.

The results in Figure 2 should be compared to the Modelica model in Section 3.2. Observe that Modelica constants are not included in the quantity list.

Next, we check the simulation options:

```
julia> tnk.getSimulationOptions()
```

```
Dict{Any,Any} with 5 entries:
  "startTime" => "0"
  "stopTime"  => "1"
  "solver"    => "dassl"
  "stepSize"  => "0.002"
  "tolerance" => "1e-006"
```

It should be observed that the `stepSize` is the frequency at which solutions are *stored*, and is *not* the step size of the solver. The number of data points stored, is thus (`stopTime−startTime`)/`stepSize` with due rounding. This means that if we increase the `stopTime` to a large number, we should also increase the `stepSize` to avoid storing large amounts of data.

Possible inputs are:

```
julia> tnk.getInputs()
Dict{Any,Any} with 1 entry:
  "md_i" => "None"
```

where value `None` implies that the available input, `md_i`, has yet not been set. The simulation will not work with value `None`; let us instead set $\dot{m}_{\mathrm{i}} = 3$, simulate for a long time, and then change "initial state" parameter `h(0)` to the steady state value of `h`:

```
julia> tnk.setInputs("md_i=3")
julia> tnk.setSimulationOptions(["stopTime=1
    e4", "stepSize=10"])
julia> tnk.simulate()
julia> h, = tnk.getSolutions("h")
julia> tnk.setParameters("h_0=$(h[end])")
```

Observe that the syntax `h,` is needed to unpack the time series for `h` when the vector of solutions has a single element.

Next, we reset the stop time to 10, and specify an input sequence with a couple of jumps:

```
julia> tnk.setSimulationOptions(["stopTime=10
    ","stepSize=0.02"])
julia> tnk.setInputs("md_i = [(0,3),(2,3),
    (2,4),(6,4),(6,2),(10,2)]")
```

Finally, we simulate the model with the time varying input, and plot the result:[21]

```
julia> tnk.simulate()
julia> tm, h = tnk.getSolutions(["time","h"])
julia> plot(tm,h,linewidth=LW1, color=
    usn_blue, label=L"$h$")
julia> plot!(title="Water tank level")
julia> plot!(xlabel=L"time $t$ [s]")
julia> plot!(ylabel=L"$h$ [dm]")
```

The result is displayed in Figure 3.

3.6 Monte Carlo Simulation

It is of interest to study how the model behavior varies with varying uncertain parameter values, e.g., the effluent valve constant K. This can be done as follows:

[19] `https://www.w3schools.com/colors/colors_groups.asp`

[20] Jupyter is denoted `IJulia` in Julia.

[21] `plot()` plots a result, `plot!()` overlays information on an existing plot.

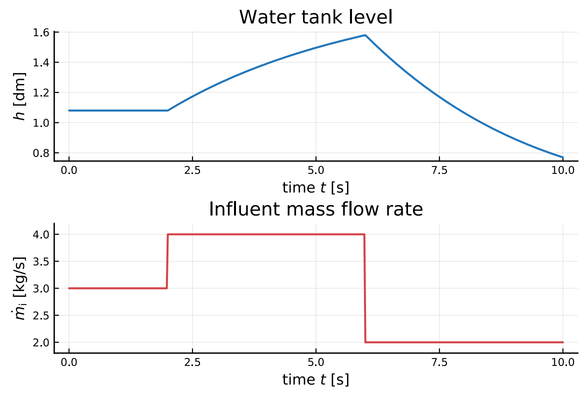


Figure 3. Tank level when starting from steady state, and $\dot{m}_{\mathrm{i}}(t)$ varies in a straight line between the points $(t_j, \dot{m}_{\mathrm{i}}(t_j))$ given by the list `[(0,3),(2,3),(2,4),(6,4),(6,2),(10,2)]`.

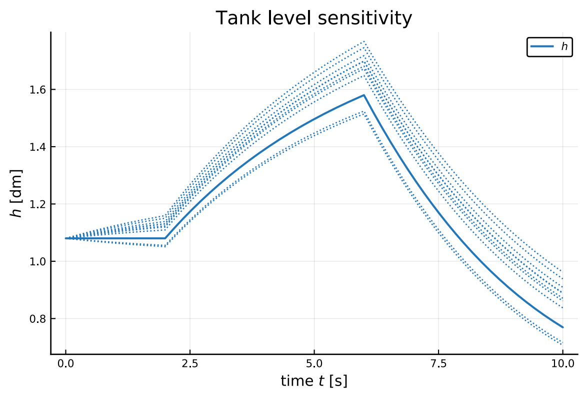


Figure 4. Uncertainty in tank level with a 5% uncertainty in valve constant K. The input is like in Figure 3.

```
julia> par = tnk.getParameters()
julia> K = parse(Float64,par["K"])
julia> Nmc = 10;
julia> KK = K + (randn(Nmc)-0.5)*K/20;
julia> tnk.simulate()
julia> tm, h = tnk.getSolutions(["time","h"])
julia> v_h = Vector{Vector}(Nmc)
julia> for (i,K) in enumerate(KK)
           tnk.setParameters("K=$(K)")
           tnk.simulate()
           v_h[i] = tnk.getSolutions("h")
       end
julia> plot(tm,h,lw=LW1,lc=usn_red,label=L"
    $h$")
julia> plot!(tm,v_h,lw=LW2,ls=:dot,lc=usn_red
    ,legend=false)
julia> plot!(title="Tank level sensitivity")
julia> plot!(xlabel=L"time $t$ [s]")
julia> plot!(ylabel=L"$h$ [dm]")
```

The result is as shown in Figure 4.

3.7 Linearizing Model

We can find a linearized approximation of the system. First we reset K to 5, then set the stop time of the linearization to 10^{-6} before we linearize the system and extract matrices A, B, C, and D.

```
julia> tnk.setParameters("K=5")
julia> tnk.setLinearizationOptions("stopTime
    =1e-6")
julia> A,B,C,D = tnk.linearize();
```

If we use the Julia `ControlSystems` package,[22] we can define an LTI system and find the transfer function:

```
julia> using ControlSystems
julia> sys = ss(A,B,C,D)
julia> tf(sys)
ControlSystems.TransferFunction{
    ControlSystems.SisoRational{Float64}}
           0.2
---------------------------
1.0*s + 0.2587650960551352

Continuous-time transfer function model
```

We may also like to know the state which OpenModelica has chosen:

```
julia> tnk.getLinearStates()
1-element Array{Any,1}:
 "m"
```

4 Case study: PI control of reactor

4.1 Reactor

We consider an extension of a reactor described in (Seborg et al., 2011); see (Sund et al., 2018), (Khalili and Lie, 2018) for details of the model and linearization of the model. The reactor is exothermal with water cooling via a heat exchanger, and is unstable at the operating point. The original model (`org`) in (Seborg et al., 2011) has 2 states: reactor temperature T and concentration c_{A} of species A. An extended model which only assumes ideal solution (`is`) has 3 states: the states of the `org` model as well as concentration c_B of species B. Both models exhibit nonlinear oscillations when forced away from the equilibrium point. A possible control problem is to control the reactor temperature T by means of the cooling water temperature T_{c} of the heat exchanger.

4.2 PI Controller

A linearized model can easily be found by using the `mod.linerize()` method of `OMJulia` — the linearized model is as in (Khalili and Lie, 2018), with cooling temperature T_{c} as control input. The closed loop matrix A_{cl} with a proportional controller (P controller) is

$$A_{\mathrm{cl}} = A - K_{\mathrm{p}} BC \tag{5}$$

where B is the input matrix and K_{p} is the controller gain. Looping through $K_{\mathrm{p}} \in [-1, 8]$ leads to the closed loop eigenvalues as depicted in Figure 5.[23]

[22] A similar tool in Python is limited in scope, and rather complicated to install.

[23] Here, Julia's `ControlSystems` package has been used, together with a user-modified `rlocus()` function.

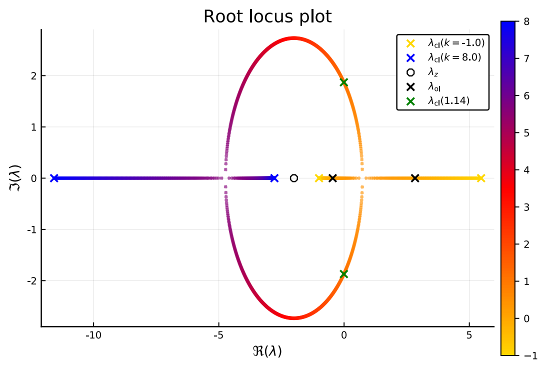


Figure 5. Root locus plot $\lambda\left(A_{\mathrm{cl}};K_{\mathrm{p}}\right)$ for $K_{\mathrm{p}} \in [-1,8]$.

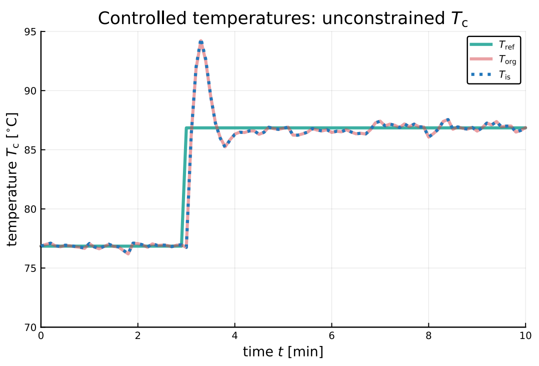


Figure 6. Output T as controlled with PI controller tuned for `org` model, and applied to `org` and `is` model.

The P-controller stabilizes the system for $K_{\mathrm{p}} \gtrapprox 1.14$; $K_{\mathrm{p}} = 5.7$ gives two real, closed loop eigenvalues/poles at approximately $\lambda \approx -5$, which implies closed loop time constants $\tau_j \approx \frac{1}{5} = 0.2$.

For a proportional + integral controller, it is reasonable to let the reset time (= integral time) be, say, 10 times larger than the closed loop time constants of a P controller. Thus, the PI controller

$$T_{\mathrm{c}}(s) = T_{\mathrm{c}}^{*} + K_{\mathrm{p}}\frac{1+T_{\mathrm{int}}s}{T_{\mathrm{int}}s}\cdot e(s) \qquad (6)$$

$$e(s) = T_{\mathrm{ref}}(s) - T(s) \qquad (7)$$

with $K_{\mathrm{p}} = 5.7$ and $T_{\mathrm{int}} = 2$ may be an acceptable choice.[24] Nominal input T_{c}^{*} is not needed with integral action, but is useful to avoid an initial "kick" in the control action. T_{ref} is the reference temperature. If we let $T_{\mathrm{int}} \to \infty$, the controller becomes a P controller.

In the time domain, we can express the PI controller as

$$T_{\mathrm{c}} - T_{\mathrm{c}}^{*} = K_{\mathrm{p}}e + \tilde{T}_{\mathrm{c}} \qquad (8)$$

$$\frac{d\tilde{T}_{\mathrm{c}}}{dt} = \frac{K_{\mathrm{p}}}{T_{\mathrm{int}}}e. \qquad (9)$$

To handle constraints for $T_{\mathrm{c}} \in [4,96]\,^{\circ}\mathrm{C}$, if $T_{\mathrm{c}} = K_{\mathrm{p}}e + T_{\mathrm{c}}^{*} + \tilde{T}_{\mathrm{c}}$ breaks this constraint, we set T_{c} equal to the constraint and $\frac{d\tilde{T}_{\mathrm{c}}}{dt} = 0$ to avoid controller wind-up. The controller is implemented in Modelica, but controller parameters and constraints in T_{c} are set from Julia using OMJulia.

4.3 Proportional + Integral Control

Figure 6 shows the use of a PI controller to keep reactor temperature T close to a reference T_{ref}. The PI controller tuned for the `org` model, is also applied to the `is` model.

The result indicates that the controller easily handles the model difference between the two models. Figure 7 shows the applied control input T_{c} as well as the integral state $\bar{T}_{\mathrm{c}}$ in the controller for the two model cases.

Figure 7 clearly shows a problem for the controller: the cooling water can not take on negative temperatures T_{c}

[24]The integral time is denoted T_{int} in order to make a distinction between integral time and influent temperature, T_{i}.

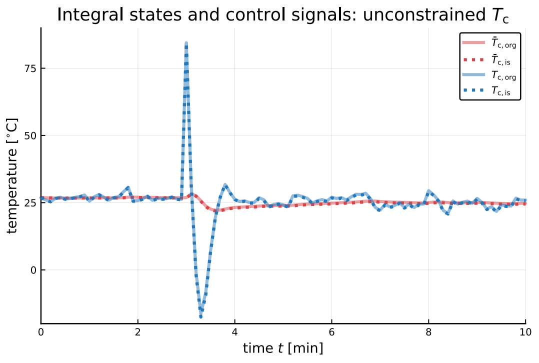


Figure 7. PI control signal T_{c} and integrator state $\bar{T}_{\mathrm{c}}$ for `org` and `is` models.

[°C]. We therefore add the constraint that $T_{\mathrm{c}} \in [4,96]\,^{\circ}\mathrm{C}$, which together with anti-windup leads to the results in Figures 8 and 9 for output T and controller T_{c}, respectively.

5 Discussion and Conclusions

This paper presents OMJulia, a first effort to interface a relatively complete Modelica tool, OpenModelica, to Julia, giving access to an open source set-up for modeling and analysis, including control synthesis, easily installable from a unified package manager.

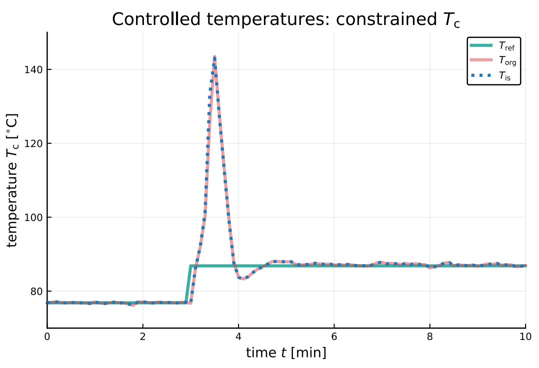


Figure 8. Output T as controlled with PI controller tuned for `org` model, and applied to `org` and `is` model: control input T_{c} is constrained to $[4,96]\,^{\circ}\mathrm{C}$ and anti-windup is applied.

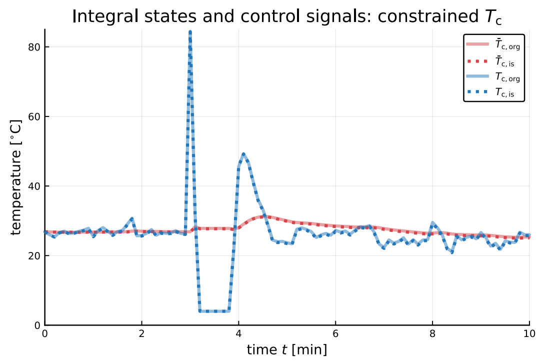


Figure 9. PI control signal T_c and integrator state $\bar{T}_c$ for `org` and `is` models: control input T_c is constrained to $[4,96]$ °C and anti-windup is applied.

Some design choices of the Julia API are briefly described, and the syntax and possibilities of OMJulia are then detailed. The use of the API is illustrated with a simple example of a water tank model, then some possibilities for control analysis of a chemical reactor are detailed. The API has also been tested on more complex models not shown here.

The key contribution of the OMJulia package is not within Modelica as a language, but rather to increase the usefulness of Modelica into new fields such as control engineering. Future work will include a package OMMatlab, updating the syntax of OMPython, and possibly extension of the API to the optimization and symbolic sensitivity analysis routines in OpenModelica. Another possibility is to consider a translator from OpenModelica to Julia (design choice 3).

References

Marcus Baur, Martin Otter, and Bernhard Thiele. Modelica Libraries for Linear Control Systems. In *Proceedings, the 7th International Modelica Conference*, Como, Italy, 2009.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Sha. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 49(1):65–98, 2017. doi:10.1137/14100067.

K. E. Brenan, S. L. Campbell, and Linda R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. North-Holland, New York, 1989.

Hilding Elmqvist, Toivo Henningsson, and Martin Otter. Innovations for Future Modelica. In *Proceedings of the 12th International Modelica Conference*, Prague, Czech Republic, May 2017. doi:10.3384/ecp17132693. May 15-17, 2017, Prague, Czech Republic.

FMI Consortium. Functional Mock-up Interface for Model Exchange, version 2.0, 2018. URL `https://fmi-standard.org/`.

Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley-IEEE Press, Piscataway, NJ, second edition, 2015. ISBN 978-1-118-85912-4.

Peter Fritzson, Adrian Pop, Adeel Asghar, Bernhard Bachmann, Willi Braun, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Alejandro Danós, Rüdiger Franke, Mahder Gebremedhin, Bernt Lie, Alachew Mengist, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, Volker Waurich, and Per Östlund. The OpenModelica Integrated Modeling, Simulation and Optimization Environment. In *Proceedings of the 1st American Modelica Conference*, Cambridge, MA, USA, October 2018. LIU Electronic Press, www.ep.liu.se. October, 8-10, 2018.

Anand Kalaiarasi Ganeson. Design and Implementation of a User Friendly OpenModelica - Python interface. Master's thesis, Linköping University, 2012.

Anand Kalaiarasi Ganeson, Peter Fritzson, Olena Rogovchenko, Adeel Asghar, Martin Sjölund, and Andreas Pfeiffer. An OpenModelica Python Interface and its Use in PySimulator. In *Proceedings of the 9th International Modelica Conference*, September 2012. doi:10.3384/ecp12076537. September 3-5 2012.

Pieter Hintjens. *ZeroMQ. Messaging for Many Applications*. O'Reilly Media, March 2013.

JuliaLang. The Julia Programming Language, 2018. URL `https://julialang.org/`.

Mohammad Khalili and Bernt Lie. Comparison of Linear Controllers for Nonlinear, Open-loop Unstable Reactor. In *Proceedings, SIMS 2018*, Oslo Metropolitan University, September 2018. SIMS, Linköping University Press.

Bernt Lie, Sudeep Bajracharya, Alachew Mengist, Lena Buffoni, Arunkumar Palanisamy, Martin Sjölund, Adeel Asghar, Adrian Pop, and Peter Fritzson. API for Accessing OpenModelica Models from Python. In *Proceedings of EuroSim 2016*, Oulu, Finland, 2016, September 2016.

Modelica Association. The Modelica Standard Library, v. 3.2.2, 2016. URL `https://github.com/modelica/ModelicaStandardLibrary/`.

Modelica Association. Modelica® — a Unified Object Oriented Language for System Modeling Language Specification, version 3.4, 2017. URL `https://modelica.org/documents/ModelicaSpec34.pdf`.

Christopher Rackauckas and Qing Nie. DifferentialEquations.jl — A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of Open Research Software*, 5(15), 2017. DOI: http://doi.org/10.5334/jors.151.

Dale E. Seborg, Thomas F. Edgar, Duncan A. Mellichamp, and III Doyle, Frank J. *Process Dynamics and Control*. John Wiley & Sons, Hoboken, NJ, third edition edition, 2011. ISBN 978-0-470-12867-1. ISBN 978-0-470-12867-1.

Sveinung M. Sund, Marianne Plouvier, and Bernt Lie. Comparison of Simulation Tools for Dynamic Models. In *Proceedings, SIMS 2018*, Oslo Metropolitan University, September 2018. SIMS, Linköping University Press.

“hello, (Modelica) world”: Automated documentation of complex simulation models exemplified by expansion valves

Christian Vering   Sven Hinrichs   Moritz Lauster   Dirk Müller

Institute for Energy Efficient Buildings and Indoor Climate, RWTH Aachen University, Germany,
cvering@eonerc.rwth-aachen.de

Abstract

The constantly increasing computing power enables the implementation of complex simulation models. Therefore, it is possible to create more detailed models to predict system behavior more accurately. Modelica, for example, has proven great suitability in modelling complex systems, because of its high degree of reusability. However, understanding these models is quite difficult and many simulation models are poorly documented. Consequently, it is very time-consuming to retrace given model structures especially for novice. The Unified Modeling Language (UML) provides a user-friendly and graphical structure for documentation to simplify working with existing simulation models. Hence, an algorithm (ADoCSM) is developed to automatically present the structure of a Modelica simulation model in UML. This algorithm is exemplarily applied to a refrigerant circuit expansion valve model. Thereby, we contribute to an increase of simulation model quality as well as simplifying the entry in the world of Modelica. ADoCSM and the expansion valve model are freely available on GitHub:

https://github.com/RWTH-EBC/ADoCSM
https://github.com/RWTH-EBC/AixLib/tree/issue590_ExpansionValve

Keywords: Modelica introduction, simplify modelling, automated model documentation

1 Introduction

In the last decades, modelling complex systems has gained importance. In engineering, we utilize modelling in order to support designing processes or to enable the application of sophisticated control strategies like model predictive control. Therefore, detailed simulation models with high software quality are necessary. However, careful modelling is time-consuming and the documentation of models is exhausting.

In the context of building energy systems’ heat supply, heat pumps are awarded to be a key technology supporting the achievement of stated climate goals in this sector (EEA, 2016). The heat pumps’ lifetime strongly depends on the operation of its compressor. Avoiding droplet impact within the compression process, the heat pumps’ expansion valves adjust a level of superheat of the refrigerant at the compressor inlet (Jahnke, 2000). Thus, the expansion valve is very important for the lifetime of heat pumps. Hence, modelling expansion valves is essential to understand its behavior within the refrigerant circuit and increase the lifetime of the compressor by advanced expansion valve control.

Due to superposition of thermodynamic and fluid mechanic interactions, modelling expansion valves is challenging (Cao, 2016). In particular, the complexity of superposition makes good documentation necessary. Therefore, the expansion valve modelling and concurrent documentation offers a suitable application to show functionality of the presented algorithm ADoCSM (“Automated Documentation of Complex Simulation Models”). In order to ensure both a well-defined simulation model and a well-documented one, a modelling as well as a documentation language need to be chosen.

One suitable modelling language for thermal systems is Modelica, because of its DAE-based modelling options as well as its high degree of reusability.

Many approaches for modelling of thermal systems like expansion valves already exist and are utilized in literature. Thereby, two approaches are common. On the one hand, mathematical black box approaches are applied to ensure short simulation times by sufficient prediction accuracy for individual refrigerants (Müller, 2016). Going for modular and scalable models on the other hand, grey box approaches include physical behavior by applying fundamental equations to predict change of states. As a result, prediction accuracy can be increased for a wider range of refrigerants by simultaneous loose of computational speed (Müller, 2016). Joining presented advantages, we show automated documentation of a modular and scalable expansion valve simulation model.

A well-known and well-established model documentation approach is using the graphical Unified Modeling Language (“UML”) (Weilkiens, 2006). In order to utilize UML for Modelica many tools already exist (Loeffler, 2006). However, an automation algorithm is not known.

Therefore, reducing entry barriers into both the structure of Modelica as well as our algorithm, we present an open-source solution that is applicable with a graphical

user interface (GUI). Hence, we want to educate users in understanding Modelica as well as to improve our algorithm to be suitable for many use-cases as easy as possible.
Thus, within this paper, we contribute to three main ideas by breaking up rather complicated structures:

1) Modular and scalable modelling approach for expansions valves on refrigerant level is set up in Section 2, combining thermodynamic and fluid mechanic effects by consideration of
    a. metastable coefficient of mass flow and transition of two-phase flow and
    b. choke effects.
2) In Section 3, we present the algorithm that translates Modelica code into a UML code, which enables a structured and automated documentation of complex simulation models.
3) Reducing entry barriers in Modelica and the use of the algorithm, we show in Section 4 a GUI that easily offers all functionalities of the current state of our work.

In the end of the paper, we conclude the work and discuss the outlook for further developments.

2 Modeling expansion valves

Heat pumps are a key technology coupling sectors of heat and electricity regarding the building stock (Huchtemann, 2009). This enables a systematic electrification of heat supply in order to increase the flexibility of the energy system. Exploiting the whole potential of this technology, detailed simulation models are necessary to predict system behavior. Therefore, Modelica has proven great suitability.
Figure 1 depicts a schematic of a modular and scalable heat pump in Modelica (Dymola) that strictly separates physical system description from system control. Further information about this grey-box approach is published in Storek et al. (Storek, 2018) and Vering et al. (Vering, 2018).

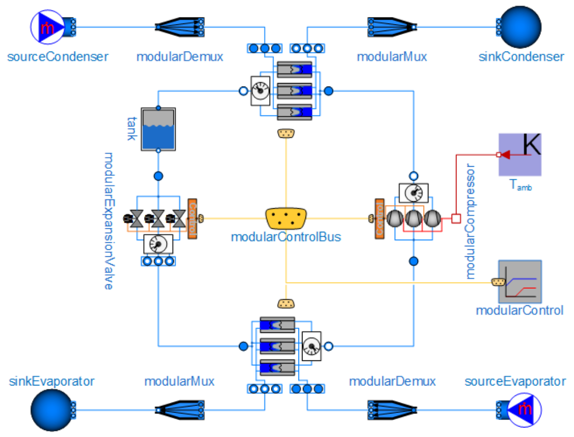


Figure 1: A modular and scalable heat pump in Modelica showing the main physical components separated from control blocks.

The expansion valve (EV) is an important component to ensure mass flow adjustment within the refrigerant circuit. The mass flow directly correlates to the refrigerant mass in the evaporator and dwell times in this component. Thus, EV allows controlling the level of superheated vapor at evaporator outlet and thereby at the inlet of the compressor.
Compressors’ lifetime can be reduced by supplying it with non-superheated vapor, which causes droplet impact (Jahnke, 2000). Therefore, it is necessary to avoid those refrigerant conditions. Hence, controlling the level of superheat is important while operating a heat pump. As consequence, a resilient prediction of the fluid state at compressor inlet using simulation models requires careful modelling.
The expansion valve throttles the refrigerant from a high-pressure level to a lower one by decreasing the flow area with a positioning cylinder as shown in Figure 2. In this process, a superposition of thermodynamic and fluid mechanics effects occurs, which is a complex phenomenon (Huo, 2010). The main effects defining this process are choke, flash, cavitation and evaporation waves (Moreira, 2003). Modelling all physical interactions is not recommended in literature as well as just using the Bernoulli equation for an incompressible fluid and frictionless fluid flow (Cao, 2016).
In consistence to the whole heat pump model, we choose a grey-box modelling approach to estimate refrigerant states. Therefore, basic physical equations as well as some assumptions are covered and implemented. The mass flow $\dot{m}$ through an EV considering expansion by an expansion factor Y can be written as (Davies, 1973):

$$\dot{m} = C_{\mathrm{d}} Y A_{\mathrm{th}} \sqrt{2\,\rho_{\mathrm{in}}(p_{\mathrm{in}} - p_{\mathrm{out}})}, \tag{1}$$

$$Y = 1 - \frac{p_{\mathrm{in}} - p_{\mathrm{out}}}{3 F_{\mathrm{Y}} X_{\mathrm{T}}}. \tag{2}$$

C_{d} describes the flow coefficient, A_{th} means flux area, ρ_{in} fluid density at inlet and p_i is the pressure of the fluid at inlet and outlet. $F_{\mathrm{Y}} X_{\mathrm{T}}$ is the product of a specific heat ratio F_{Y} and a pressure drop factor of the valve X_{T}.

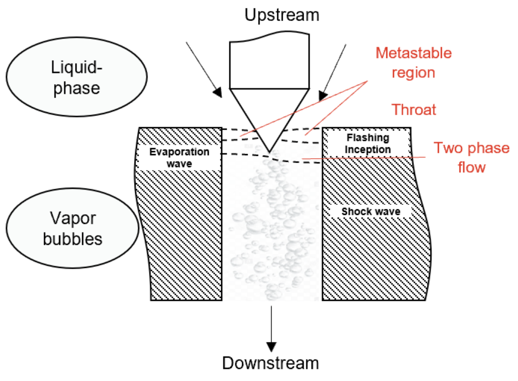


Figure 2: Schematic of fluid phenomina within an expansion valve showing the throtteling process for refrigerants with phase change.

For a fluid flow with constant density without phase change Equation 1 is simplified by $Y = 1$ to Bernoulli Equation (Li, 2013).
Modelling a choked mass flow with constant EV inlet pressure p_{in} Equation 1 can be written as

$$\dot{m} = \frac{2}{3} C_{\mathrm{d}} A_{\mathrm{th}} \sqrt{2\, \rho_{\mathrm{in}} p_{\mathrm{in}} F_{\gamma} X_{\mathrm{T}}}. \qquad (3)$$

It is obvious, that a two phase mass flow is always lower than for a one phase flow. Regarding detailed modelling approaches of the flow coefficient, we refer to (Li, 2013).
We implemented Equation 3 in Modelica to model the physical behavior of an expansion valve and the choke effect of the refrigerant considering two phase flow.
Showing the main governing equations, interdependencies between different variables occur, which are related to the reformulation of the problem. The modeler exactly knows which variable and equation stands for what. However, making a model open-source available, not only the modeler uses the model, but also end users. Simplifying the use of a model, a structured documentation is necessary. Within this work we chose UML to be a suitable graphical way of documentation because of its dissemination and establishment. Enabling the use of UML, an algorithm that translates Modelica code into an UML readable format is required. This algorithm is presented in Section 3.

The expansion valve model is freely available on GitHub:
https://github.com/RWTH-EBC/AixLib/tree/issue590_ExpansionValve

3 Automated Documentation of Complex Simulation Models

ADoCSM stands for "Automated Documentation of Complex Simulation Models". It is a tool that scans Modelica libraries and translates them into a code, which can be interpreted by UML language. Therefore, we use the freely available tools PlantUML and Papyrus because they support 9 (Papyrus 13) different UML diagrams and they are easily extendable (PlanUML, 2018).

These tools need to be interconnected via a parser to translate Modelica main functions into UML readable and interpretable code.
The parser is based on the Python programming language and has to recognize keywords from Modelica that relate to various Modelica functions.
We started implementing four main relations "Inheritance", "Aggregation", "Composition" and "Polymorphism" that are summarized in Table 1, which are key ideas of object-oriented modelling and allow a high level of reusability. Inheritance passes all methods and attributes from a parent to the child class and is initialized with the keyword "extends". In the composition several objects are assembled into a new overall system and is initialized with the "*modelname"*.
The relation polymorphism is used to specify interchangeable classes or object types that can be exchanged later in the parameterization.

Table 1: Relations in Modelica that the parser finds and translates into UML code.

Relation	Modelica key word	UML Notations
Inheritance	extends	───▷
Aggregation	outer / inner	───◇
Composition	model	───◆
Polymorphism	replaceable model	- - - -▷

In a pre-processing step, a structured reformulation of the Modelica code decreases the parser's error rate. Hence, we introduce two steps before starting the parser. In a first step, the path of the Modelica model is defined. This allows a systematic naming of packages and classes. After that, we translate a well-defined Modelica code into an easily parser readable formulation. Therefore, we choose a structure that follows these designs:

<< Variable >> << Comment >> << Annotation >>
<< Method >> << Comment >> << Annotation >>

Using the new code, the workflow of the parser is shown in Figure 3. Further reducing error rates, there is a first check whether a library exists. After that there is a lookup for the existing files.

Within this, file packages are declared. A package with its models is taken to be translated into the UML notation. Therefore, different key words are translated into a PlantUML format:

- Method
- Attribute
- Stereotype (model, record, type, block, function, connector)

Using these three inputs, the parser defines all relations within this model and repeats these steps for the whole list of files.
After passing all steps, the parser analyses the next layer i. The user defines the number of layers beforehand. Finally, the Modelica code is translated into a UML readable notation that can be interpreted e.g. by PlantUML. The converted Modelica code can now be loaded into the PlantUML tool and will be graphically generated in the UML form.
Thereby, the model structure can be broken up in order to get an overview of the model.

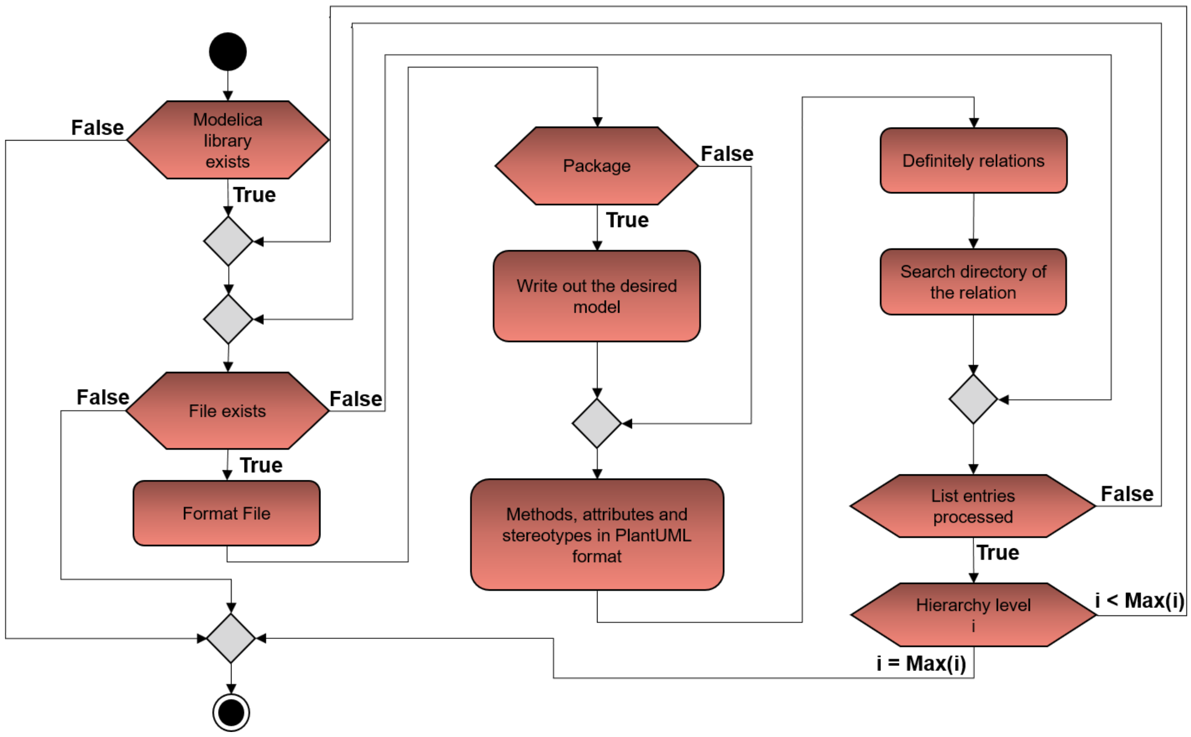
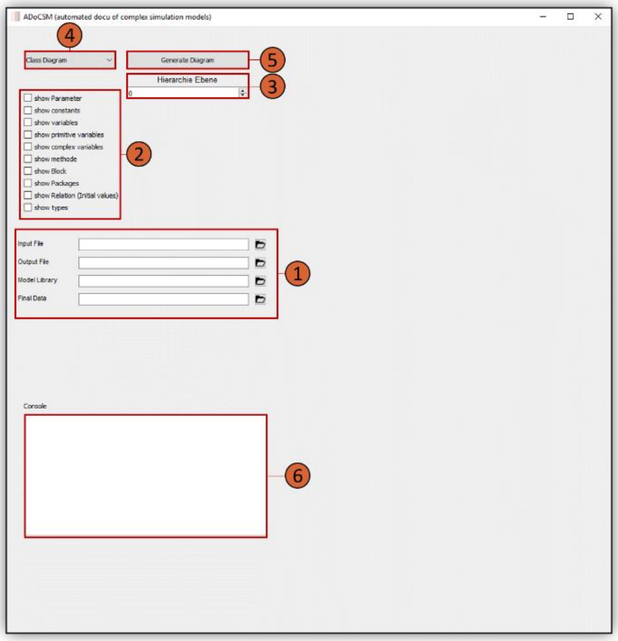


Figure 3: Workflow of the parser translating Modelica code into an UML readable notation.

With the aid of the UML class diagram, the structure of the model can be displayed graphically. For example, inherited models, parameters or exchangeable classes can be displayed. The visual representation reduces the complexity of the model and the relationships and composition of the model can be detected more quickly. Furthermore, simplifying the use of this algorithm and disseminating it, we develop a graphical user interface (GUI) that is presented in the next section by applying UML documentation to the expansion valve simulation model.

4 Use of ADoCSM

Supporting the entry in the world of Modelica, by making the structure of Modelica models more transparent, the parser translates Modelica code into the UML notation. For a user-friendly design, we develop a graphical user interface (GUI).
The GUI of ADoCSM is mainly separated in six parts, which are shown in Figure 4. These are initialization (1), settings (2-5) and console (6).
For "initialization" the input file (1) has to be chosen. With this file the paths of the files as well as libraries are defined for the parser. This example uses the *AixLib* (D. Müller, et al. 2016) and the Modelica standard library in the Model Library directory.
Additionally, the result file path can be redefined.
After that, the "settings" for the parser are selected. The user can choose, which information e.g. parameter, constants or variables (2) of a model should be plotted within the UML representation. In particular for complex models with a large number of parameters, constants or variables, it is helpful to show or hide a specific group of attributes in order to retain the overall overview. In particular, depicting model structures with a high degree of polymorphism, whole packages can be superimposed or hided out to keep overview. Additionally, the user immediately learns, which packages can be replaced by other ones.



0

- [ ] show Parameter
- [ ] show constants
- [ ] show variables
- [ ] show primitive variables
- [ ] show complex variables
- [ ] show methode
- [ ] show Block
- [ ] show Packages
- [ ] show Relation (Initial values)
- [ ] show types

Input File
Output File
Model Library
Final Data

Figure 4: Graphical User Interface (GUI) to simplify the use of ADoCSM.

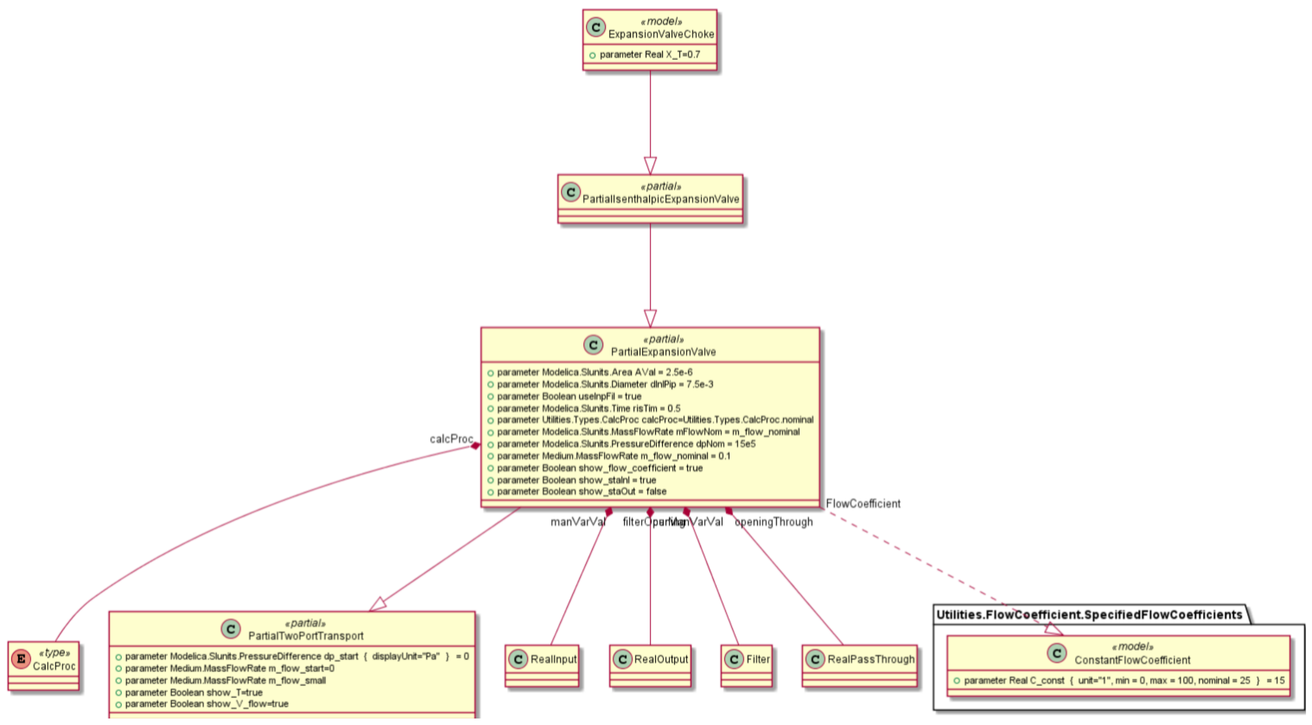

«model»
ExpansionValveChoke
parameter Real X_T=0.7
«partial»
PartialIsenthalpicExpansionValve
«partial»
PartialExpansionValve
parameter Modelica.SIunits.Area AVal = 2.5e-6
parameter Modelica.SIunits.Diameter dInlPip = 7.5e-3
parameter Boolean useInpFil = true
parameter Modelica.SIunits.Time risTim = 0.5
parameter Utilities.Types.CalcProc calcProc=Utilities.Types.CalcProc.nominal
parameter Modelica.SIunits.MassFlowRate mFlowNom = m_flow_nominal
parameter Modelica.SIunits.PressureDifference dpNom = 15e5
parameter Medium.MassFlowRate m_flow_nominal = 0.1
parameter Boolean show_flow_coefficient = true
parameter Boolean show_staInl = true
parameter Boolean show_staOut = false
calcProc
manVarVal
filterOpening
partManVarVal
openingThrough
FlowCoefficient
«type»
CalcProc
«partial»
PartialTwoPortTransport
parameter Modelica.SIunits.PressureDifference dp_start { displayUnit="Pa" } = 0
parameter Medium.MassFlowRate m_flow_start=0
parameter Medium.MassFlowRate m_flow_small
parameter Boolean show_T=true
parameter Boolean show_V_flow=true
RealInput
RealOutput
Filter
RealPassThrough
Utilities.FlowCoefficient.SpecifiedFlowCoefficients
«model»
ConstantFlowCoefficient
parameter Real C_const { unit="1", min = 0, max = 100, nominal = 25 } = 15

Figure 5: Representation of an expansion valve in an UML class diagram that was translated from Modelica code showing the model structure and the relations.

Regarding a structured representation, the number of investigation layers (3) is set and UML diagram type (4) can be chosen.
By default, a class diagram is predefined. For further work, it should be considered whether additional diagram types should be added. Pushing the button (5) executes the generation of the UML structure regarding defined paths and settings. Using the expansion valve as example, in Figure 5 the referring UML class diagram is depicted with all relations and model parameter. The “console” (6) shows in the end, if a workflow was successfully executed or if errors occurred. A successful execution is then shown in a separate window.
In Figure 4 all structure investigation layers are depicted. On top, in the first layer there is the first box of the expansion valve model with respect to choke effects. The box is regarding UML class diagram requirements divided into three parts. In the upper one the indication is shown as a “model” named “ExpansionValveChoke”. In the lower one the user can see the corresponding parameter X_{T}. The third box is not required in this case, because no operators are defined within this model. It inherits its properties from the next layer, where the partial model “PartialIsenthalpicExpansionValve” is shown.
On the next deeper layer, “PartialExpansionValve” with all typical properties of an expansion valve is illustrated. Different parameter such as “Area” or “Diameter” are defined with a value and the corresponding SI unit. Additionally, parameter of the type Boolean like “useInpFil” are shown.
In the last layer, the inheritances, compositions and a polymorphism are pointed out. The only inheritance of a “PartialTwoPortTransport” model is revealed. A combination of “CalcProc”, “RealInput”, “RealOutpt”, “Filter” and “RealPassThrough” constitutes all model compositions. Furthermore, a polymorphism “ConstantFlowCoefficient” is shown. As consequence, the user immediately knows, that this is a replaceable model within the “ExpansionValveChoke” model.
Compared to typical Modelica tree structures, the UML class diagram easily illustrates the expansion valve simulation model structure. That supports the understanding of inheritances and compositions as well as polymorphism. Additionally, parameter and initial values are revealed by the graphical representation of the model.
All these points simplify the understanding of complex simulation models. As consequence, training times of model structures can be reduced and simultaneously the code quality is increased by better documentation.
The algorithm is freely available on GitHub. We kindly invite external users to use and improve the functionalities of ADoCSM by online cooperation using this link:

https://github.com/RWTH-EBC/ADoCSM

Improving the quality of the algorithm and doing systematic troubleshooting, we will apply ADoCSM to the AixLib (Müller, 2016 - 2). Thus, on the one hand, the model quality increases due to documentation improvements. On the other hand, the algorithm is tested and error rate of application will be reduced.

5 Conclusion

Within this work we show that Modelica code can be translated into UML code via the presented tool that is freely available on GitHub:
https://github.com/RWTH-EBC/ADoCSM.
It allows to visualize the model structure and the parameter interdependencies, which is a powerful tool increasing the understanding of Modelica.
Using the recent version, we show for an expansion valve example that it is possible to create UML diagrams considering both important key words and relations of the model.
Our presented expansion valv simulation model for refrigerant circuits is freely available on GitHub:
https://github.com/RWTH-EBC/AixLib/tree/issue590_ExpansionValve
The model considers idealized expansion valve functionality as well as chocked mass flows for different pressure to pressure drop ratios.
All in all, we show functionality of both the modelling approach and of the parsers algorithm. The next steps will consider the investigation of further models to start systematic troubleshooting in order to increase the quality of our algorithm. Therefore, AixLib will be the first large use-case.

References

European Environment Agency (EEA). "Trends and projections in Europe 2016", 2016.

A. Jahnke. Webasto Schulungs-Handbuch: *Kälte-Klima*, 2000.

X. Cao, Z.-Y. Li, L.-L. Shao und C.-L. Zhang. Refrigerant flow through electronic expansion valve: Experiment and neural network modeling. *Applied Thermal Engineering*, 92:210–218, 2016.

D. Müller. Simulationsmodelle für die Heiz- und Raumlufttechnik: Heizflächen. *Vorlesungsvortrag, RWTH Aachen University, Aachen*, 2016.

M. Loeffler, Michaela Huhn, Christoph Richter, Roland Kossel. Modelica CDV A Tool for Visualizing the Structure of Modelica Libraries, *In The 5 International Modelica Conference*, pp. 55-62., 2006.

T. Weilkiens. Sysml: Ein neuer Standard der omg. omg-Kolumne, pages 12–15., 2006.

K. Huchtemann und D. Müller. Advanced simulation methods for heat pump systems. *In The 7 International Modelica Conference*, 798–803. Linköping University Electronic Press, 2009.

T. Storek et al. A modular modelling approach for thermodynamic systems applied to heat pumps. *In 31st ECOS Conference*, 2018.

C. Vering et al. Transiente Modellierung eines Verdichters zum Vergleich von niedrig GWP-Kältemitteln für Kompressionswärmepumpen. In *BauSIM* 2018.

M. Huo. A study in the characteristics of the flow inside a thermostatic expansion valve, Master thesis, University of Illinois at Urbana-Champaign, 2010.

J. R. Simões-Moreira, C. W. Bullard. Pressure drop and flashing mechanisms in refrigerant expansion devices, *Int. J. Refrig.*, 26:840–848, 2003.

A. Davies, T.C. Daniels. Single and two-phase flow of dichlorodifluoromethane, R12 through sharp-edged orifices, *ASHRAE Transactions 79 (Part 1) (1973)* 109-123., 1973.

W. Li. Simplified modeling analysis of mass flow characteristics in electronic expansion valve, *Applied Thermal Engineering*, 54:8-12, 2013.

Drawing UML with PlantUML: Language Reference Guide (Version 1.2018.2)., http://plantuml.com/PlantUML_Language_Reference_Guide.pdf, 2018

D. Müller, et al. AixLib – An open-source Modelica library within the IEA-EBC Annex 60. In *BauSIM*, 2016: 3-9, 2016.

SESSION 6D: AUTOMOTIVE 3

Integration and Analysis of EPAS and Chassis System in FMI-based Co-Simulation

Weitao Chen[1,2] Shenhai Ran[1] Bengt Jacobson[2]

[1]Vehicle Dynamics CAE, Volvo Cars, Sweden,
{weitao.chen,shenhai.ran}@volvocars.com
[2]Vehicle Dynamics, VEAS, Chalmers University of Technology, Sweden,
bengt.jacobson@chalmers.se

Abstract

The vehicle steering characteristics and active functions can be virtually developed with a high-fidelity electric power assisted steering (EPAS) model and a multibody chassis model. The simulation of the EPAS model requires small integration step due to high stiffness and interfacing with the controller. The multibody chassis model is computationally heavy for each integration step due to calculation of large matrices. A mono-simulation based on a single solver is not efficient for this case. Instead a co-simulation (solver coupling) approach has been used to overcome the drawbacks.

In this paper the EPAS system and chassis system are modeled in Dymola and further exported as separate functional mockup units (FMUs) and integrated with the control algorithms in Matlab. A co-simulation based on the explicit parallel calculation scheme (Jacobi scheme) has been used. A huge simulation speed-up has shown the potential and effectiveness of the approach. To understand its accuracy and tolerance, analysis on the numerical error and dynamics of the coupled-system are given.

Keywords: EPAS system, Chassis system, Co-Simulation, FMU

1 Introduction

Modern vehicles involve more electric and functional subsystems with a trend of electrification and automation. Multi-domain subsystems need to be modeled and integrated by co-simulation for a holistic development. This modular approach is quite common because the models might be from multiple sources (e.g., OEM-suppliers relationship) in different tools. Furthermore, it enables each model efficiently solved by a domain-specific numerical method. The approach has been applied in many engineering cases such as an integration of large-scale pantograph-catenary system (Arnold, 2010), a distributed simulation of a 4 cylinder engine (Saidi et al., 2016).

For accurate simulation of vehicle handling, steering and active function tests, a mechanical multibody chassis model and an EPAS model are needed. The chassis model has hundreds of degrees of freedom and its dynamics is relatively slow especially for handling and steering simulation on the flat road. The EPAS model has much faster dynamics because of the lightweight components, friction elements, electric parts and the control algorithms. Its fidelity is critical for the steering feel. As the chassis model and EPAS model differ in terms of dynamics and requirements. A mono-simulation based on a single solver might not be the optimal solution. In this paper, a FMI-based co-simulation has been tested. The coupled-system is constituted by FMUs of a chassis model, an EPAS mechanical model, a S-function for the EPAS electrics and control algorithms.

The modeling work in Dymola is presented in Section 2 and Section 3. The integration based on FMI standard and the co-simulation setup are shown in Section 4. In Section 5 the co-simulation results and analysis on simulation speed and system dynamics are discussed.

2 EPAS System

2.1 EPAS mechanism

The EPAS mechanism comprises mainly a steering wheel and column, a steering rack and an EPAS motor as shown in Figure 1. The steering column is connected to the rack and pinion by a compliant torsion bar. A belt transmission connects the motor and the ball screw which transfers the motor rotation into the rack translation.

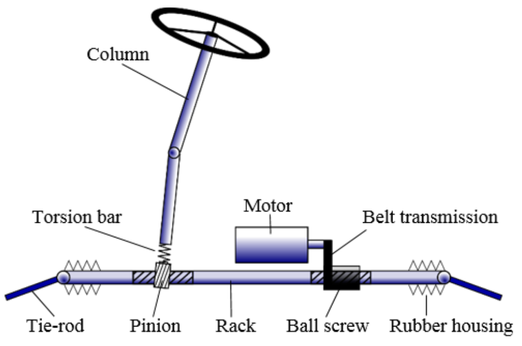


Figure 1. The EPAS system with axle-parallel drive.

The steering rack is articulated to the vehicle chassis through the suspension tie-rods and steers the front wheels. This mechanical chain builds a direct interaction between the driver and the road. The auxiliary electric motor can deliver an assist torque according to the torsion bar deflection and vehicle speed to reduce the steering effort. The steering feel defined by the introduced mechanisms is a key metric in vehicle development and needs to be accurately simulated and evaluated on a driving simulator.

The EPAS mechanism is modeled by 3 degrees of freedom: 2 degrees of rotation for the column and motor, 1 degree of translation for the rack. Different from the Modelon Vehicle Dynamics Library (VDL) steering template, the non-linear effect from the column CV-joint has been neglected and no multibody components have been used for simplicity. Instead, the friction and motor dynamics are very important for EPAS in terms of vehicle steering response, subjective feeling for the driver and the stability of EPAS controller (Harrer and Pfeffer, 2017), more detailed effects have been considered.

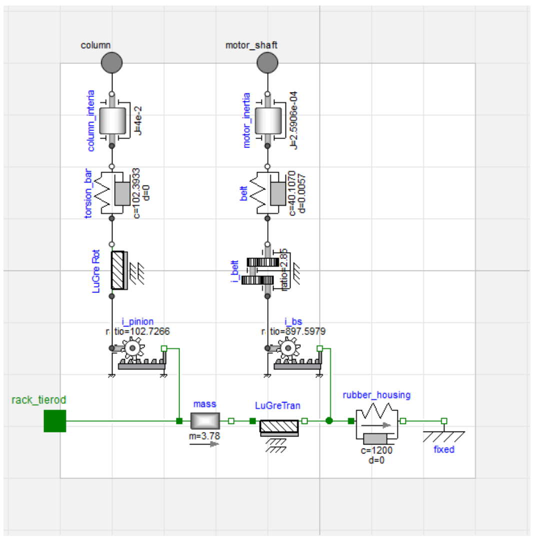


Figure 2. The EPAS mechanism modeled in Dymola.

The model based on basic Modelica mechanical components and detailed friction elements, shown in Figure 2, is created according to the dynamics on the column, the motor and the rack:

$$J_{column}\ddot{\delta}_s = T_s - T_{pinion} - T_{c_{friction}} \tag{1}$$

$$J_{motor}\ddot{\delta}_m = T_{motor} - T_{belt} \tag{2}$$

$$m_{rack}\ddot{x}_R = F_{pinion} + F_{assist} - F_{rod} - F_{r_{friction}} - F_{housing} \tag{3}$$

The states and parameters of the model are given in Table 1. The force F_{pinion} and F_{assist} are calculated from the respective torques and transmission ratios:

$$F_{pinion} = T_{pinion}/i_{pinion} \tag{4}$$

$$F_{assist} = T_{belt}/(i_{belt}i_{bs}) \tag{5}$$

where the torque T_{pinion} and T_{belt} are calculated based on the deflection of the torsion bar and belt.

Table 1. States and parameters of the EPAS model.

Notation	*Definition*
δ_s, δ_m	angle of the steering wheel, motor
x_R	rack displacement
J_{column}, J_{motor}	inertia of the column and wheel, motor
m_{rack}	rack mass
$i_{pinion}, i_{belt}, i_{bs}$	transmission ratios of the rack pinion, the belt, the ball screw
T_s	steering torque
T_{pinion}	torsion bar torque
$T_{c_{friction}}$	column friction torque
T_{motor}	applied torque from the motor
T_{belt}	load torque on the output shaft
F_{pinion}	force transmitted by the rack pinion
F_{assist}	assist force from the ball screw
F_{rod}	tie-rod force along the rack
$F_{r_{friction}}$	friction on the rack
$F_{housing}$	a spring-damper force from the housing

2.2 Friction elements

The mechanical friction is mainly divided into the upstream element $T_{c_{friction}}$ and the downstream element $F_{r_{friction}}$. The friction elements are modeled by the LuGre friction model (Astrom and Canudas de Wit, 2008). In Modelica the standard friction element is implemented by discrete events switching between stuck and slide mode. An appropriate numerical method is needed for this continuous/discrete approach. The LuGre friction model adds the hysteresis effect and it expresses the friction by differential equations:

$$\dot{z} = v - \sigma_0 z/g(v)|v| \tag{6}$$

$$g(v) = F_c + (F_s - F_c)e^{-(v/v_s)^2} \tag{7}$$

$$F_{friction} = \sigma_0 z + \sigma_1 \dot{z} + \sigma_2 v \tag{8}$$

where v is the sliding velocity, z is the internal state. The bristle stiffness σ_0 and micro-damping σ_1 produce a spring-like behavior in small displacements. σ_2 is the viscous friction coefficient. $g(v)$ is a velocity-dependent term relating to the Coulomb friction F_c, the static friction F_s and the Stribeck velocity v_s.

Numerical methods for continuous system can be used to solve this model. However, its dynamics is so stiff that small tolerance value for variable step solver or small time steps for fixed step solver is needed. As a result, the simulation speed gets slow. A detailed implementation and analysis of the LuGre friction model in Modelica has been introduced in (Aberger and Otter, 2002).

The friction model parameters have been partially identified from experiments using a steering system test rig with a steering robot connecting the steering wheel and two linear actuators connecting the rack. Pull-by-torque and pull-by-rack tests (Harrer and Pfeffer, 2017) have been taken with the EPAS controller deactivated. The steering system is excited accordingly either by velocity-controlled steering wheel input or rack input in free load condition. Thanks to the acausal modeling, the recorded data can be conveniently taken as input to the EPAS model. The comparison of the simulation results and the measurement data are given in Figure 3 and Figure 4.

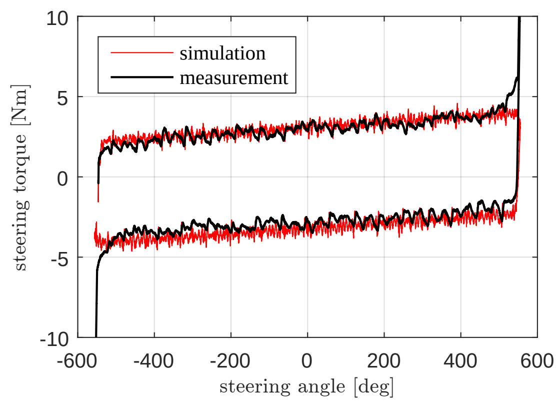


Figure 3. Steering torque in a pull-by-torque test with a steady steering velocity input of 13 deg/s.

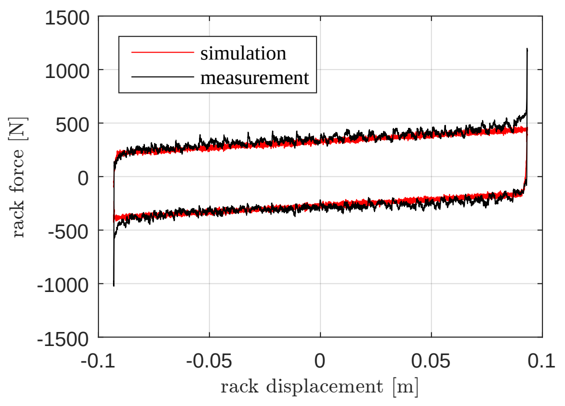


Figure 4. Rack force in a pull-by-rack test with a steady rack speed of 2 mm/s.

2.3 EPAS control

The large inertia, high friction and less damped behavior from the EPAS mechanism is counterbalanced by the basic steering functions involving the inertia compensation, friction compensation, active damping and power-assist. The advanced driving functions like the lane keeping aid (LKA) and Pilot-Assist are added to the motor torque request $T_{request}$ in Figure 5, which is further delivered to the electric motor.

The detailed models of the control algorithm, ECU and electrics are provided from the supplier as black-box S-functions with inputs of vehicle speed $V_{vehicle}$, torsion bar angle δ_{pinion}, motor speed $\dot{\delta}_m$ and the external request from the advanced functions. So that the system needs to run in the Simulink environment with a forward Euler method with 1 ms integration step.

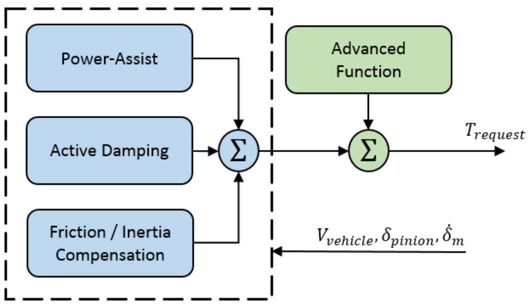

Power-Assist
Active Damping
Friction / Inertia Compensation
Advanced Function
Σ
Σ
$T_{request}$
$V_{vehicle}, \delta_{pinion}, \dot{\delta}_m$

Figure 5. A block diagram of the EPAS control architecture.

3 Chassis Model in Dymola

A chassis model based on the Modelon VDL has been used. It is constituted by the car body, Pacejka tire models and suspensions (Figure 6). To facilitate the computation, the suspension linkages are represented by kinematic tables. The wheel orientation and translation varies according to the wheel jounce and steering input. A validated model of Volvo XC90 has been used in the work.

To integrate the chassis model with the created EPAS model, 1D translation interface is attached to the rack and the original VDL steering model is disconnected as a dummy part. In this way the translation is still relative to the front subframe whose compliance may have a great impact on the steering feel.

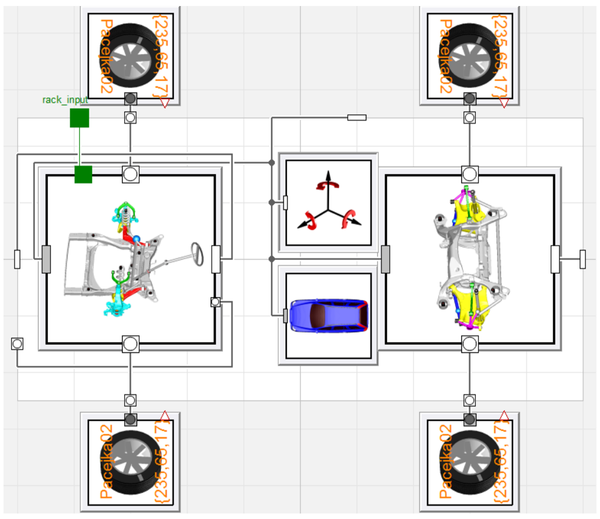


Figure 6. The multibody chassis model in Dymola.

4 Co-Simulation Setup

The EPAS model and chassis model are compiled to separate FMUs embedded with variable step and variable order Dassl solvers. The EPAS FMU, chassis FMU and EPAS control S-function are coupled by specified input-output signals (Figure 7). At the coupling interface the chassis FMU takes the rack velocity $\dot{x}_R$ as input and EPAS model takes the force F_{rod} as input. The decision is based on our analysis from a previous work (Chen et al., 2018), briefly:

- The force variable should be applied towards the heavier and stiffer part for robustness. The EPAS system due to the gear ratio effect is much heavier and stiffer than the lateral dynamics of chassis system.
- The displacement input results an improper dynamic system (more zeros than poles in the transfer function). Thus, the derivation of input variable is needed and this might generate noisy or incorrect results.

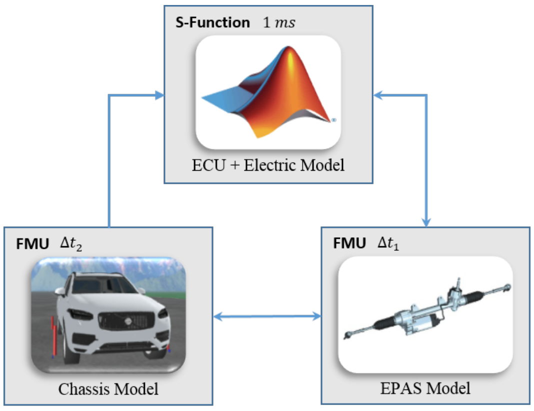


Figure 7. The layout of the co-simulation setup.

The chassis FMU is setup with a communicative step of Δt_2 which is the time size that the local solver updates the input and output. The communicative step of EPAS FMU is Δt_1 with a default value of 1 ms because of the coupling with the controller.

For simplicity, the co-simulation is implemented with an explicit parallel calculation scheme (i.e., during the communicative interval each model is integrated independently and the input is approximated from extrapolation). In this work, a common constant extrapolation (zero-order hold) has been used. Although this calculation scheme suffers from the numerical stability and coupling errors. It is advantageous for less computational burden and easy implementation in practice because no control of computation sequence or iterative process is needed from a master algorithm (Busch, 2012).

5 Co-Simulation Results

The co-simulation have been tested with various scenarios as given in Table 2. For comparison, a mono-simulation reference, denoted by *Ref-1*, is made by compiling the EPAS and chassis model together as a whole FMU with the same solver. The tests are performed on a laptop with 32GB RAM and one Intel Core i7 processor which runs 8 cores at 2.70 GHz.

5.1 Simulation speed-up

A 5 seconds steering maneuver with a sine wave steering torque input has been simulated. From the CPU time of each simulation case (Figure 8), one can see that comparing with *Ref-1* the co-simulation cases are much faster

Table 2. Simulation Cases

Case	*Communicative step*	
Ref-1	$\Delta t_1 = 1ms$	no Δt_2
Ref-2	$\Delta t_1 = 5ms$	no Δt_2
CS-1	$\Delta t_1 = 1ms$	$\Delta t_2 = 1ms$
CS-2	$\Delta t_1 = 1ms$	$\Delta t_2 = 5ms$
CS-3	$\Delta t_1 = 1ms$	$\Delta t_2 = 10ms$
CS-4	$\Delta t_1 = 1ms$	$\Delta t_2 = 15ms$
CS-5	$\Delta t_1 = 1ms$	$\Delta t_2 = 20ms$

especially when Δt_2 gets larger. In mono-simulation case *Ref-1*, the chassis model needs to take a small integration step due to the stiff EPAS model. Instead, in co-simulation each solver can adapt to the local dynamics more efficiently.

In another mono-simulation case *Ref-2* with increased Δt_1, the CPU time reduces a lot as well but the time saving is not so effective as the co-simulation cases with a same or larger Δt_2 setup. It can be observed that a big time saving is from a relaxation of communication with the chassis model.

The co-simulation case *CS-1* does not show an obvious advantage in the simulation speed. Because the adaptability of the local solver is constrained by a very frequent communication of 1 ms. In such a case the speed-up capability of co-simulation cannot be fully used even though the stiff part has been decoupled.

For other co-simulation cases, a further relaxation of Δt_2 does increase the simulation speed but the improvement gets reduced at a larger step. If a rather large Δt_2 has been taken, the two models can be seen as nearly decoupled and calculated independently. Therefore, the simulation time might just depend on the dynamics and solver of each part. In practice, the Δt_2 size setup needs to be compromised considering the stability and coupling error which is discussed in the following section.

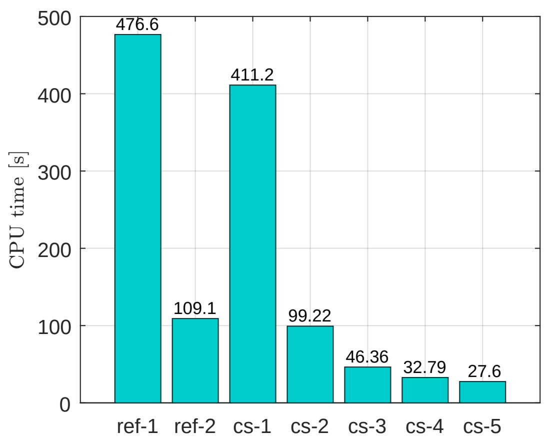


Figure 8. CPU time for a 5 seconds simulation.

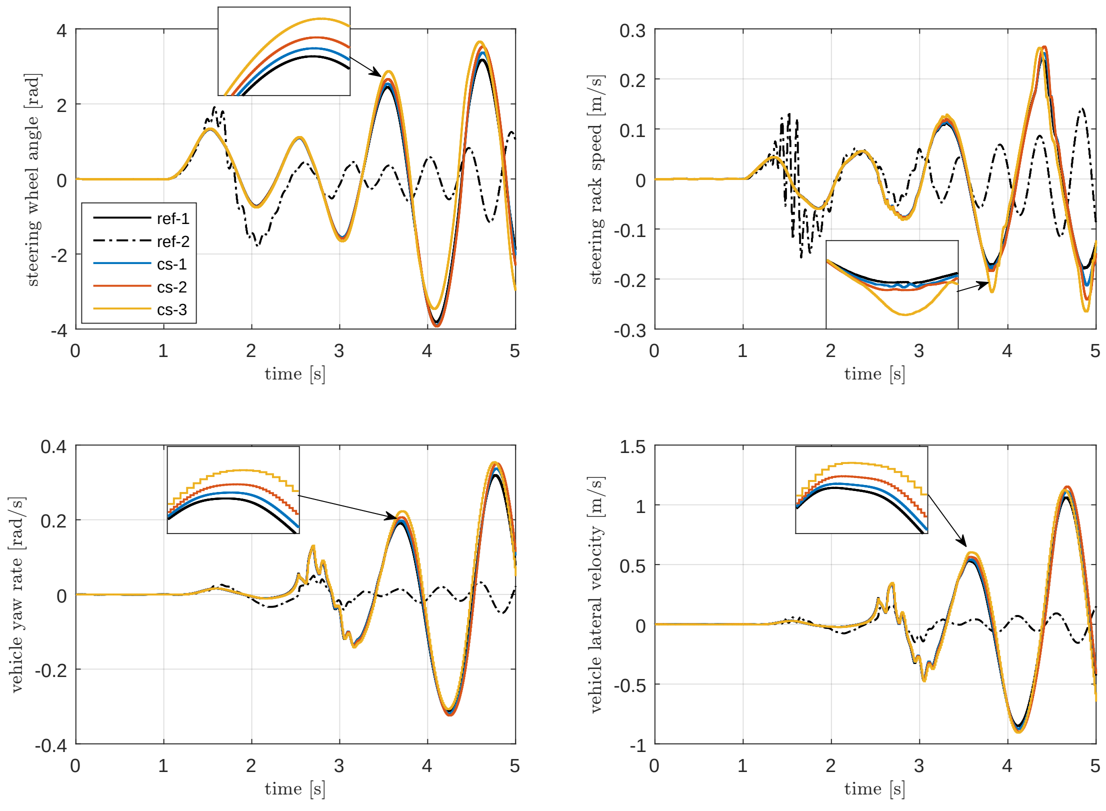


Figure 9. Simulation results of the EPAS system and chassis states.

5.2 Error analysis

Two EPAS system states (steering wheel angle δ_s, rack speed $\dot{x}_R$) and two chassis states (yaw rate, lateral velocity) from the previous simulation tests are plotted in Figure 9. It can be seen that *Ref-2* gives inaccurate and useless results although it can run really fast in the previous analysis. Because the high bandwidth coupling between the EPAS control, electric and the mechanism, the coupling variables are poorly approximated by extrapolation and the simulation error gets quite large.

The co-simulation cases, due to a more robust integration, have shown more stable and consistent results even their simulation speeds are faster. Case *CS-3* in Figure 9 shows larger stepwise signals from the chassis model. The co-simulation results deviate more at the peaks which is very intuitive since the accuracy of extrapolation is worse when the signal changes direction.

The relative global error $\varepsilon_{g,x}$ of selected state x are computed by the normalized root-mean-square error as:

$$\varepsilon_{g,x} = \frac{\sqrt{\sum_{t=0}^{T}\left((x_{cs}(t)-x_{ref}(t)\right)^2/T}}{x_{ref}^{max}-x_{ref}^{min}} \tag{9}$$

where x_{ref}^{max} and x_{ref}^{min} are the maximum and minimum reference state value during simulation time $t \in [0,T]$. The relative global error $\varepsilon_{g,x}$ is plotted in Figure 10. One can see that $\varepsilon_{g,x}$ in case *Ref-2* is clearly the worst and the error increases as the step Δt_2 grows, which limits the relaxation of communication for the simulation speed-up. To reduce the error and enable a further relaxation, some explicit coupling methods from (Khaled et al., 2014) and (Benedikt et al., 2013) can be potentially applied, which is out of the scope of this paper. Thus, only a basic constant extrapolation is presented in this work.

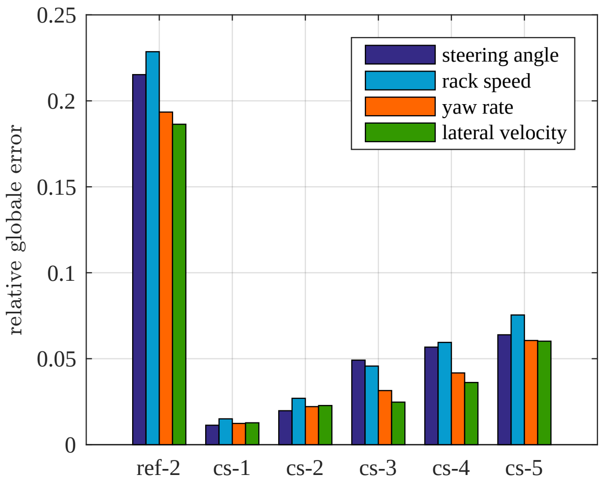


Figure 10. Comparison of the relative global error $\varepsilon_{g,x}$.

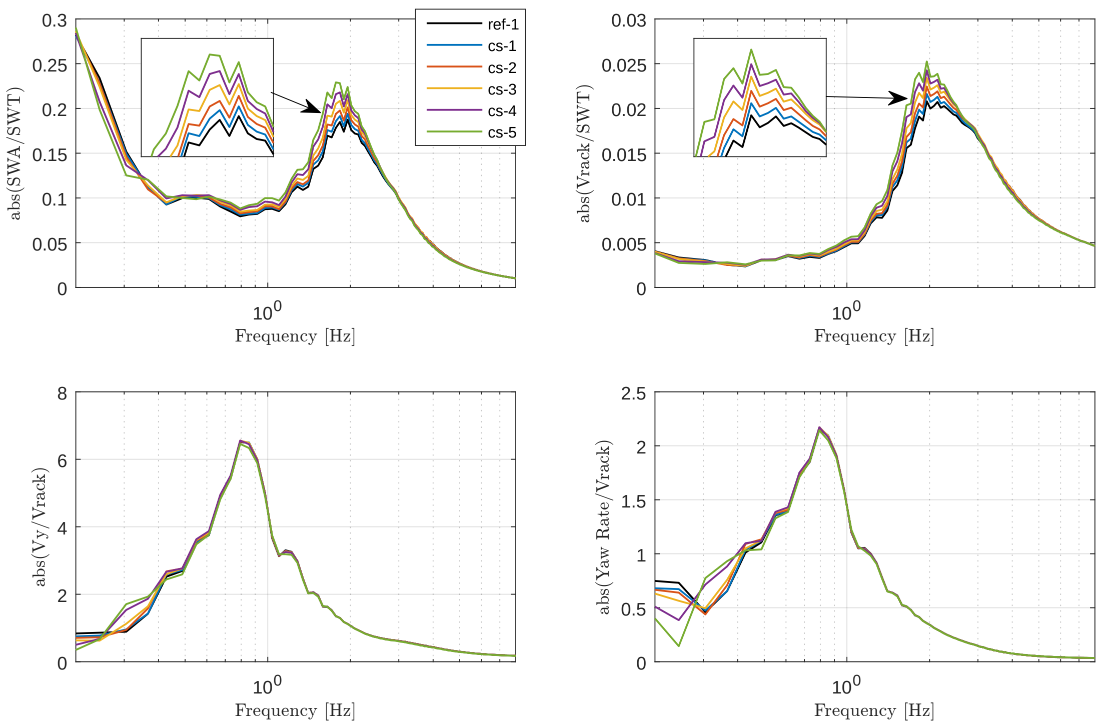


Figure 11. The identified transfer behaviors of EPAS and chassis.

5.3 System dynamics analysis

To investigate the system dynamics in multiple conditions, a frequency domain comparison is more intuitive as a second analysis. In this analysis, a steering torque input from low frequency up to high frequency has been applied. The simulation time is long enough that the system can be excited sufficiently within the frequency range of interest. Two pairs of transfer functions are identified from the simulation results. The first is steering torque (SWT) to steering wheel angle (SWA) and rack speed (Vrack), which is more relevant to steering behavior. The second is from rack speed (Vrack) to chassis lateral speed (Vy) and yaw rate which mainly shows the chassis lateral dynamics.

The magnitudes of the transfer functions are plotted in Figure 11. The steering feedback character (SWA/SWT) and the EPAS transfer behavior (Vrack/SWT) are influenced in a certain range. The deviation gets larger around 1.1 Hz which is close to the chassis yaw eigenfrequency. As Δt_2 increases, more delayed rack force resistant to the rack motion gives an increased steering wheel angle and rack speed.

The chassis transfer behaviors are relatively more consistent to the reference. It might be the reason that the chassis has a slower dynamics and more robust to the coupling effect. The deviation of chassis dynamics occurs mainly below 0.5 Hz and the magnitude of deviation is correlated to the relaxation condition.

Furthermore, the dynamics of the EPAS and chassis system limited the bandwidth of the coupling signals. In the high frequency range of steering input, the only excitation to chassis system has been filtered out and the coupling effect gets minor.

6 Conclusion

In this paper a FMI-based co-simulation of EPAS and vehicle chassis system has been presented. The solver coupling approach is used for mechanical-functional system integration and also for mechanical systems in large time scale. The accelerated simulation speed makes the simulation tool more useful for design optimization and control tuning work. A controllable coupling error without severe numerical instability is induced by the explicit parallel calculation scheme. The approach can also be applied on real-time applications where the simulation speed is crucial. However, the CPU time from the current test is still huge that model order reduction might be needed to make each system real-time capable first.

The approach is quite promising for vehicle chassis and other mechatronic systems (e.g., active suspension, electric propulsion and automated driving system).

Acknowledgement

The authors would like to thank ITEAM project funded by the European Union Horizon 2020 research and innovation program under Marie Sklodowska-Curie Grant Agreement No. 675999.

References

Martin Aberger and Martin Otter. Modeling friction in Modelica with the Lund-Grenoble friction model. *Proceedings of the*

2nd International Modelica Conference, 3:285–294, 2002.

Martin Arnold. Stability of Sequential Modular Time Integration Methods for Coupled Multibody System Models. *Journal of Computational and Nonlinear Dynamics*, 5(3):031003, 2010.

Karl Johan Astrom and Carlos Canudas de Wit. Revisiting the LuGre model; Stick-slip Motion and Rate Dependence. *IEEE Control Systems Magazine*, 6:101–114, 2008.

Martin Benedikt, Daniel Watzenig, Josef Zehetner, and Anton Hofer. NEPCE - A nearly energy-preserving coupling element for weak-coupled problems and co-simulations. *V International Conference on Computational Methods for Coupled Problems in Science and Engineering*, pages 1–12, 2013.

Martin Busch. *Zur effizienten Kopplung von Simulationsprogrammen*. PhD thesis, Kassel University, 2012.

Weitao Chen, Shenhai Ran, and Bengt Jacobson. Design of Interface in Co-simulation for Electric Power Assisted Steering System Development. *Proceedings of the 14th International Symposium on Advanced Vehicle Control (AVEC' 18)*, 2018.

Manfred Harrer and Peter Pfeffer. *Steering Handbook*. 2017.

Abir Ben Khaled, Laurent Duval, Mohamed El Mongi Ben Gaïd, and Daniel Simon. Context-based polynomial extrapolation and slackened synchronization for fast multi-core simulation using fmi. In *International Modelica Conference*, pages 225–234. Linköping University Electronic Press, 2014.

Salah Eddine Saidi, Nicolas Pernet, Yves Sorel, and Abir Ben Khaled. Acceleration of FMU Co-Simulation On Multi-core Architectures. *The First Japanese Modelica Conferences*, (124):106–112, 2016.

Virtual Proving Ground Testing: Deploying Dymola and Modelica to recreate Full Vehicle Proving Ground Testing Procedures

Theodor Ensbury[1] David Briant[2] Mike Dempsey[2]

[1]Claytex USA, Inc., Ann Arbor, Michigan, USA, theodor.ensbury@claytex.com

[2]Claytex Services Ltd. Edmund House, Rugby Road, Leamington Spa, CV32 6EL, UK

{david.briant, mike.dempsey}@claytex.com

Abstract

Physical testing of new automobiles is often a lengthy and expensive process. Virtual testing of vehicles offers a much more flexible, efficient solution to testing new vehicles, whilst also providing a more consistent and easier to mange testing environment. This paper presents how the VeSyMA suite of libraries contains the necessary features required to recreate 2 typical physical proving ground tests in the virtual world. Key new features added to the VeSyMA suite to enable this are presented, namely: a new method of defining the proving ground road model using GPS and body accelerometer data, a new driver model capable of conducting a series of scheduled driving tasks (mimicking a human test driver) and new tyre contact models more suitable to typical proving ground rough roads.

Keywords: Virtual testing, proving ground, vehicle testing

1 Introduction

Development of a robust, high quality product naturally relies upon an extensive program of testing to identify anything which negatively impacts the customer experience. Designed to replicate all elements within a product's usage envelope, such a program seeks, and enables the manufacturer to find, potential issues and validate effective solutions. A program like this can take many forms, including from monitoring actual usage within the field during pre-production; accelerated lifecycle simulation in both deployed or simulated settings; various specific specialist use cases and design limit exceeding, but to name a few.

Automotive product development is no different, complicated by a long, arduous product lifecycle and adherence to various regulatory constraints all over the world. Added complexity arises from the safety-critical nature of an automobile, which further enhances the need for a wide scale testing program to be conducted by the Original Equipment Manufacturer (OEM).

This paper will establish how the Vehicle Systems Modelling and Analysis (VeSyMA) suite of libraries can be deployed to recreate typical real-world testing scenarios in the virtually using Dymola and Modelica, focusing on offline, non Driver-in-the-Loop (DiL) simulation.

1.1 Standard Automobile testing

To prepare an automobile for market, testing at various stages of the automobile development process is conducted by an OEM. Some testing will be to fulfil regulatory requirements, with other testing unique to the OEM, often focused on specific areas of interest which the OEM prioritizes. This can include proof-of-concept evaluation. Typically, an OEM test program involves a high amount of physical mileage conducted in prototype, pre-series production and series production vehicles. Various scenarios can be chosen depending on the need, which can involve the vehicle having to be driven extensive amounts of distance or transported around the world to undertake environmental testing. Some testing can be done in laboratory conditions, but non-static full vehicle testing often requires the vehicle to be transported to another part of the world.

This means full vehicle testing can be an expensive and time-consuming endeavour, due not only to the costs associated with producing prototype or pre-series vehicles, but also to the need to transport the vehicles around the globe for testing. Resource time allocation requirements of testing programs can thus impact the product development cycle, often dictating at what point certain test programs can be conducted, rather than deployment at the optimum time during the development cycle.

1.2 Virtual testing advantages

Harnessing simulation tools to test virtually can endow multiple benefits. Elimination of the need to produce the same number of prototype and pre-series vehicles is an obvious area cost and time saving to the OEM, in addition to being more environmentally sound. Reducing the amount of travel associated with testing programs further reduces cost, time and environmental expenditure.

Beyond this, virtual testing is also an inherently more flexible process than physical testing. Proof-of-concept testing can be done at a much earlier stage of the design process, whilst durability evaluation can begin to be conducted at earlier stages of the design cycle, as immature designs can be evaluated in the same manner as mature designs at the full vehicle level. This leads to the possibility of shortened development schedules, as design issues can be identified earlier and eliminated sooner, with less resources going into failed design elements. Virtual testing can also aid the efficiency and optimisation of components; more inclusive, targeted and varied testing of components to cover more use cases can be conducted during the component development stage.

Virtual testing also offers the possibility of a more consistent testing environment, with greater control of various factors which effect results. Ambient temperature, wind speed and wind direction can all effect physical testing; they can be controlled far more tightly in the virtual environment than in the real world. Human variance when driving the vehicle can also eliminated in the virtual testing environment, as driver models can be used in their place if desired. All this leads to testing conditions being continually repeatable in the virtual world, something unachievable in the physical realm.

1.3 Virtual testing requirements

To faithfully recreate a physical real-world testing scenario virtually, the modelling requirements can be broken down into 3 individual categories:

- Vehicle Model
- Road Model
- Driver Model

The vehicle model is termed as the model tasked with recreating the physical vehicle platform's behaviour, including the tyres. A road model accounts for the ground position at the wheel contact point, with a Driver model simulating the control actions applied to the vehicle by the human driver. Complex aerodynamic and thermal interaction with the atmosphere is neglected in this example to reduce the performance load of the simulation.

Logically, a vehicle model deployed in such an application is required to recreate the vehicle behaviour to the satisfaction, and confidence, of the user. In addition, to be able to fully realize the flexibility virtual testing can provide, it follows that it should be built in a modular way with scalable detail. This enables the user to be able to tailor the complexity of the simulation to their desire, effectively isolating specific use cases or effects. Dymola and Modelica's object-oriented nature makes them the obvious choice for this task.

Requirements for a road model are slightly simpler. It should represent the topography of the road surface the vehicle traverses in enough detail to induce the same vehicle response as occurs in the real world. Local and global topical phenomena (rain, dust, dirt or tyre rubber accumulation), which effect the road surface and influence vehicle response behaviour can be included by varying the road mu value (Segers, 2014). Similarly, the driver model utilized must be capable of recreating the same control inputs as conducted in the real-world test.

2 Vehicle Modelling

All vehicle, vehicle system models and tyre models described in this chapter form part of the VeSyMA suite of libraries from Claytex.

2.1 VeSyMA Approach to vehicle modelling

As outlined in previously (Hammond-Scott and Dempsey, 2018), the VeSyMA suite of libraries from Claytex provides a complete vehicle simulation solution, scalable to specific customer needs and desires. Comprised of the VeSyMA library itself with multiple subject specific expansion libraries, each subject library can either be used in isolation or with extension libraries at both the system model and full vehicle level. Underpinning the VeSyMA suite is the Modelica Standard Library (MSL) and the Vehicle Interfaces Library; the latter defining the standard vehicle level model interfaces, thus enabling each of the VeSyMA subject libraries to interact on a modular level. This is shown in Figure 1.

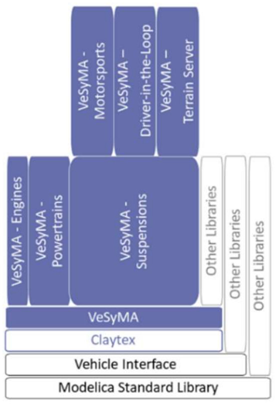


Figure 1. Schematic overview of the VeSyMA suite of simulation libraries.

Acting as a parent library, the VeSyMA library itself defines model templates as well as having the capabilities for straight-line drive cycle testing. Examples of subject specific libraries available for use with the VeSyMA suite include VeSyMA – Suspensions, for detailed analysis of automotive vehicle dynamics including 3D multibody suspension, high-fidelity tyre models, road and driver models; VeSyMA – Engines, a library devoted to high-fidelity internal combustion engine models; VeSyMA – Powertrains, a library of full 3D multibody driveline and gear train models; VeSyMA – Driver-in-the-Loop, which enables the deployment of vehicle models developed using the VeSyMA – Suspensions (or VeSyMA – Motorsports)

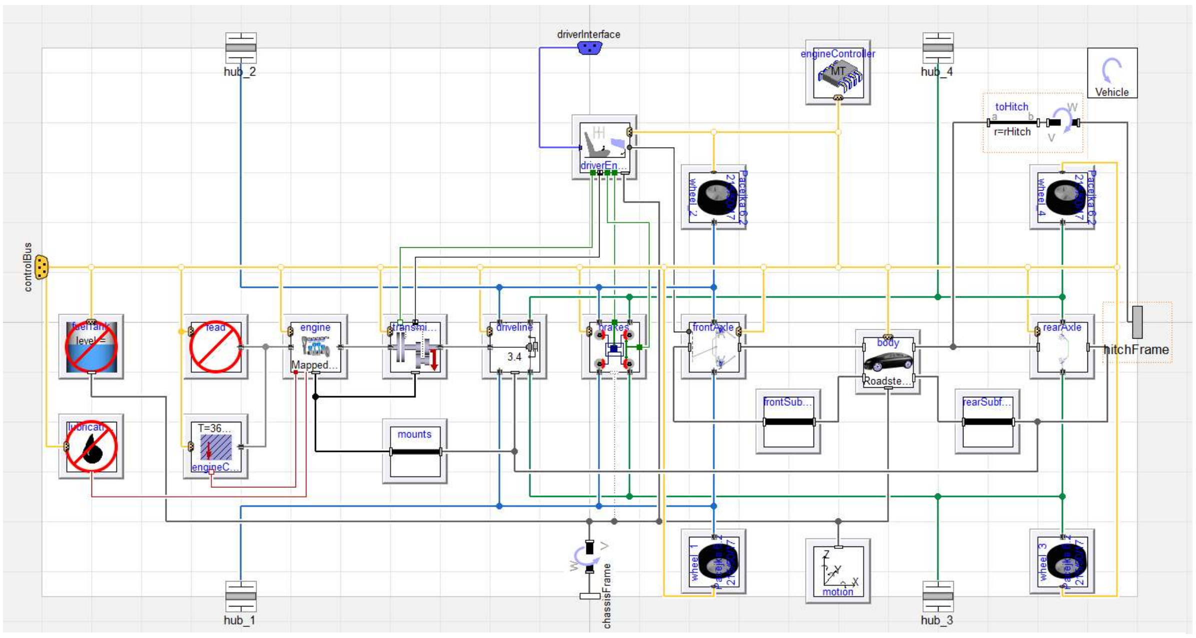


Figure 2. Schematic overview of the RoadsterSportMTRT vehicle model

libraries in DiL real time virtual environments, such as rFPro.

The concept of model reuse is a core principle of the VeSyMA suite. An efficient modelling philosophy, this enables the VeSyMA suite to be a fully scalable, modular solution, where the same models can be deployed in simulations ranging from detailed desktop scenarios, to DiL applications and quick drive cycle evaluations. This enables the user the flexibility to fully utilize several aspects of simulation-based development and investigation over physical testing.

2.2 Vehicle model

To demonstrate the capabilities of virtual testing, an example vehicle model was taken from the VeSyMA – Suspensions library, namely the RoadsterSportMTRT vehicle model as shown in Figure 2. Being a complete multibody vehicle model, the body is free to move within the virtual environment in all 6 degrees (lateral, longitudinal, heave; roll, pitch and yaw). All subsystem models within this model can be found in either the VeSyMA library or the VeSyMA-Suspensions library.

With a Euro NCAP roadster sport class segment vehicle platform, the RoadsterSportMTRT features a traditional rear-wheel drive layout (RWD) with the drive torque only sent to the rear wheels through an ideal manual transmission model and 1D rotational driveline. No electronic stability modules were included in the vehicle. Lubrication or front engine ancillary drive (FEAD) models were also omitted. A mapped engine model simulated an example internal combustion engine (ICE). More detailed powertrain models could be included by deploying the VeSyMA – Engines or VeSyMA – Powertrains libraries. Such models are not included as part of the VeSyMA – Suspensions library, thus not featured in this example vehicle model.

Double wishbone suspension was featured at both the front and rear of the vehicle; innovative “aggregate” joints capable of DiL running were utilized. Compliances within the suspension mountings were omitted, as were flexures within the suspension members themselves. No suspension bushing models were used. As the suspension linkages were full multibody models, each link had its own specific mass and inertia properties. Ride spring and damping elements were modelled using translational models, with full multibody anti-roll bar models deployed at both the front and rear axle. Aerodynamic effects were considered, using a simple lift, side force and drag model. All four wheels featured independent brake models, utilizing a controlled elasto-plastic model (Dankowicz, 1999) to model the friction between the disc and the pad. Each brake model featured individual rotating mass models to account for the gyroscopic effects of rotation, along with a separate, static calliper mass model.

The variable step Radau IIa – order 5 stiff solver was used with a tolerance of 0.0001 to simulate the vehicle model during test scenarios. 116 continuous time states were selected when the vehicle model was translated. As this is an generic example vehicle model, it is a qualitative representation of a vehicle using approximate data. Due to this, it enables qualitative judgements to be made regarding it (and surrounding simulation tools), rather than specific quantitative judgements.

2.3 Tyre and Contact Modelling

A key component of any vehicle simulation is the tyre model used. In the example vehicle a fully combined lateral and longitudinal slip Pacejka tyre model was used, corresponding to Magic Formula (MF) 6.2. Behaviour and response were thus non-linear. Included is asymmetric tyre behaviour due conicity and ply-steer with a kelvin spring damper modelling the vertical dynamics of the tyre (Pacejka, 2012). Contact frame information is obtained by interfacing with the road model to gain position, road normal and friction coefficients. These inputs are then used by the tyre model to produce forces and torques that are applied to the suspension and powertrain models.

Typical implementations of the Pacejka model feature only a single point of contact; this limits the model's ability to handle terrain which varies with high frequency, where surface/tyre contact can occur in multiple places on a single tyre.

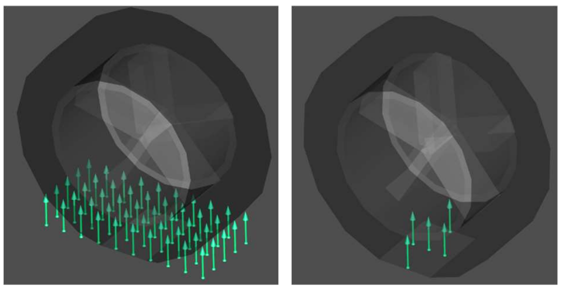


Figure 3. Comparison of multi-point tyre contact models. On the left is an example of the grid contact point model, with the 5 point model on the right.

Tyre model fidelity can be enhanced by deploying a multi-point contact model. Such models interface with a higher frequency or undulating road models, reducing the multiple contact point data down to a single point input for the tyre force model. One outcome is a natural smoothing effect of high frequency/high amplitude road topographical variations. There are 2 implementations of weighted multi-point contact models available in the VeSyMA – Suspensions library, shown in Figure 3; a 5-point diamond and a grid pattern. The 5-point contact model features 4 auxiliary outer points arranged in a diamond around a central contact point; spacing of the outer points relative to a central point is controlled by the user by specifying the dimensions of the contact patch. The cross product of the vectors between the central point and the outer points are used to generate 4 normal vectors. And a weighted average of the position of these points is used to produce a contact position. In contrast, the grid pattern contact model sees the contact patch discretized into a rectangular grid of points, aligned with the heading of the wheel hub. The weight of each point is dependent on the penetration depth of the point. The user can define the dimensions and resolution of the grid; each row and each column is equidistant from the next, with the distance ahead being important when considering step changes.

Moving further, the VeSyMA – Suspensions library also supports use of the FTire (Flexible Structure Tyre Model) model, a physics-based tyre model developed by Cosin scientific software. Dymola support is provided by the FTire library, which provides an interface to the FTire external program. The VeSyMA – Suspensions library builds on this interface to integrate into the vehicle models through the road am wheel models only. Unlike mathematical based tyre models, FTire is a fully physical model, which "explains complex tyre phenomena on a strictly mechanical, tribological, and thermodynamic basis". Applicable in short-wave-length, high-frequency studies, such as for vehicle noise, vibration and harshness (NVH), ride comfort studies, load prediction and durability testing, FTire details "temperature, wear, air vibration, (and) rim flexibility" with both rigid and non-rigid road models (Cosin scientific software, 2018).

FTire models require license for use from Cosin scientific software, and thus were not deployed in the use cases presented in this paper.

3 Road Modelling

A road model (also termed a ground model), is the model responsible for defining the position of the road surface and commonly includes the driving line information to follow, such as position and velocity targets. Within the VeSyMA libraries there are several different road models, all using the methodology of defining road data relative to the distance along the centreline or driving line, perpendicular offset and road normal height (s, w and z). The inner definition of the road at the top level of the experiment is used in conjunction with outer road blocks that can read data from the road using those s, w and z inputs.

The height of the surface of the road is defined by two elements of the road; the underlying topography and the roughness, which is overlaid over the top. Topography of the road defines its general profile (low frequency undulations), which are smooth and considered the general position of the surface. Roughness is a deviation around the topographical position along the road normal to apply higher frequency roughness (bumps) and surface height change events such as kerbs or potholes. Such separation of the two improves efficiency and allows for more accurate measurement relative to the road. For example, during vehicle pitch or roll, the roughness does not influence the measurements taken as the data is measured relative to the smooth underlying surface, improving the accuracy and speed of simulation.

There are several other road types that are available to use in the VeSyMA libraries, which allow for alternative methodology while maintaining the same interface, using the road blocks. But FTire uses its own road model which defines the road models and is passes data to the wheels within vehicle model.

3.1 Defining road topography

Two approaches can be taken to defining the road model topography; either a real physical road/venue can be recreated, or a fictional road can be used. Both approaches have value depending upon the type of study to be undertaken. For instance, a fictional road can be a road which is impossible to build in real life, like a perfectly flat plane to evaluate spring and damper settings without interference from body forces generated by road topography or roughness. Within the VeSyMA and VeSyMA – Suspensions libraries, there are several functions which enable a road model to be generated from inputted data points. These enable the user to build their own road models of any topography (3D geometry, width, banking angle, friction coefficient and roughness) they wish. In addition, road model building from curvature is also supported, with several functions supporting the development of various specialist roads, such as constant radius circles, slalom courses and figure of eight roads.

It is however often desirable to recreate a physical road for use in virtual testing. One method of doing this is to have the road (or circuit) laser scanned using LiDAR to generate a 3D point cloud of the surface. Such methods are highly accurate; VeSyMA – Terrain Server provides the tool chain required to interface a VeSyMA suite vehicle model with the full laser scanned driving environments found in rFpro. However, getting a road scanned in such a way is a prohibitively cost intensive endeavour, which may not be suitable. Therefore, the VeSyMA – Suspensions library features a specialist road building function called “RoadFromLoggedData” which builds a road model from GPS and body accelerometer data. This method of building a road model is useful in applications where it is not possible to or simple to laser scan a road (too costly, too time consuming, or the user does not have the permission to undertake such an endeavour) yet the user desires a road model. Furthermore, this method only requires 6 channels of data from 2 sensors, which can often be found on data logging equipment as simple as phone.

As most commercially available GPS units use the Universal Transverse Mercator (UTM) format, the GPS data can easily be converted into x, y and z coordinates by reversing the Gauss-Krüger projection (Kawase, 2013). Based upon dividing the Earth’s surface into 60 north/south zones each measuring 6° longitude, each zone is an effective projection of the curved surface of the Earth onto a flat plane (U.S. Geological Survey, 1997). Transformation parameters utilized are those found in the World Geodetic System 1984 (WGS 84) spheroid model of the Earth (National Geospatial-Intelligence Agency, 2005). The only data the user is required to have beyond the GPS sensor output is knowledge of whether the road to be modelled is in the northern or southern hemisphere (a sign change occurs depending on the hemisphere inhabited in the UTM system) and the UTM zone the data was logged in, which can be looked up on a standard map (Kawase, 2013). This is a valid method of detailing road topography, as the Gauss-Krüger projection has been shown to be accurate to “a few nanometers” (Karney, 2011).

Body accelerometer data (x, y and z axes) is utilized to determine the banking angle of the road, as experienced by the vehicle. For the function to give accurate results, it is advised that the accelerometer data is smoothed sufficiently to present the low frequency fluctuations of data clearly, in addition to being recorded on a vehicle which experiences low roll and pitch angles. Correction for vertical accelerations created by extremely high aerodynamic forces (such as downforce on a prototype racing car) is advised. The following equation is used to determine the banking angle as experienced by the vehicle:

$$\theta = \operatorname{atan}(\frac{G_{Vertical} - 1}{G_{Lateral}})$$

This method of deriving the banking angle is considered the standard method when converting logged vehicle data and has been used extensively in motorsport applications (Segers, 2014).

As the aim of the function is to describe the low frequency undulations in road topography, it is advised that the data used to build a road using the “BuildRoadFromLoggedData” function is filtered prior to usage with an appropriate filter and pass band frequency, as well as being logged a non-excessive rate (around 20Hz).

3.2 Defining road roughness

As explained earlier in this paper, road roughness is effectively “overlaid” onto the existing road topography, meaning roughness as defined is the deviation of z height relative to the underlying road topographical data. Therefore, this means that the standard VeSyMA road models require the roughness to be defined independently of the road topography. There are two main ways to do this: implement a standardised friction grade or define a specific roughness table, with data generated from another source.

Each VeSyMA suite standard road model features the ability to deploy an ISO 8608 standard roughness grade; various grades are available, designed to be characteristic of paved road surface quality in different states of wear (Agostinacchio et al, 2013). Available options are:

- A (Very Good)
- B (Good)

- C (Medium)
- D (Bad)
- E (Very Bad)

Defined as a 2D implementation of roughness, each roughness profile is comprised of various excitation frequencies to generate a singular roughness value. Various differing grades provide different amplitudes, whilst the user can determine the range of frequencies which are used to define the roughness in each case. Roughness values generated this way are valid along the heading direction of the road, therefore care should be taken if the vehicle yaws at high angles relative to the road heading direction. Despite this, this method provides a useful method of including generalised roughness values when specific data does not exist.

A user can also specify a roughness table directly into the road when the road model is built using one of the in-built VeSyMA road building functions. Being a 2D table, this enables the user to specify the roughness not only as a function of the road centreline, but also at various road width points perpendicular to the road centreline.

3.3 Road feature modelling

Actual physical roads do not solely comprise the driving surface, nor do the vehicle tyres only contact the surface immediately within the contact patch; therefore, there needs to be consideration of road features within the road models.

Such features typically comprise of large protrusions above the road surface, such as kerbs, speed bumps and irregularly large bumps. Features such as these are available to deployed in a cylindrical form, as additions to the road surface. The user can specify their location on the road, as well as their radius, which enables control of their height above the road surface. The multi-point tyre contact models are recommended for use with road kerbs. Specifying a negative height of such a road feature can be done to model depressions in the road surface, such as pot holes.

3.4 Advanced road formats (VeSyMA - Terrain server, Open CRG, Open Drive)

Moving beyond road models and features which form part of the standard VeSyMA roads, various advanced options are available for the user to deploy in a VeSyMA vehicle simulation.

Despite the increased cost of production, point cloud data remains a popular method of defining a road in acute detail. Road models made of point cloud data are very high fidelity, with even the smallest details of the surface, surroundings and topography represented. Such formats are popular in DiL applications, such as rFpro, where they are used to recreate various real-world environments, alongside standard rFpro road model definitions. The VeSyMA – Terrain server library provides an interface between the rFpro Terrain Server and the offline Dymola simulation, meaning rFpro point cloud road surfaces (and standard format) can be deployed in VeSyMA vehicle experiments. This enables consistency between the road model used within the DiL and offline simulations. It also allows offline simulations to use high fidelity road models, which exactly match measured road surfaces for improved accuracy of evaluation experiments. Support for all tyre contact methods used in rFpro is also included.

OpenCRG (Curved Regular Grid) format road files are also supported by a specialist road model within VeSyMA, termed the OpenCRGRoad. Growing out of an internal Daimler AG road format in 2008, OpenCRG is a widely used file format for the detailing road surfaces (OpenCRG, 2018). Characterised by a curved reference line with a regular elevation grid, the Open CRG format enables the road topographical data to be accurately and efficiently stored. It defines a specific file structure for this purpose, therefore a specialist road model is used in the VeSyMA suite to support use. (Rauh et al, 2008).

Complimentary to the Open CRG format is OpenDRIVE, a file format for the description of road networks. Another open source format, it is developed and maintained by VIRES Simulationstechnologie GmbH and designed to enable the user to select routes from larger Open CRG databases (OpenDRIVE, 2018). The VeSyMA – Suspensions library features a tool which creates road files in the VeSyMA road format using a GUI that uses the OpenDRIVE road file to define the route and add velocity targets.

4 Driver Modelling

To be able to perform vehicle manoeuvres using conventional vehicle controls, driver models output all demands matching those found in a normal vehicle. Such models can be considered as belonging to one of two categories: Open Loop or Closed Loop, depending on control type. Open Loop drivers do not use inputs from the vehicle or road to produce control demands; they have predetermined outputs, normally dependent on time. Closed Loop drivers use inputs from the vehicle and compare them to targets, either gathered from the road or internally from the driver, producing thus a control demand to achieve those targets.

The interface from the driver to the vehicle uses the same variable naming that is found in the vehicleInterfaces library, which is used as a base, allowing for improved interfacing with other vehicle models.

4.1 Closed Loop Driver Methodology

The Closed Loop driver models have a methodology laid out by the template with replaceable components allowing for adjustable methodology. The methodology follows in following order and is broken into separate sub-models within the driver, as shown in Figure 4:

1. Senses: Gathers information from vehicle and interprets inputs into usable variables.
2. Planning: Uses interpreted variables to compare against targets.
3. Controllers: Uses the comparisons in the planning to produce a Normalised demand.

This demand is then outputted to the vehicle, where it is received by a driver environment that converts it from normalised demands to real inputs. These inputs, used by the vehicle components, match real measurable values, such as steering wheel angle or pedal position.

The Senses block, alongside routing variables from the vehicle to the driver, also converts position into road and driving coordinates, such that they then can interact with the road to produce target variables.

The planning block is broken down into 3 separate planning areas: longitudinal, lateral and gearbox planning. Each planning block, while predominantly independent, can also be fed by other planning blocks. These blocks take the inputs from either the road and then output achievable targets, such as velocity, that the driver achieves.

The controller blocks, Longitudinal, Lateral and Gearbox, use the targets from the planning block and compare them to the output of the senses block to provide demands.

Both the lateral and longitudinal controllers use modified PID based controllers to produce demands with multiple, separate "Look Ahead" points from the planning block that gather target data ahead of the current vehicle position. For Longitudinal planning that could be target speed, for lateral planning it is driving line position and heading.

To control the driver characteristics, all control variables have been propagated to the driverProfile, a replaceable record, that allows large changes in driver response to be changed with one redeclaration.

Given that there are several different types of vehicle control that a driver would encounter, a controlType selection regarding the gearbox type defines which elements of the driver are active and the types of outputs. This can reference the same variable that exists in the vehicle which further automates the linking of driver to vehicle, requiring less modification of the driver to change between vehicles.

4.2 Test Drivers

The Closed Loop driver, described above, uses a consistent method of target and demand generation. However, for a test such a durability cycle, which can contain both speed, gear and other Closed Loop demands and Open Loop, control specific output demands in the same test, this is not enough. An extension to the pre-existing Closed Loop driver is required to include this flexibility. Said extension is required to maintain the same ability to follow Closed Loop targets, as well as specified open and closed demands in any order; all with the ability to return to full Closed Loop mode smoothly.

There are two versions of the test driver extension, the first with a predefined step initiated by a simulation time or driving line position. A Closed Loop demand is swapped for the demand provided by a replaceable source. This is predominantly used for simple tests, where there is only one change in demand. An example of this is the ISO double lane change where the vehicle is to coast through the manoeuvre after arriving at a constant speed.

The second version of the test driver is the test sequence driver, where both the target outputs of the planning block and the demand output from the driver can be controlled. With an expandable number of steps, defined by a record vector, each step contains an activation condition and driver action or target to be followed. Resulting actions or targets must either be output directly as a demand or as a target for the controllers to follow. This, alongside the controlType definition, allows the same driver and test sequence to be built and run with any vehicle. Target or control output is then defined by output from a choice of source.

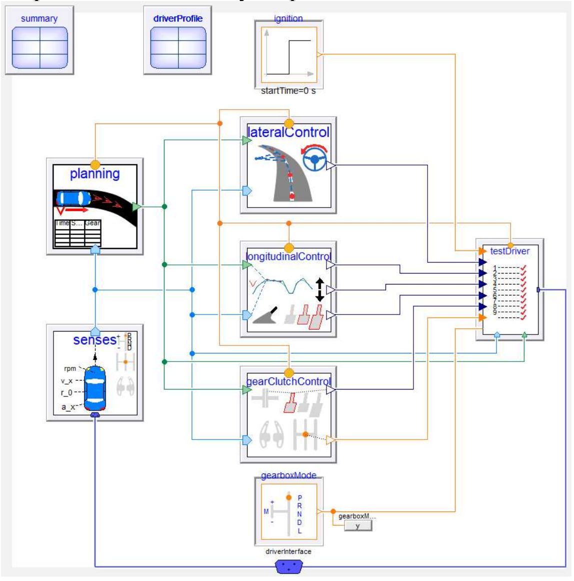


Figure 4. Test Sequence Driver Layout

4.3 Test Sequence methodology and generation

The test sequence driver, as mentioned above, uses an expandable record vector with each step defining the condition and output to be used for each step. The activation condition is to be selected from a list of available conditions, which include position, time, velocity and yaw rate. With each condition is the comparison type, either greater or less than or equal to. The driver action, to reduce overheads and complexity, is primarily broken down into individual signal-based demand changes. There are also blanket changes to the controls, such as reverting all controls to closed loop. If the driver action has associated targets to define then these are then chosen from a list. If the target, such as a target velocity, is defined then the same closed loop control is used, with the outputs from the planning block being overridden. This therefore retains all the same characteristics as a standard closed loop driver.

During a simulation the driver proceeds through the sequence with the step condition only able to be activated once the previous step is active. This reduces overheads for the driver, with only calculations being active when the current step or awaiting activation.

5 Example test scenarios

It has been stated that "Proving grounds are an extremely efficient means of qualifying the durability of vehicle components. They accelerate damage accumulation rates so failures are detectable in a very short period of time" (Halfpenny and Pompetzski, 2011). Therefore, to show the suitability of the VeSyMA suite to simulate proving ground tests, two examples of vehicle durability studies are presented. Specific areas investigated in this paper are high speed laps around a banked circuit and chassis twist/warp bumps, to demonstrate the new additions to the VeSyMA library. High speed lapping of a banked circuit is a standard method of accelerating high-mileage for durability testing, with chassis twist/warp bumps a typical test structural durability, often found at physical proving grounds (Berg, 2015). Each example tests different areas of the vehicle and requires different directions for the driver and vehicle setup. As explained in section 2, due to the vehicle model being a generic example,

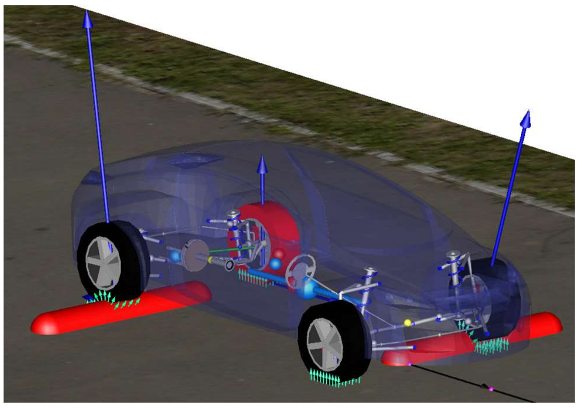


Figure 5. Vehicle rolling over kerbs with wheel loads

results presented here are intended for qualitative analysis rather than specific quantitive analysis.

5.1 Chassis Twist Bumps

In the chassis twist experiment there are bumps spaced at roughly a wheelbase length apart on either side of the vehicle. This applies most of the weight of the vehicle onto the opposite corners of vehicles, applying a large amount of torque down the length of the chassis.

For this type of experiment there are several methods of generating road models, as described above. This experiment utilises the VeSyMA – Suspensions road with inbuilt roughness and cylindrical kerbs. The kerbs are spaced equidistantly, roughly a wheelbase length apart on alternating sides of the vehicle, as shown in Figure 5. The roughness is considered as a medium road surface, using the Grade C roughness. The contact model used for each of the tyres was the grid contact with 40 contact points each. This allows for the Pacejka

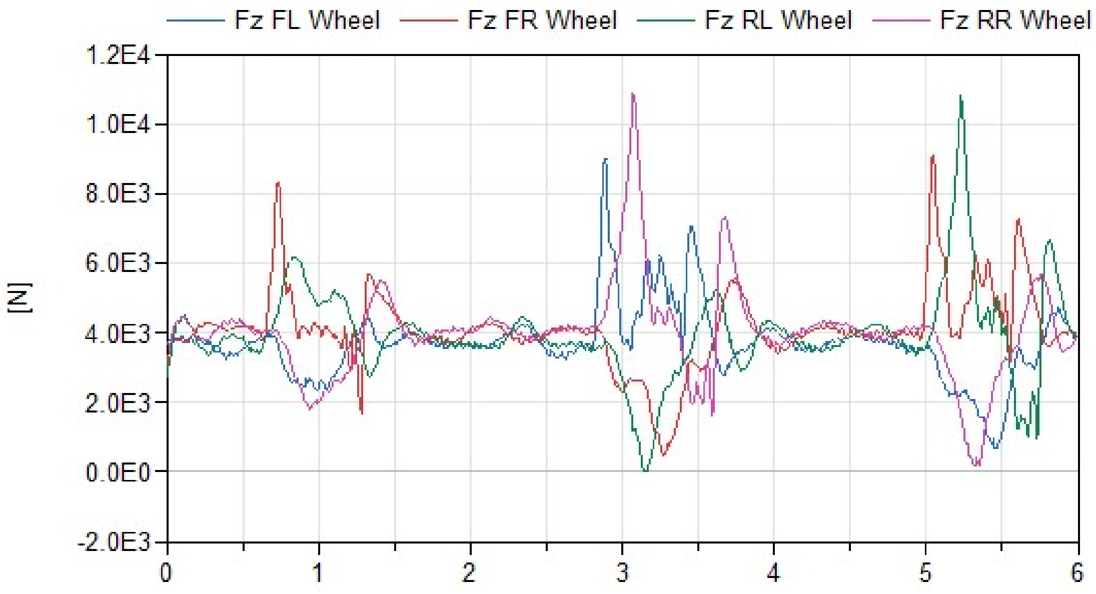


Figure 6. Vertical Wheel Loads against time (seconds) during the chassis twist experiment

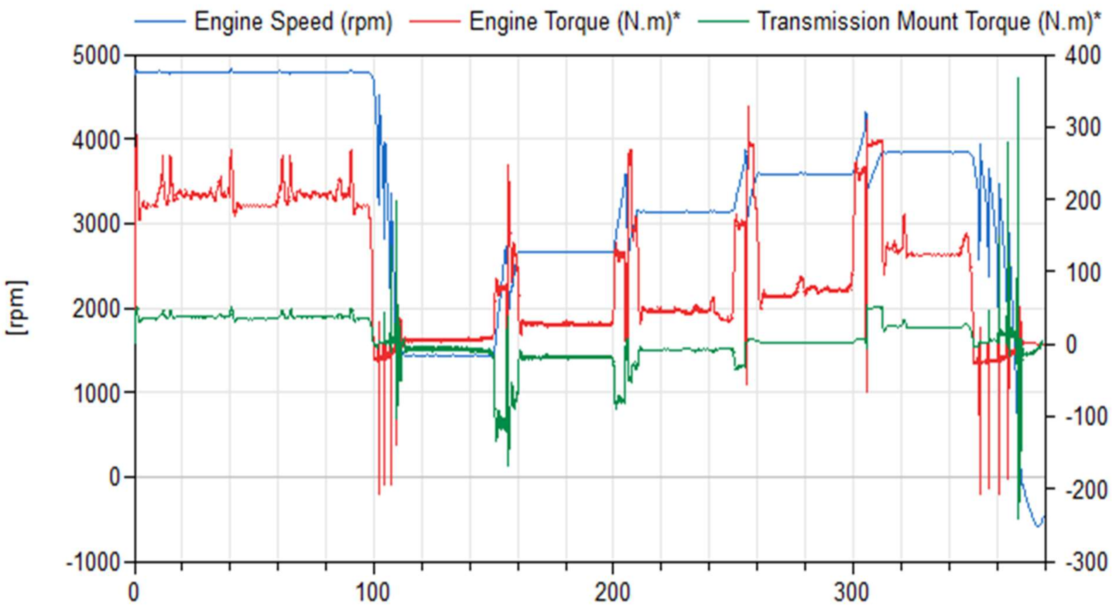


Figure 7. High speed engine speed and Powertrain reaction torques against time (seconds)

model to interact with the kerbs correctly, once more shown by Figure 5.

The target for the driver is to maintain a constant velocity of 1m/s, it holds 1st gear and doesn't use the brake. This allows the vehicle to progress over the kerb at a steady rate without applying brakes such that it rolls over smoothly.

5.2 High Speed Banked Circuit

The other test being investigated in this paper is a set of high-speed laps around an oval track with banked corners. In this test the vehicle will be lapping the circuit with both open and closed demands. The roughness will be a smoothest surface Grade A, this matches the road roughness of a refined road track.

This shortened durability cycle includes high-speed laps, high braking sections and incremental speed increases to test elongated driving in different gears. This evaluates drive train durability and is used to gain loads both throughout the drivetrain and reaction forces/torques to the drivetrain mounts.

5.3 Results

The chassis twist experiment generated wheel, wishbone, chassis mount and strut top loads that can be used with FEA and fatigue analysis software to generate lift time analysis of both suspension components and the chassis. Figure 6 shows these results, displaying from a qualitive viewpoint the expected wheel load trends, within a 'sensible' force range. As a generic vehicle model was used, the actual specific numerical values of the presented results are not of primary importance. Rather, the trends shown are representative, indicating the simulation is behaving in an accurate manner. With specific vehicle data, this test could be modified to gain kerb strike analysis if the speed was increased to gain a more violent reaction from the wheel input. It can also be extended to move mass around the cabin and include trailers to investigate peak tow bar loads. These results, combined with other, more varied tests can generate a wider understanding of the life time requirements of suspension components. The high-speed durability test (Figure 7) provides additional, higher frequency loads applied to all components, improving the fatigue analysis ability. This test specifically yielded engine reaction and transmission loads applied to the transmission housing mounts at higher frequencies. Changes in the engine and transmission mount torque can be observed as the result of the actions undertaken by the test driver. Conducting such a simulation would provide to the user the expected loads experienced by the vehicle undergoing this test, which could be deployed in more advanced FEA or fatigue analysis software for detailed analysis. Furthermore, they could be used on their own with fatigue metrics to understand how design changes will effect the lifecycle fatigue load the vehicle will endure. Similar caveats regarding the specific numerical value of the results presented in Figure 7 apply as were explained for Figure 6.

6 Conclusions

The results presented in this paper demonstrate the capability of the VeSyMA suite by Claytex to provide a virtual testing solution which is capable of recreating proving ground tests, with the introduction of a new driver model, multi point tyre contact models and road modelling features as presented. Furthermore, other methods available to the user through the VeSyMA suite are presented, which can be used to include further detail into their analyses.

Example testing scenarios presented show that the vehicle model behaves in a representable manor when travelling at high speed on a banked proving ground circuit, demonstrating the validity of the vehicle and the road model. Multi-point tyre contact models are shown to enable representative rough road and kerb strike experiments to be conducted. The same vehicle model was deployed in both example experiments, albeit with different tyre contact models. This delivers on the assertion of the flexibility of the VeSyMA suite of libraries as a simulation solution, with model reuse enabling the user to focus more on the investigation at hand rather than reformulating simulation models.

6.1 Further work

Virtual testing is not just limited to the offline examples demonstrated in this paper. To fully harness the advantages of the simulation solution, it is recommended that such detailed offline work be deployed alongside other forms of simulation, such as DiL and Hardware-in-the-Loop (HiL) testing. Furthermore, as alluded to in the paper, the fidelity of the vehicle model can be improved upon by deploying other subject-specific libraries. Incorporating the VeSyMA – Terrain Server library would enable both detailed offline simulation and DiL testing to be deployed effectively in conjunction with one another.

Related Work

H. Hammond-Scott and M. Dempsey (2018). Vehicle Systems Modelling and Analysis (VeSyMA) Platform. *Proceedings of the 2nd Japanese Modelica Conference, 2018.*

M. Dempsey, G. Fish and J. G. Delgado Beltran (2015). High Fidelity Multibody Vehicle Dynamics Models for Driver-in-the-Loop Simulators. *Proceedings of the 11th Modelica Conference, 2015.*

References

M. Agostinacchio, D. Ciampa and S. Olita (2013). The vibrations induced by surface irregularities in road pavements – a Matlab approach. *European Transport Research Review.* (2014) 6:267–275. DOI: 10.1007/s12544-013-0127-8

T. Berg, (2015). *Navistar to Use Indiana Proving Grounds For Extensive Testing*. Online. Accessed 11 January 2019. Available at: https://www.truckinginfo.com/129556/navistar-to-use-indiana-proving-grounds-for-extensive-testing

Cosin Scentific Software (2018). *FTire: Features and Capabilities*. Online. Accessed 08 November 2018. Available at: *https://www.cosin.eu/products/ftire/*

H. Dankowicz (1999). Modelling of dynamic friction phenomena. *ZAMM* 1999;79:399–409.

A. Halfpenny and M. Pompetzki (2011). Proving Ground Optimization and Damage Correlation. *SAE International. DOI: 10.4271/2011-01-0484*

H. Hammond-Scott and M. Dempsey (2018). Vehicle Systems Modelling and Analysis (VeSyMA) Platform. *Proceedings of the 2nd Japanese Modelica Conference, 2018.*

C. F. F. Karney (2011). Transverse Mercator with an accuracy of a few nanometers. *SRI International*. DOI: 1002:1417

K. Kawase (2013). Concise Derivation of Extensive Coordinate Conversion Formulae in the Gauss-Kruger Projection. *Bulletin of the Geospatial Information Authority of Japan.* Vol. 60 March, 2013.

National Geospatial-Intelligence Agency (2005). *World Geodetic System 1984*. Technical Bulletin, United Nations Office for Outer Space Affairs, 2005.

OpenCRG, 2018. *Background.* Online. Accessed 09 November 2018. Available at: http://www.opencrg.org/project.html

OpenDRIVE, 2018. *Background.* Online. Accessed 09 November 2018. Available at: http://www.opendrive.org/project.html

Hans B. Pacejka (2012). *Tyre and Vehicle Dynamics*, 3rd edition. Oxford: Elsevier, pp 356-404.

J. Rauh, H. Schindler, L. Witte, T. Kersten and W. Zipperer (2008). *OpenCRG: A unified approach to represent 3D road surface data.* AK 6.1.3 Tire Models for Vehicle Dynamics 11.12.2008.

J. Segers (2014). *Analysis Techniques for Racecar Data Acquisition,* 2nd edition. Warrendale, PA. USA: SAE International.

U.S. Geological Survey (1997). The Universal Transverse Mercator (UTM) Grid. *USGS Fact Sheet 149-97.* September, 1997.

Hierarchical Coupling Approach Utilizing Multi-Objective Optimization for Non-Iterative Co-Simulation

Franz Rudolf Holzinger[1] Martin Benedikt[1]

[1]Department Electric/Electronics & Software, VIRTUAL VEHICLE Research Center, Austria,
{franzrudolf.holzinger,martin.benedikt}@v2c2.at

Abstract

A hierarchical scheduling approach for non-iterative co-simulation is presented. With an increasing number of subsystems the number of possible combinations and permutations increases dramatically, resulting in an unsolvable problem to define a proper co-simulation scheduling for application engineers. This paper shows an approach to get an optimal trade-off between simulation duration and simulation accuracy by the usage of a multi-objective optimization approach to find an optimal scheduling for hierarchical co-simulation.

Keywords: hierarchical co-simulation, co-simulation graph, multi-objective optimization

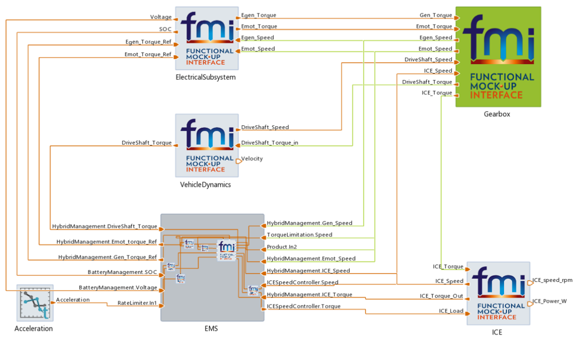


Figure 1. Co-simulation topology of the HEV example.

1 Introduction

Virtualization of products is common practice in industry in order to reduce costs in design, analysis and test phases. Tailored simulation tools are available for covering the different engineering domains and applications. However, when it comes to overall system considerations the different subsystems must be virtually integrated for enabling analysis of their interactions. In contrast to a time-consuming remodeling, by the co-simulation approach the individual subsystems are simulated within their dedicated simulation tools and predefined coupling variables are exchanged at specific points in time for synchronization purposes (Kübler and Schiehlen, 2000).

First activities in the field of co-simulation were published in the 1970ies and 1980ies, motivated by the idea of massive parallelization for electrical circuit simulation in analog system design (Lelarasmee et al., 1982). Algebraic constraints and non-linear behaviours of the components require implicit numerical solvers for these applications, which were directly implemented within the dedicated simulation environments. Nowadays, a lot of domain-specific simulation tools are available on the market, some of them are equipped with dedicated API's (application programming interfaces) for enabling an integration in terms of co-simulation. Most of the tools only supports the exchange of coupling variables at predefined points in time (no iterations over steps; no exchange of model information), which renders the co-simulation approach to a pure explicit numerical scheme in general. The FMI Standard (Functional Mockup Interface (Blochwitz et al., 2012)) represents a promising path for enabling broadly horizontal subsystem integration and the application of implicit co-simulation master algorithms. By considering a system simulation in general, the individual subsystems typically possess different dynamic ranges and properties, which makes the use of different step-sizes, kinds of extrapolation and orders of execution relevant. With the increasing number of integrated subsystems an engineer is typically not able to configure the co-simulation master as required for ensuring stable and accurate co-simulation results. But, as more information about the individual subsystems is available in beforehand or may be gathered online, i.e. during the co-simulation run, as more automation is possible in order to support the user in configuration of co-simulation settings. Recently this idea was discussed (Benedikt and Holzinger, 2016) by the authors; within this contribution especially the aspect of subsystem scheduling is addressed. Scheduling relates the proper selection of the order of execution of the involved subsystems, i.e. the determination of the optional trigger-sequence.

The outline of the paper is as follows. Coupling mechanism within non-iterative co-simulation are introduced. Based on this, the trigger sequence of sub-models is discussed. After that a multi-objective optimization approach is presented to determine a scheduling configuration for hierarchical co-simulation. An example is used to illustrate the different coupling configurations as well as the multi-objective approach.

The topology of the co-simulation example is shown in Figure 1. It represents a hybrid electric vehicle (HEV).

Table 1. Subsystem of the HEV example.

	Sub-Model	*Calc. Effort d_j*
S1	Electrical Subsystem	17.3%
S2	EMS Battery Management	1.9%
S3	EMS Hybrid Management	4.9%
S4	EMS ICE Speed Controller	2.3%
S5	EMS Product	1.9%
S6	EMS Rate Limiter	2.0%
S7	EMS Torque Limitation	1.8%
S8	Gearbox	27.8%
S9	ICE	11.4%
S10	Vehicle Dynamics	28.7%

The HEV system is based on a Matlab/Simulink example and was split into 10 subsystems (Miller, 2017). The subsystems, compiled as FMUs, are integrated within a co-simulation framework (Model.CONNECT^{TM}, R2018a). Table 1 represents the sub-elements of the HEV example with their calculation effort d_j.

2 Coupling Mechanism

The most common used coupling approach for co-simulation is to calculate all sub-models at the same time. Each subsystem has not to wait for each other and so this coupling mechanism has the best simulation performance. Nevertheless, this parallel coupling approach causes the most coupling errors, due to the high number of extrapolated inputs. If subsystems are calculated sequentially, i.e. a subsystem starts the calculation when the previous subsystems already finished the calculation step, no inputs have to be extrapolated. With a sequential coupling approach a minimum number of extrapolation can be reached, but the simulation performance will suffer. A hierarchical approach on the other side allows a combination of sequential and parallel scheduling. Several subsystems can be nested, where e.g. the subsystems within a group are calculated in sequential order and the several subsets (groups) are calculated in parallel.

2.1 Simulation Performance

The simulation performance referenced by the real-time behaviour for different coupling mechanism related to the HEV example is shown in Table 2. The real-time factor (RTF) describes the relation between simulation duration and the wall-clock time. A real-time factor $RTF < 1$ means faster than wall-clock time and $RTF > 1$ means a simulation duration greater than the wall-clock time. For real-time applications it is required, that the $RTF < 1$ in each coupling time step, otherwise the real-time behaviour is not ensured.

The performance results in Table 2 show, that the real-time behaviour of the parallel coupling approach is lower than real-time. On the other hand, the sequential coupling approach has a real-time factor greater than one, i.e. it is not real-time capable. From the timing point of view a parallel coupling approach represents a reasonable co-simulation scheduling setting.

Table 2. Real-time capability regarding the coupling mechanism.

Coupling Mechanism	*RTF*
parallel	0.44
sequential	1.1
hierarchical	0.5 – 0.75

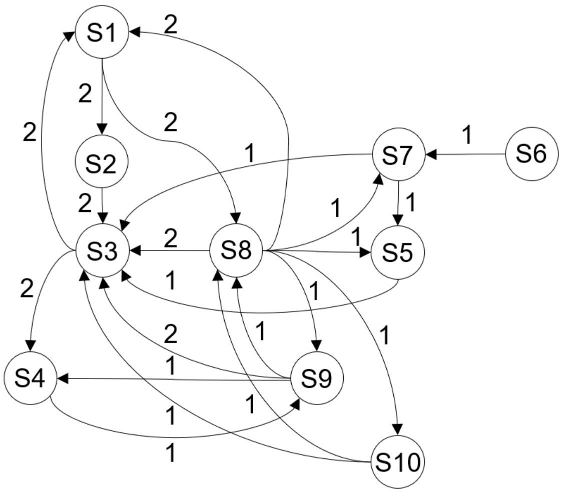


Figure 2. Co-simulation graph of the HEV example.

2.2 Trigger Sequence

The calculation order or trigger sequence of a sequential coupling mechanism defines the extrapolated inputs and so the induced extrapolation errors.
Related to HEV example with $n = 10$ subsystems it exists $n! = 3628800$ different permutations to set the calculation order of the subsystems. With the knowledge of the topology it is possible to find at least an optimal trigger sequence with respect to a minimal number of extrapolations (Glumac and Kovacic, 2018).

Co-simulation networks can be interpreted as a directed graphs. The nodes of the graph represent the several subsystems and the edges describe the directed dependences of the subsystems. The weight of the edge describes the strength of the dependency e.g. the individual number of connections between the subsystems.

Figure 2 illustrates the co-simulation graph of the HEV example with 10 subsystems. The edges show the directed dependency of the subsystems and the weight of the edges represent the number of signals which are exchanged between the subsystems.

The trigger sequence can be interpreted as a Hamilto-

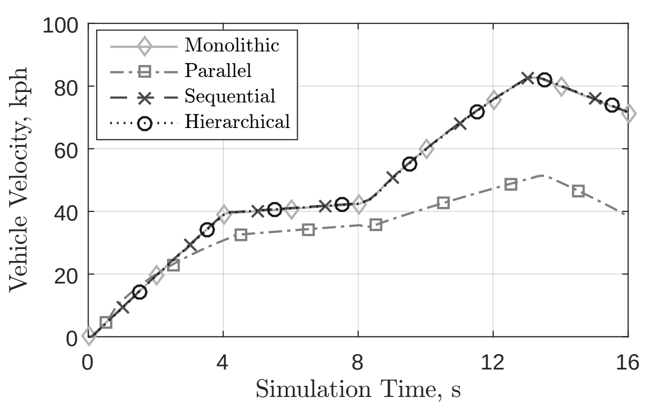


Figure 3. Simulation result (Vehicle Velocity) regarding different coupling mechanisms.

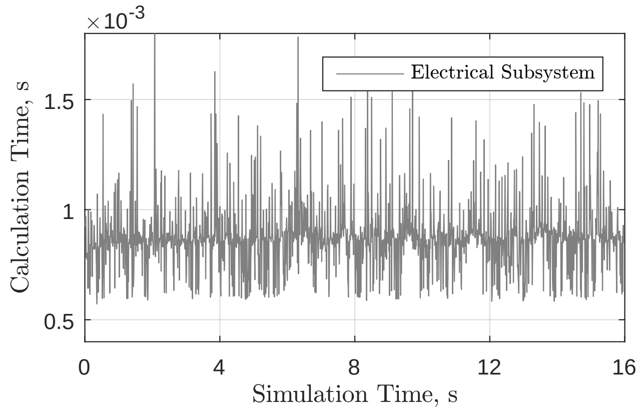


Figure 4. Calculation time of the Electrical Subsystem.

nian cycle, where each node has to be visited once. The weights of the edges represent the number of inputs to be extrapolated. The shortest way to visit all nodes represents the optimal trigger sequence with respect to minimal number of to be extrapolated inputs. Nevertheless, the connections of already visited nodes do not need an extrapolation and so the weights of these edges (connections) becomes zero.

A comparison of sequential and parallel coupling approach (in contrast to the monolithic simulation) is shown in Figure 3. The sequential simulation delivers almost the same results than the monolithic simulation. The results of the parallel coupling approach clearly differ from the reference.

3 Optimal Hierarchical Approach

Parallel scheduling turns out the best behaviour with respect to the simulation duration but induces, on the other hand, the most coupling errors into the co-simulation. A minimal number of extrapolated inputs can be reached by a sequential coupling approach at an expense of the simulation duration. A hierarchical coupling approach allows to find an optimal trade-off between simulation duration and accuracy.
The permutations of the subsystems with sequential method enlarges by the number of possible combinations. An upper estimate of possible combinations in consideration of the simple nested scheme (parallel scheduling of sequential calculated groups) is as follows:

$$\sum_{j=0}^{n-1} \binom{n-1}{j} n! = 2^{n-1} n!. \tag{1}$$

For a number of $n = 10$ subsystems there are almost $1.8 \cdot 10^9$ possible combinations and permutations for setting the calculation order. This huge number makes it practically impossible for co-simulation application engineers to find a proper set without any automated optimization approach.

Beforehand some assumptions have to be applied:

- The calculation time (the time which the subsystem needs to compute the results) is significantly bigger than the synchronisation time (time which is needed to exchange the coupling signals). So the synchronisation time is unattended.
  Note: For a high number of coupling signals or fast calculating subsystems, the synchronisation time is not negligible. In this case both, the synchronisation time and the calculation time have to be considered.

- The calculation effort has an ergodic behaviour. Figure 4 shows the calculation duration of the model *Electrical Subsystem* (S1) during the simulation. The mean computation duration is about $9\,ms$ and is almost constant throughout the simulation.

- The coupling error is directly associated to the number of extrapolations and degrades overall co-simulation accuracy.
  Note: In general this is not the case. The introduced coupling error depends on the subsystem dynamic as well as the coupling signal behaviour. Nevertheless, if no additional subsystem information is available and the system behaviour is unknown, the introduced coupling error can be assumed equal for each input.

The relative calculation effort of the several subsystems is illustrated in Table 1. The *Vehicle Dynamics* (S10), *Gearbox* (S8) and the *Electical Subsystem* (S1) together need about 75% of the computation time. It is obvious to combine several subsystems and calculate the different sets in a parallel way, to reduce the overall calculation effort.

A combination set $C = \{C_1, C_2, ..., C_i\}$ consists of several subsets C_j and includes all subsystem indices. The duration D_j of a subset C_j is calculated as follows:

$$D_j\left(C_j\right) = \sum_{i \in C_j} d_i, \tag{2}$$

where $d_i \in \mathbb{R}$ is the calculation effort of the individual subsystems S_i (see Table 1). The objective function J_D concerning the calculation effort can be written as the maximum duration of all subsets scaled by the sum of all elements:

$$J_D = \frac{1}{\sum_i d_i} \max_j D_j. \tag{3}$$

With respect to the HEV example a reasonable combination C consists of the subsets $C_1 = \{6,10\}, C_2 = \{1,2,3,8\}$ and $C_3 = \{4,5,7,9\}$. Based on the effort of the combinations $D_1 = 30.7$, $D_2 = 51.9$ and $D_3 = 17.4$ a cost value $J_D = 0.519$ is determined.

For each possible subset C_j an optimal calculation order can be found, where the number of extrapolated inputs is minimized. Therefore the adjacency matrix A is used.

$$A = \begin{bmatrix} - & 0 & 2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 2 & - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & - & 0 & 1 & 0 & 1 & 2 & 2 & 1 \\ 0 & 0 & 2 & - & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & - & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & - & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & - & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & - & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & - & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & - \end{bmatrix} \tag{4}$$

The adjacency matrix A in (4) represents the graph of the HEV example (see Figure 2). The number of required extrapolated inputs, which is directly associated to the simulation accuracy, can be calculated as follows:

$$e_k = \sum_{i=1}^{N} A_{ik}. \tag{5}$$

Beginning with one element of the subset $k \in C_j$ the whole column is summed up. This represents the whole number of extrapolated inputs for the index. Regarding to the HEV example for an index $k = 3$ the whole column is summed up and $e_3 = 4$.

In contrast to the calculation effort, the number of extrapolated inputs is depending on the execution order of the subsystems. Therefore the entire row of the considered index k (or subsystem) has to be set to zero.

$$A_{ki} = 0, \quad i \in C_j \tag{6}$$

If the node 3 is already visited, the subsystem has been calculated and the results are available. There is no extrapolation needed anymore for these coupling signals, i.e. the row of the node has to set to zero, see (7).

$$A = \begin{bmatrix} - & 0 & 2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 2 & - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & - & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & - & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & - & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & - & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & - & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & - & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & - \end{bmatrix} \tag{7}$$

The extrapolation effort E_j of the subset C_j is given as sum of the required to be extrapolated inputs.

$$E_j(C_j) = \sum_{i \in C_j} e_i \tag{8}$$

The normalized objective function regarding the number of extrapolations can be written as follows:

$$J_E = \frac{1}{\sum_j \sum_i A_{ij}} \sum_j E_j. \tag{9}$$

A multi-objective optimization problem with minimization of number of extrapolated inputs and minimization of the simulation duration can finally formulated as follows:

$$\min \{(1-w)J_E + wJ_D\}, \tag{10}$$

where the factor w enables to set the focus of the optimization to the extrapolation error J_E or to the calculation duration J_D. A small factor w weights the optimization in the direction of the minimum extrapolation error and so sequential calculation is preferred. On the other hand

Table 3. Optimized hierarchical scheduling with respect to weighting factor w.

w	*Trigger Sequence*
0.0	(S6),S10,S9,S8,S7,S5,S3,S4,S1,S2
0.25	(S6),S10 S9,S8,S7,S5,S3,S4,S1,S2
0.5	(S6),S10 S9,S8,S7,S5,S3,S4 S1,S2
0.75	(S6),S10 S9 S8 S7 S5,S3,S4,S1,S2
1.0	(S6),S10 S9 S8 S7 S5,S4,S3,S2,S1

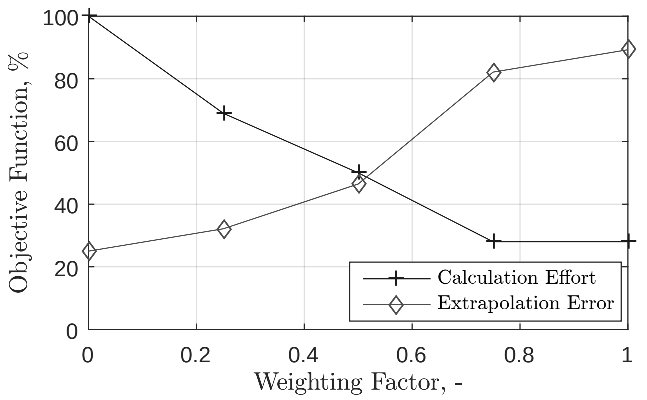


Figure 5. Relative calculation effort and relative extrapolation error regarding weighting factor w.

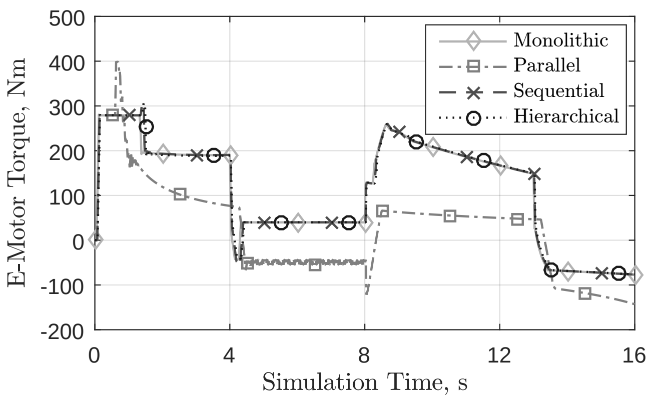


Figure 6. Simulation result (Electric Motor Torque) regarding different coupling mechanisms.

the factor $w = 1$ is the focus on the optimization of the simulation duration and so parallel approach is selected.

The solutions of the hierarchical optimization problem for the HEV example is shown in Table 3. The optimization is analysed by different weighting factors $w = [0, 0.25, 0.5, 0.75, 1]$. The order of the subsystems in a row indicates the trigger sequence. The several rows within the w cases mean the parallel calculation of these bundles. In the case of $w = 0.5$ the combined models $S6, S10$ are calculated parallel to the sequential calculated group $S9, S8, S7, S5, S3, S4$ and $S1, S2$. The subsystem $S6$ is considered separately and therefore in Table 3, it is written in parentheses because it has no dependencies on other subsystems.

The overall simulation duration increases with increasing w and, on the other hand, the number of extrapolated inputs decreases. The behaviour of the extrapolation error and the calculation effort regarding the weighting factor w for the HEV example is shown in Figure 5. A proper trade-off between simulation duration and accuracy for this example is at $w = 0.5$.

The comparison of the different coupling approaches is illustrated in Figure 6. The sequential and hierarchical ($w = 0.5$) approach are equal to the monolithic simulation result. The result of the parallel scheduling shows a different behaviour.

The HEV example in Table 3 shows the identical results regarding the calculation order for different weighting factors. Only the parallelization of subsystems changes dependent on w, because in contrast to the optimization part regarding the extrapolation error J_E, the calculation effort J_D is not dependent on the execution order. Therefore it is conceivable that the execution order is calculated first and after that the parallelization is determined. This will at least reduce the computation effort to find an optimal trigger sequence.

4 Conclusion and Future Work

Especially with increasing number of subsystems hierarchical co-simulation can bring a balance between simulation duration and simulation accuracy. A high number of subsystems makes it almost impossible to find a proper configuration set regarding the execution order. The HEV example in this work shows that the simulation duration of parallel scheduling is better than sequential coupling but the simulation result on the other hand differs significantly from the monolithic simulation caused by the introduced extrapolation errors. The presented hierarchical approach allows a real-time capable simulation with accurate simulation results.
In the discussed HEV example all subsystems have the same coupling time-steps. For future works hierarchical co-simulation with different coupling time-steps is considered. Therefore subsystems with similar dynamics and coupling time-steps are clustered together. So they can be interpreted as a separate hierarchical layer.

Acknowledgements

This work was accomplished at the VIRTUAL VEHICLE Research Center in Graz, Austria. The authors would like to acknowledge the financial support of the COMET K2 - Competence Centers for Excellent Technologies Programme of the Austrian Federal Ministry for Transport, Innovation and Technology (bmvit), the Austrian Federal Ministry of Science, Research and Economy (bmwfw), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

References

M. Benedikt and F. R. Holzinger. Automated configuration for non-iterative co-simulation. In *2016 17th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics* *and Microsystems (EuroSimE)*, pages 1–7, April 2016. doi:10.1109/EuroSimE.2016.7463355.

T. Blochwitz, M. Otter, J. Åkesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International Modelica Conference*, pages 173–184. The Modelica Association, 2012. ISBN 978-91-7519-826-2. URL `http://dx.doi.org/10.3384/ecp12076173`.

Slaven Glumac and Zdenko Kovacic. 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, pages 157–160, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5092-1. doi:10.1145/3200921.3200924. URL `http://doi.acm.org/10.1145/3200921.3200924`.

R. Kübler and W Schiehlen. Modular simulation in multibody system dynamics. *Multibody System Dynamics*, 4, 2000.

E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1(3):131–145, July 1982. ISSN 0278-0070. doi:10.1109/TCAD.1982.1270004.

Steve Miller. Hybrid-electric vehicle model in simulink. (`https://www.mathworks.com/matlabcentral/fileexchange/28441`) MATLAB Central File Exchange, 2017. Accessed: 2017-08-30.

Model.CONNECT^{TM}. Hybrid electric vehicle (hev). (`https://www.avl.com/-/model-connect`), R2018a. Accessed: 2018-04-11.

Poster Session

Flow Network based Diagnostics for Incorrect Synchronous Models

Hans Olsson[1]

Dassault Systemes, Sweden, hans.olsson@3ds.com

Abstract

This paper will present a novel way to give diagnostics for incorrect synchronous models.

The goal is that this will ease the introduction of synchronous models, since unclear diagnostics often create a barrier for new users. In particular the case of separating the clocked and continuous parts will be considered, and shown to be equivalent to finding a "leak-flow" in a certain flow network, which can be solved using max-flow/min-cut techniques.

The result is efficient, easy-to-adapt, and gives diagnostics focused on correcting the issue.

We have not seen this idea used before in this context, even if in retrospect it seems natural and straightforward.

The methods have been implemented in Dymola 2019 (released in June 2018) and also in 3D Experience Platform 2019x.

Keywords: synchronous, graph theory, flow networks, minimal cut, error diagnostics

1 Introduction

Diagnostics for incorrect Modelica models is an important part of Modelica tools. Tools can implement advanced diagnostics, either by additional analysis based on the current Modelica language (Bonus and Fritzson, 2002), or in combination with adding restrictions to Modelica such as balanced models (Olsson *et al*; 2008).

After a short discussion about different forms of diagnostics for errors, we will start by introducing the synchronous part of Modelica 3.3, and then flow networks and the max-flow/min-cut theorem.

When presenting error diagnostics, important aspects include how early the diagnostics is given, and how localized the error is. The ideal situation is early detection and that at least one plausible correction is clearly located.

In particular, some diagnostics can be given as soon as the error is made, and tools can in those cases prevent the error from being introduced in the model, e.g., attempting to connect an electrical pin to a mechanical flange.

Other diagnostics can only be given when translating the complete model, e.g., missing a source signal in an expandable connector set.

An intermediate variant is those where we can give diagnostics for incomplete models without introducing false positives – i.e., we avoid diagnostics for issues that will naturally be corrected as part of completing the model; but it is still a global property.

The clock partitioning problem is one of these intermediate variants, which adds the restriction that the diagnostics should work on such incomplete models – in particular when equations are missing.

That also implies that we could present the diagnostics after each operation, but that is currently not implemented. The errors are not necessarily local – but it may still be that they could be corrected in one or a few places.

2 Synchronous Modelica

Modelica 3.3 added synchronous primitives (Elmqvist *et al*, 2012) intended to make it easier to model control systems that run on a sampled clock and connect to the continuous plant model. This section will only describe the concepts needed in this paper and is not a general introduction to synchronous modeling.

To illustrate we have a simplified model illustrating some of the concepts:

```
model FirstOne
  Real x,y,z;
equation
  when Clock(1) then
    2*x=sample(y);
  end when;
  when Clock() then
    z=x+1;
  end when;
  y=hold(z)+time;
end FirstOne;
```

The equation `2*x=sample(y);` is a clocked equation and only active when the corresponding clock ticks (every second as given by `Clock(1)`). Note that it is an actual equation – but only active when the clock ticks, in contrast to non-clocked when-clauses in Modelica which only allow a restricted form of equations where the left-hand side must be a variable.

One important aspect of the synchronous extension is that variables and equations are not declared to be continuous or clocked (in the example `x` and `z` are clocked and `y` is continuous), instead the clock-partition can be inferred using "clock inferencing".

Additionally, sub-models can be used in both the continuous and clocked domains; be restricted to one domain or the other, or connect the two domains in certain ways. That the same sub-model can be used in both domains imply that we cannot infer properties of the used models in general – but only infer properties for each specific component of those sub-models.

Specifically equations in `when Clock` must be clocked, and `sample` takes a continuous time input and returns a clocked value; and `hold` converts in the other way. It is not possible to directly use a clocked variable when it is not active, instead one must explicitly use `hold(z)` to get the last active value for `z`. Additionally `time` is always continuous time.

If the modeler makes mistakes, the clock inferencing may fail. A trivial example would be writing `y=z+time;` in the model above, since `time` must be continuous and `z` is given by a clocked equation. The algorithm in this paper gives diagnostics and recommended solution for such errors (mixing clocked and continuous) – and the algorithm is particularly suited for larger models where there are a large number of (potentially incorrect) intermediate steps in the inferencing.

In this example the clock for `z` is not specified, but automatically inferred to be the same as for `x`. The `Clock(1)` could also be duplicated, and it is then verified that the two clocks tick at the same time – ensuring that different clocks are not used accidentally. The example could also be written as:

```
model SimplerFirst
  Real x,y,z;
equation
  2*x=sample(y, Clock(1));
  z=x+1;
  y=hold(z)+time;
end SimplerFirst;
```

In this case we automatically infer that `x` and `z` are clocked, and `y` continuous time. The equation `z=x+1;` can on its own be either clocked or continuous time (and could in general be in a sub-model that works in both parts).

An advanced feature is that clocked equations may include differential equations provided a discretization method is specified for the clocked partition, e.g.:

```
model Discretized
  Real x,y,z;
equation
  2*x=sample(y, Clock(Clock(1),
        solverMethod="ExplicitEuler"));
  der(z)=x+1-z;
  y=hold(z);
end Discretized;
```

The solverMethod argument ensures that any differential equation in the partition (in this case `der(z)=x+1-z;`) will cause the corresponding variable, `z`, to be updated using one step of explicit Euler when the clock ticks. The `der`-operator cannot occur in clocked partitions without a deduced solverMethod (there are additional details regarding different partitions that are not relevant for the analysis in this paper).

A final important aspect is that Modelica supports graphically connecting components – continuous, clocked, and even components mixing the two domains. There are also libraries of models, including Modelica_Synchronous that contain tested standard models.

3 Flow networks

A flow network (Ford and Fulkerson, 1956; or any general overview such as Cormen *et al*, 1993); is a directed graph where each edge has an arbitrary nonnegative capacity. When modifying the graph the capacities can become zero, and in that case we view it is as if the edge is not present.

A flow in such a network satisfies a number of constraints, in particular the flow in each edge does not exceed its capacity and except for source and sink vertices the in-flow to a vertex matches the out-flow from that vertex. In Figure 1 a small flow network with flows is shown, the source is marked with “s” and the sink with “t” (for target) and each edge has two numbers, the first is the current flow and the second is the capacity. The edges where the current flow equals the capacity are saturated.

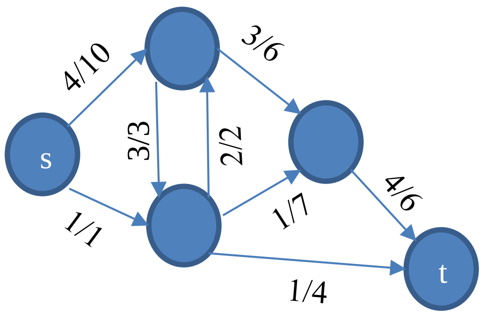


Figure 1 A small flow network.

Without loss of generality we can assume that there is only one source and one sink (Ford and Fulkerson, 1956). If there are e.g., multiple sources it is known that we can introduce a “super-source” with edges of infinite capacity going to each source, and treating those original sources as normal vertices; unless there are additional restrictions on the flows.

3.1 Minimal cut theorem

A **disconnecting set** of edges partitions the vertices into two sets – one containing the source and another the sink. A disconnecting set without redundant elements is a cut.

The max-flow min-cut theorem, also known as “minimal cut theorem” (Ford and Fulkerson, 1956); states that the maximal flow obtainable in a network is the minimum of the sum of capacities of the edges in the set taken over all disconnecting sets. (Note: even if it is the minimum for all disconnecting sets the minimum is clearly for a disconnecting set without redundant elements, i.e., for a cut.)

If we revisit the previous small flow network we see that the maximum flow is 10, and the minimal cut is shown in red in Figure 2; and the other edges as dotted.

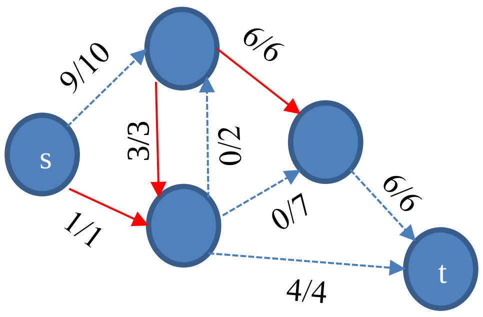


Figure 2 Minimal cut

Note that the minimal cut is not necessarily unique – another option would be to replace the 6/6 edge with the other 6/6 edge, and a third option would be the two edges going into “t”. The two edges going from the source are clearly a cut, but its total capacity is 11 and it is therefore not a minimal cut. The red 6/6 edge and the 4/4 have a sum of capacities of 10, but is not a disconnecting set and thus not a cut. The red edges in union with the 4/4 edge form a disconnecting set that is neither a cut nor minimal.

There exists a number of algorithms for constructing the minimal cut and the maximum flow, with different running time in terms of number of edges and vertices; (Cormen *et al*, 1993).

3.2 Augmentation path

If a flow network allows a flow between the source and the sink we can find a path – called chain of edges in (Ford and Fulkerson, 1956) connecting them. The maximum flow through that path is the minimum capacity of any of the edges in the path.

After “subtracting” this flow from the graph one can attempt to find an additional path (called “augmentation path”) connecting the source and the sink, and repeating this leads to the algorithm called Ford-Fulkerson based on (Ford and Fulkerson, 1956).

Subtracting the flow means both reducing the capacities of the used edges, and adding a capacity in the reverse direction; the latter is needed since we will sometimes later reduce the flow through specific vertices.

The path will later be used for error diagnostics, and thus redundant edges will cause a problem in at least two cases:

If the graph has cycles a vertex could appear multiple times in the path, but that can only decrease the flow through the path and the algorithm thus avoids revisiting vertices.

Additionally, if there are multiple sources a path could start at one source and then have an edge leading to a different source (and similarly for sinks). By treating all sources as visited at the start and avoiding revisiting vertices that is avoided for the sources, and by stopping at the first sink reached it is avoided for sinks.

The current implementation does not use a breadth-first search for the path, but that would naturally avoid the previous issues.

A major restriction of the algorithm is that this only converges in a finite number of steps if the capacities are integers (or in general rational numbers); and has a running time of O(number of edges*maximum flow). This follows from the fact that we can find one augmentation path in running time proportional to the number of edges, and each augmentation path has a flow of at least one.

We currently do not use any specific heuristic for finding the augmentation path, but a breadth-first search is generally a good heuristic avoiding specific problems for large maximum flow (Cormen *et al*, 1993).

Assuming the maximum flow is small this simple algorithm compares favorably to recent algorithms; that instead are superior if the maximum flow is large or the capacities are real numbers; as their running time only depend on the number of edges and vertices.

4 Min-cut and Clock partition

We will now combine the clock partition and the flow network.

4.1 Flow networks for synchronous models

Based on a model with synchronous parts we can construct a flow network where the variables and equations that must be continuous are sources, and variables and equations that must be clocked are sinks.

Both equations and variables are vertices in this graph, and edges connect equations to variables appearing in the equation – unless the variables appear inside certain primitives, in this paper we will only discuss `sample` and `hold`, but in general it also includes “Clock with Boolean condition”. The variables inside these primitives are instead sources or sinks. The edges are also added in the opposite direction (with the same capacity in both directions) so that we get a symmetric directed graph. Alternatively, we can replace this pair of edges with one bidirectional edge with capacities in both directions – initially equal.

Having edges in both directions implies that there are cycles in the graph, but the chosen algorithm can handle that.

The capacities for all edges can be selected as positive integers; the exact values will be discussed later.

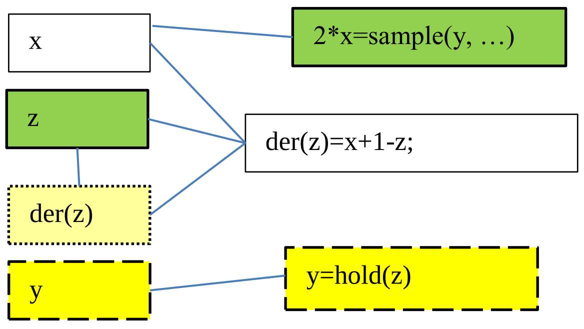


Figure 3 Flow network for the model "Discretized"

In Figure 3 we see the flow network corresponding to the previous model "Discretized" illustrating most of the concepts, where continuous equations and variables (argument of `sample`) are marked in yellow (and dashed outline) and clocked equations and variables (argument of `hold`) are marked in green. The `der(z)`-variable is special and marked in lighter yellow (and dotted outline), indicating that without a solverMethod it must be in a continuous partition (which would cause a leak-flow between the partitions). There is also an edge from `der(z)` to `z` indicating that they should be in the same partition (in later graphs derivative-variables will not be separate nodes). Since there is a solverMethod attached to the clock-partition, `der(z)` is just a normal variable and there is no leak-flow.

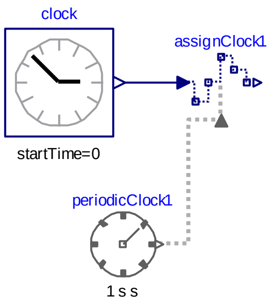


Figure 4 Incorrect assignClock

In Figure 4 we have an incorrect model from MCP-0030 (Frenkel, 2018). The corresponding flow network is shown in Figure 5, where the continuous part (due to `time`) is marked in yellow and the clocked part (due to `when Clock()`) in green, and the edges that are not part of the augmentation path are dotted. The arrows on the edges indicate the direction of the flow.

The saturated edges (i.e., potential minimal cuts) are shown in red and wider. It is common that the saturated edges occur in pairs and the implementation handles that, but we will in the future investigate alternative formulations that avoid this.

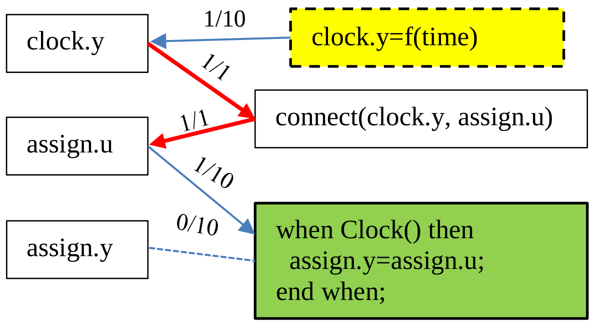


Figure 5 Synchronous simple flow network

The graph is bipartite with equations and variables forming the two parts as is normal in Modelica, but since both sources and sinks can appear in both parts this fact does not seem useful for analyzing this flow network. This implies that the edges in the cut can go both from equation to variable and vice-versa.

Note that the problem of assigning variables to equations in Modelica is equivalent to solving a maximum-flow problem on such a bipartite graph.

4.2 The significance of the flow

A correct model can be partitioned into zero or more clocked parts, and zero or more continuous parts. This corresponds to separating the graph into separate parts, and thus a zero flow.

If the flow is positive it indicates that graph cannot be partitioned in this way and the flow gives "leakage" between continuous and clocked parts. If there are multiple disjoint errors there will be multiple "leakages" increasing the flow; i.e., a higher flow can be seen as an indication of a more incorrect model.

The cut indicates which variables to remove from the equations to restore the partition. Replacing the variables by `sample()` or `hold()` variants of the same variables removes the edge, without excessively altering the model structure.

Similarly, the corresponding path(s) between source and sink is important, since that allows the user to see that there is an unwanted path between the clocked and continuous parts.

4.3 The capacities of edges

The previous method would work for any set of positive integers as edge-capacities, and give some cut between clocked and continuous parts.

Good diagnostics for errors can thus be seen as finding a suitable heuristic for the capacities. The basic idea is that we give high capacity to edges that we do not want in the cut-set; or roughly that high capacity corresponds to high trust in that equation.

As a first attempt, we chose capacity 1 for edges corresponding to connection-equations, and 10 for other edges. In the future, we are considering having higher weights for equations from tested libraries.

4.4 Algorithm

The following presents pseudo-code outlining the algorithm. Note that similarly as (Ford and Fulkerson, 1956) it is not a completely specified algorithm as there are multiple ways of finding the augmentation path.

Additionally, the source-cut part can use any algorithm that finds reachable nodes in a graph.

```
Sources={time}
Targets={}
Edges={}

// Build graph based on equations:
for eq in Equations loop
  // Low capacity for likely errors
  cap=if eq is connection then 1 else 10;
  for var in Incidence(eq) loop
    if var inside sample then
      Sources+={var};
      eq.isClocked=true;
    elseif var inside hold then
      Targets+={var};
      eq.isNonClocked=true;
    else
      // Edge(from->to, cap)
      Edges+={Edge(eq->var, cap)};
      Edges+={Edge(var->eq, cap)};
    end if;
  end for;
  if eq.isClocked then
   Targets+={eq};
  end if;
  if eq.isNonClocked then
   Sources+={eq};
  end if;
end for;

// Ford-Fulkerson finding max-flow:
maxFlow=0;
loop  // Find path of edges having cap>0
  augmentPath=FindPath(Sources, Targets);
  if augmentPath=={} then
    break;
  end if;
  // Possible flow for path
  flow=min(e.cap for e in augmentPath);
  maxFlow+=flow;
  // Subtract augmentPath flow:
  for e in augmentPath loop
    e.cap-=flow;
    reverse(e).capacity+=flow;
  end for;
end loop;

// Find source-cut, i.e. the cut
// closest to the source

// We first find all vertices
// reachable from the sources
SourceConnected={};
AddTo=Sources;
while not AddTo.empty() loop
  vertex=AddTo.front();AddTo.pop_front();
  if not vertex in SourceConnected then
    for e in Edges.from(v) loop
      if e.cap>0 then
        AddTo.push_back(e.target);
      end if;
    end for;
  end if;
end while;

// Find all edges with 0 capacity
// that leaves this set
SourceCut={};
for v in SourceConnected loop
  for e in Edges.from(v) loop
    if e.cap==0 and
      not (e.to in SourceConnected) then
      SourceCut+={e};
    end if;
  end for;
end for;
// And similarly for the target-cut
```

5 Examples

MCP-0030 (Frenkel, 2018) was made to solve the same problem as this paper. It contains one example of a bad model – shown in Figure 4. The rationale for the MCP was that earlier diagnostics just listed all equations and variables that failed for clock inference and it was not helpful for users. Note that the ideas presented here were implemented before the MCP.

With the approach in this paper, this example gives the diagnostics:

Continuous time parts and discrete parts don't decompose.
It is necessary to introduce sample or hold elements replacing:
connect(clock.y, assignClock1.u);
The following sequence indicates that the involved variables and equations are continuous time:
clock.y :
clock.y = clock.offset+(if time < clock.startTime then 0 else time- clock.startTime);
However, this is in contradiction with
assignClock1.u:
when assignClock1.clock then

```
    assignClock1.y = assignClock1.u;
end when;
```

The two sequences of equations and variables are constructed from the augmentation paths, cut according to the minimal cut.

It could be even clearer by showing that clock.y is continuous due to using time, and in this case the connection between clock.y and assignClock1.y was the obvious culprit.

We will now consider variations of an example from Modelica_Synchronous shown in Figure 6.

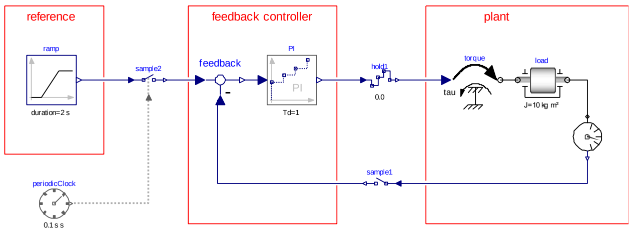


Figure 6 Textbook controller

If we had forgotten one of the synchronous primitives (there are two sample-blocks and one hold-block) the goal would be that diagnostics would recommend adding it.

The possible errors will be listed starting from the one ones that are simplest to investigate, and then proceed to the more complicated ones.

Without sample2 the diagnostics will pinpoint that:

```
It is necessary to introduce sample or hold elements
replacing:
  connect(ramp.y, feedback.u1);
```

If sample2 were replaced by a gain-block the diagnostics would be:

```
Continuous time parts and discrete parts don't
decompose.
It is necessary to introduce sample or hold elements
replacing:
  connect(ramp.y, gain.u);
or:
  connect(gain.y, feedback.u1);
or some variation of this.
```

The two proposed corrections are two different min-cuts (one close to source, one to sink). In general there could be a large number of possible min-cuts, and listing them all could be time-consuming and unlikely to help users.

If both sample1 and sample2 are missing, the diagnostics state:

```
It is necessary to introduce sample or hold elements
replacing:
  connect(feedback.y, PI.u);
```

This shows an interesting change, since the proposed change moves the feedback-component from the clocked part to the continuous part.

If only sample1 was removed the method described so far would see two possibilities:

```
connect(torque.tau, hold1.y);
connect(speed.w, feedback.u2);
```

The first suggestion has two problems: firstly, it would make more sense to remove the hold-block than introduce a sample-block, but secondly and more importantly, that model would not translate, since the der-operator is used in a clocked partition without solverMethod.

That is handled by running the algorithm on a slightly different flow network, which is constructed by considering the der-operator part of the continuous partition. That results in:

```
Continuous time parts and discrete parts don't
decompose, when there is no solverMethod attached to
the clock.
It is necessary to introduce sample or hold elements
replacing:
  connect(speed.w, feedback.u2);
```

Without hold1 we get:

```
Continuous time parts and discrete parts don't
decompose, when there is no solverMethod attached to
the clock.
It is necessary to introduce sample or hold elements
replacing:
  connect(torque.tau, PI.y);
```

Indicating that the correction is instead to introduce hold1.

If both sample1 and hold1 are missing the model is valid for check, but translation would give (flow network in Figure 7):

```
Continuous time parts and discrete parts don't
decompose, when there is no solverMethod attached to
the clock.
It is necessary to introduce sample or hold elements
replacing:
  connect(speed.w, feedback.u2); connect(torque.tau,
PI.y);
The following sequence indicates that the involved
variables and equations are continuous time:
  speed.w : speed.w = der(speed.flange.phi);
  torque.tau : torque.flange.tau = -torque.tau;
  torque.flange.tau :
load.flange_a.tau+torque.flange.tau = 0.0;
  load.flange_a.tau : load.J*load.a =
load.flange_a.tau+load.flange_b.tau;
  load.a : load.a = der(load.w);
However, this is in contradiction with
  feedback.u2 : feedback.y = feedback.u1-
feedback.u2;
  feedback.u1 : sample2.y = feedback.u1;
  sample2.y : sample2.y = sample(sample2.u,
sample2.clock);
  feedback.y : feedback.y = PI.u;
  PI.u : when Clock_0 then
    PI.x = previous(PI.x)+PI.u/PI.Td;
    PI.y = PI.kd*(PI.x+PI.u);
  end when;
```
The sub clock, BaseClock_0.SubClock_1, includes derivatives, but no solver method is specified.

The last line indicates that one way of correcting the model is to specify a solverMethod for the partition. In that case the plant-part will be discretized as part of the clocked partition.

However, the first part of the error message indicates that another solution is to introduce sample and hold. This error message lists two connect-statements, and the users has to replace both of them.

The part "The following sequence…" would as default be collapsed since it is quite long and only shows what is already stated. However, even if lengthy it still only lists relevant variables and equations, e.g. load.flange_a.phi and load_flange_b.phi are also part of the continuous-time partition – but they are not part of any augmentation path used and thus not included.

6 Implications for models

The examples demonstrate that no changes are needed to support these diagnostics. However, changes in models and/or the language can still be helpful to improve the diagnostics further.

In particular, models representing external controllers can currently be written as

```
model Controller
  extends SI2SO;
equation
  y=do_step(u1, u2);
end Controller;
```

Here do_step is an external C function (possibly part of an FMU), and each do_step updates the internal state and thus it should be run at every sampling point of the inferred clock.

That works in correct models where it is assigned to the clocked part and run at every sampling point. However, if the controller is incorrectly connected that can end up in the continuous part; which is not intended.

Modelica 3.4 (Olsson (editor), 2017) has restrictions for impure functions, but if `do_step` is not declared as impure that will not generate diagnostics. For the future we might consider treating impure function as sinks in the graph.

One possibility is to change the model to:

```
model Controller
  extends SI2SO;
equation
  when Clock() then
    y=do_step(u1, u2);
  end when;
end Controller;
```

That variant works and ensures that it is part of a clocked partition, providing better diagnostics – but it looks slightly distracting. A future possibility could be to introduce a form of "Clocked model" where all equations are seen as clocked.

7 Future work

For the future there are multiple lines of potential work – one is improving the current work by improving the diagnostics, another is using this for unrelated problems.

7.1 Other uses of min-cut

An obvious question is whether the same concept can be applied to other problems.

The characteristics leading to min-cut being a good fit for this problem are:

- Vertices can be partitioned into two parts where certain vertices (the sources and sinks) must be in certain partitions.
- Corrections correspond to removing edges.

If we consider the separation of variables into different clocks (and similarly for sub-clocks) we see that they sort of match the first, but not the second criteria:

- One Clock could be a selected as a source and another Clock-variables as sink. This works if there are two Clocks mixed together that should not be mixed, but if there are three or more Clocks mixed together this is not ideal (but at least it produces some diagnostics).
- However, removing an edge is not the only possible correction – another possibility is that the Clocks should be the same.

Clock-partitioning diagnostics should thus both include the possibility of separating the graphs, and also changing the Clocks to be the same, i.e., we can view it as two distinct cases (like for missing solverMethod or missing sample and hold).

For sub-clocks, the possibility of merging the clocks is even more complicated, since they can depend on multiple sub-clock factors.

Unrelated to synchronous models we believe this kind of diagnostic can be useful when breaking dependencies using `decouple` to allow parallelization – as described in (Elmqvist *et al*, 2014). It cannot directly help with `decouple` failing to split the system of equations into smaller part, but it can detect that the two sides of decouple are connected in unexpected ways, which has a tendency to occur.

7.2 Implementation

The algorithm was originally implemented Dymola 2019 (released in June 2018) and also in 3D Experience Platform 2019x. The handling related to solverMethod will be added in Dymola 2020, and all diagnostics are from that version.

Acknowledgements

The incorrect models from various users were one important aspect for starting this work; another inspiration to start the work was an effort to understand the existing algorithms for synchronous models developed by Sven Erik Mattsson (Elmqvist *et al*, 2012).

The proposal for a new primitive in MCP-0030 (Frenkel, 2018) gave a major inspiration to describe the approach.

Feedback from my colleagues and reviewers were helpful in making this paper clearer.

References

Peter Bunus, and Peter Fritzson (2002): Methods for Structural Analysis and Debugging of Modelica models. *Proceedings of the 2th International Modelica Conference.* 157-165.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest (1993): Introduction to Algorithms.

Hilding Elmqvist, Martin Otter, Sven Erik Mattsson (2012): Fundamentals of Synchronous Control in Modelica. *Proceedings of the 9th International Modelica Conference.* 15-26. doi: 10.3384/ecp1207615

Hilding Elmqvist, Sven Erik Mattsson, Hans Olsson (2014): Parallel Model Execution on Many Cores. *Proceedings of the 10th International Modelica Conference*. 363-370. doi:

10.3384/ECP14096363.

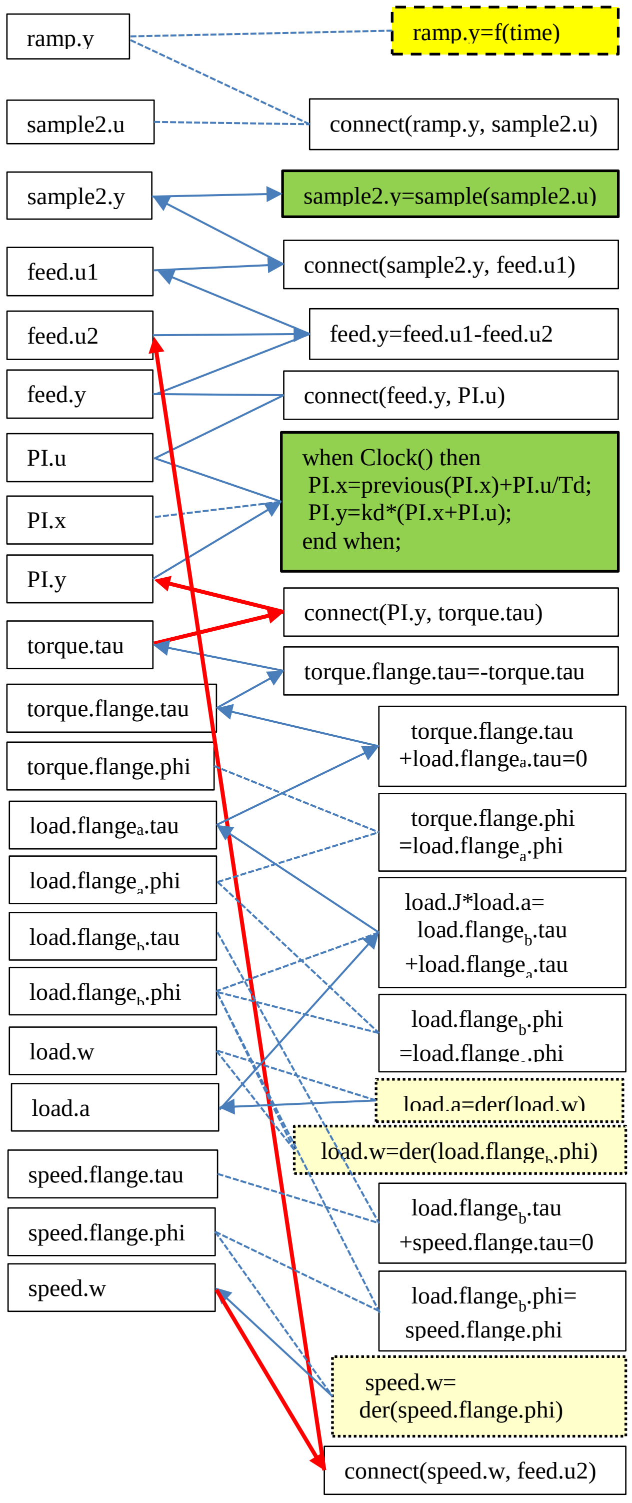
L. R. Ford, Jr and D. R. Fulkerson (1956): Maximal Flow Through a Network. *Canadian Journal of Mathematics* 8:399-404. doi:10.4153/CJM-1956-045-5.

Jens Frenkel (2018): Modelica Change Proposal MCP-0030 IsClocked Operator.

Hans Olsson, Martin Otter, Sven Erik Mattsson, Hilding Elmqvist (2008): Balanced Models in Modelica 3.0 for Increased Model Quality, *Proceedings of 6th International Modelica Conference*, vol. 1:21-33.

Hans Olsson (editor) (2017): Modelica A Unified Object-Oriented Language for Systems Modeling Language Specification Version 3.4.

8 Appendix



ramp.y
ramp.y=f(time)
sample2.u
connect(ramp.y, sample2.u)
sample2.y
sample2.y=sample(sample2.u)
feed.u1
connect(sample2.y, feed.u1)
feed.u2
feed.y=feed.u1-feed.u2
feed.y
connect(feed.y, PI.u)
PI.u
when Clock() then
PI.x=previous(PI.x)+PI.u/Td;
PI.y=kd*(PI.x+PI.u);
end when;
PI.x
PI.y
connect(PI.y, torque.tau)
torque.tau
torque.flange.tau=-torque.tau
torque.flange.tau
torque.flange.tau +load.flange$_a$.tau=0
torque.flange.phi
torque.flange.phi =load.flange$_a$.phi
load.flange$_a$.tau
load.flange$_a$.phi
load.J*load.a= load.flange$_b$.tau +load.flange$_a$.tau
load.flange$_b$.tau
load.flange$_b$.phi
load.flange$_b$.phi =load.flange$_a$.phi
load.w
load.a
load.a=der(load.w)
load.w=der(load.flange$_b$.phi)
speed.flange.tau
load.flange$_b$.tau +speed.flange.tau=0
speed.flange.phi
load.flange$_b$.phi= speed.flange.phi
speed.w
speed.w= der(speed.flange.phi)
connect(speed.w, feed.u2)

Figure 7 Flow network for advanced example

Study on Efficient Development of 1D CAE Models of Mechano-Electrical Products

Masatomo Inui[1] Tomohisa Fujinuma[2]

[1]Dept. of Mechanical Systems Engineering, Ibaraki University, Japan, masatomo.inui.az@vc.ibaraki.ac.jp
[2]Standardization Committee of New Digital Verification Technology, JEITA, Japan, tomohisa.fujinuma@toshiba.co.jp

Abstract

To promote the use of 1D CAE model in the mechano-electrical industry, it is necessary to resolve the issues associated with the model and reduce the cost of creating it. We are in the process of developing the guidelines for creating proper 1D CAE models that will help reduce the modeling cost. A mechano-electrical product is generally a complex system of mechanical, electrical/controlling, and software components. In the industry, Modelica and MATLAB/Simulink are emerging as popular tools for modeling the mechanical and electrical/controlling components, respectively. Programming languages derived from C are usually used for describing the software necessary in the mechano-electrical product. For example, SystemC is recognized as a standard tool for describing a hardware behavior in the design of electronic circuits to be incorporated in the product. In this study, we investigated a method for the combined use of these tools. We explain our findings in our experimental construction of 1D CAE models of a mechano-electrical product using Modelica, MATLAB/Simulink, and SystemC simultaneously.

Keywords: 1D CAE model, modeling guideline, Modelica, MATLAB/Simulink, SystemC, FMI

1 Introduction

Compact, high-precision, and high-performance mechano-electrical products such as multifunctional copiers, printers, and digital cameras are products of manufacturing industry in which Japan has demonstrated its excellence traditionally. While developing these products, high functionality and low price are to be ensured. Accordingly, technologies for supporting the design are considered to be the critical success factors in realizing an efficient and reliable product.

Figure 1 illustrates a typical design process for a high-tech mechano-electrical product. We divided the process into four stages: function and performance consideration, packaging and control design, system evaluation, and producibility evaluation. In this study, we focused on the first stage that determines most of the fundamental structure and parameters of the product. To realize a high functionality and a low price, it is important to utilize the computer simulations effectively in the design processes so that the feasibility of the functions is evaluated and the appropriate design options are narrowed down at an early stage.

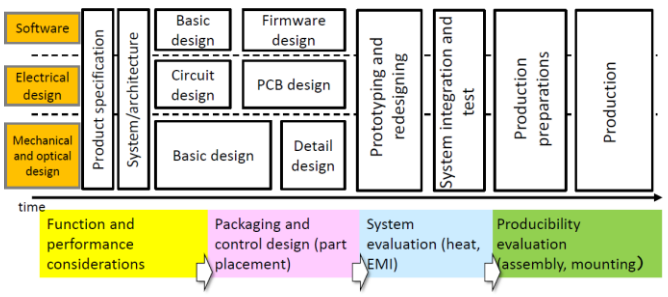

Software
Electrical design
Mechanical and optical design
Product specification
System/architecture
Basic design
Firmware design
Circuit design
PCB design
Basic design
Detail design
Prototyping and redesigning
System integration and test
Production preparations
Production
time
Function and performance considerations
Packaging and control design (part placement)
System evaluation (heat, EMI)
Producibility evaluation (assembly, mounting)

Figure 1. Typical design process of mechano-electrical products.

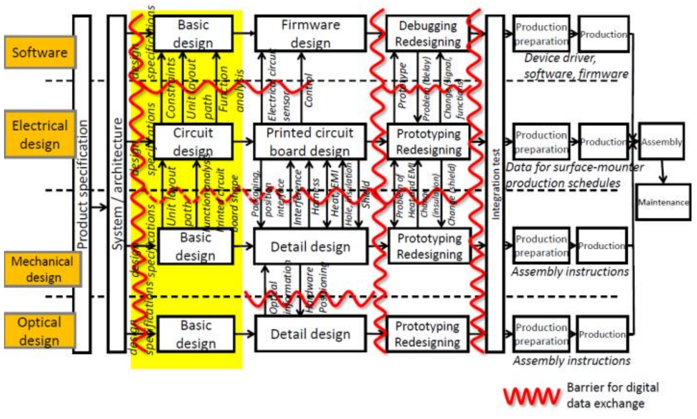


Figure 2. Barriers between design processes.

Considering that only a limited geometric information of the product is determined at an early stage of design, the existing CAE technologies are not suitable for simulations straightaway. Therefore, the Japanese manufactures prefer to develop their own original solutions such as an analysis software or a simulation model, and to use their proprietary knowledge base such as the original formulations, empirical formulae, and technology know-hows built over time. However, there are several practical issues in using these original systems especially in respect of their maintenance, integration with other CAD/CAE systems, and validation of their solutions.

Figure 2 illustrates the typical issues encountered by the Japanese mechano-electrical industry in interfacing the design processes. As shown in the figure, there are “barriers” to smooth digital data exchange between the design processes of the CAD and CAE systems. Therefore, many manufactures in Japan develop their own original interfaces and data sharing methods. However, this approach has a few inherent limitations particularly in respect of the distribution of data. Consequently, manual data exchange is still widely adopted in Japan.

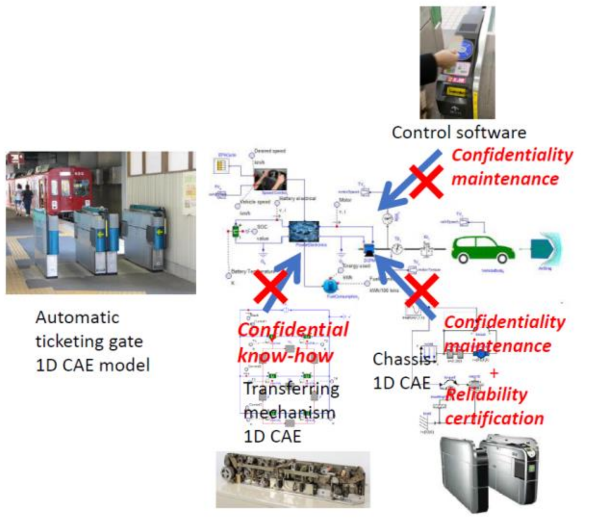

Control software
Confidentiality maintenance
Automatic ticketing gate 1D CAE model
Confidential know-how
Transferring mechanism 1D CAE
Chassis 1D CAE
Confidentiality maintenance
+
Reliability certification

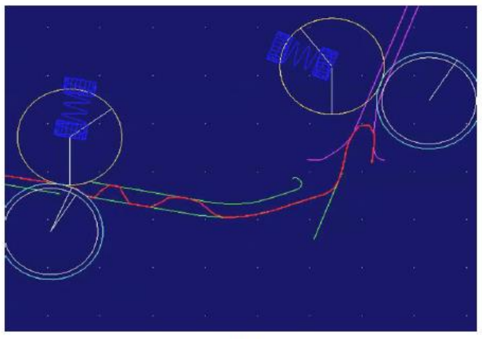
Figure 3. Problems in the joint development of a mechano-electrical product with multiple companies.

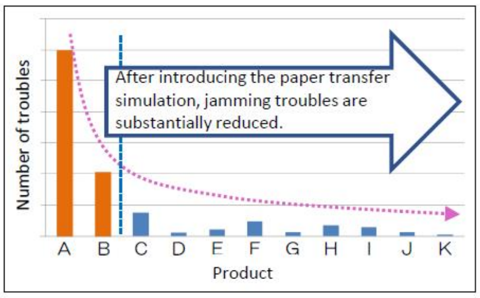
While development of a product jointly by multiple companies is becoming a common trend, there are critical issues in data exchange and co-simulations by multiple systems. Figure 3 illustrates the issues found in the joint development of an automatic ticketing gate in a railway station. In the case of a product developed by a single company, the parts are simply purchased, while their CAD models (electrical data) are required to determine the appropriate fixing and assembly methods. Nowadays, most products are developed by multiple companies jointly through collaboration. It is necessary that the engineering information (CAD and CAE data) of all the parts are available in the early stage of design to enable a comprehensive analysis of the product. In the case of the example, the major parts are the ticket transferring mechanism, chassis, and control software for automatic ticketing gate operation. Compatibility of the data and the models between the simulation systems is an essential factor to realize the collaborative product design successfully.

2 1D CAE for Mechano-Electrical Products

In Japan, we constituted a Standardization Committee of New Digital Verification Technology to establish a standard method for using the computer simulation technologies so that it can assist the design activities of a mechano-electrical product in its function and performance consideration stage.

In the design of automobiles and aircrafts, the product functions are rapidly advanced, resulting in issues similar to those related to a mechano-electrical product, and therefore a problem-solving method known as model-based development (MBD) is widely adopted. In MBD, the various conditions related to the requirements and functions of a product are defined by mathematical models. By evaluating the models, the product functions can be verified at the early design stages. Considering that simple analyses are often employed prior to determination of the 3D information, the MBD that is applied at the early functional design stage is specifically known as 1D CAE. Tools such as Modelica (Fritzson 2011) and MATLAB/Simulink (Tyagi 2012) have been used extensively in the automobile/aircraft industries. In these tools, the mathematical formulae related to the product functions are expressed as icons on the GUI of a computer, while the mathematical models for functional verification are represented by the interconnections.


(a)



(b)

Figure 4. Paper transfer simulation of a plain paper copying machine in early design stage.

The 1D CAE model is considered to be effective for assisting the functional design of mechano-electrical products. Figure 4 illustrates an example of MBD for the design process of a plain paper copying machine, which is a typical mechano-electrical product whereby it is critical to analyze the behavior of the paper while it is moving.

In earlier days, the paper behavior was checked by using a trial product. When a paper was jammed during the real-time use of a machine, the position and posture of the parts were slightly adjusted by guessing the paper behavior. Nowadays, the designers themselves analyze the paper behavior right at the initial stage of design by suitable paper transfer simulations (see Figure 4(a))(Hayakawa 2008). After introducing the paper transfer simulation, the jamming troubles are substantially reduced (see a graph in Figure 4 (b)).

As shown in this example, the 1D CAE model is effective in assisting the functional and performance design of the mechano-electrical products. Unlike the automobile and aircraft industries, the 1D CAE model is not a popular design method for mechano-electrical products. Some reasons for its limited use include the facts that:

- The development cycle of a mechano-electrical product is relatively brief, and therefore, the cost of preparing the mathematical model for 1D CAE represents a relatively large proportion of the total cost of the product.
- The basic structure of a mechano-electrical product changes rapidly, and therefore, the prior models cannot be used in new designs for obvious reasons. In other words, a new model needs to be created for every new product.
- The scale of business of a mechano-electrical product is relatively small, and therefore, it is not economical to train the engineers in 1D CAE that is a specialized subject.

To promote the use of 1D CAE in the mechano-electrical industry, it is necessary to resolve the issues associated with the use of 1D CAE as much as possible, and to reduce the cost of creating the model. We consider that the following two methods are effective in increasing the efficiency of creating a 1D CAE model.

1. **Development of modeling guidelines:** Creation of a 1D CAE simulation model is a complex task, and therefore, a trial and error process is indispensable. Accordingly, the cost of creating the model increases. To reduce the cost arising out of the trial and error process, we are developing guidelines for creating the 1D CAE models especially for mechano-electrical products. In the guidelines, the desirable steps in the modeling process as well as the important points to be noted in each step are mentioned. Accordingly, the guidelines help reduce the trials and thereby minimize the modeling cost.
2. **Clarification of important points in the combined use of Modelica, MATLAB/Simulink and SystemC:** A mechano-electrical product is generally a complex system comprising mechanical, electrical/controlling, and software components. In the industry, Modelica and MATLAB/Simulink are emerging as popular tools for modeling the mechanical and electrical/controlling components, respectively. Programming languages derived from C are usually used for describing the software necessary in the mechano-electrical product. For example, SystemC (Müller 2003) is recognized as a standard tool for describing a hardware behavior in the design of electronic circuits to be incorporated in the product. We are investigating to consolidate the important points to consider in the combined use of Modelica, MATLAB/Simulink, and SystemC.

In this paper, we explain our guidelines that facilitate efficient development of the mechano-electrical products. We also explain our findings in our experimental construction of the 1D CAE model of a mechano-electrical product using Modelica, MATLAB/Simulink, and SystemC simultaneously. In the next section, the guidelines developed by us are explained in detail. In Section 4, our findings on the combined use of Modelica, MATLAB/Simulink, and SystemC software are explained with an example. We expected that FMI/FMU (functional mock-up interface / unit) (Modelica Association 2014)(Hirano 2018) is a promising standard to connect the 1D CAE models and software components, however, the expected effect could not be realized in some cases due to some restrictions and issues. We summarize our conclusions in Section 5.

3 Guidelines for Creating 1D CAE Models of Mechano-Electrical Products

There are some researches are known on the guidelines of modeling. For example (Yazdani 2011) gives a guideline of modeling power system model. (Mathworks 2018) provides several guidelines for model construction using MATLAB/Simulink. In this paper, we explain our guideline for modeling behaviors of mechano-electrical products. Figure 5 presents an overview of our 1D CAE modeling guidelines. It depicts a flowchart of a typical model development process comprising 7 steps. In this chapter, we explain these steps briefly.

Step 1 Target selection: In this step, the design target is defined, and the function of the target product is clarified. It is desirable to perform functional development and reduce the function to physical models with proper input and output parameters. It is also desirable to establish communications with the designer in charge of the target product in advance so as to understand the solutions expected from the 1D CAE model.

Step 2 Modeling policy determination: In this step, the modeling level of the components is defined based on the results of the functional development. The functional specifications, scope of modeling, design parameters and their ranges, potential disturbance, and modeling accuracy are also formulated. The modeling method depends on the functional specifications of the model. For example, in the case of a gear train, it is adequate to model a simple gear ratio if the conveying rotation needs to be analyzed. However, if the purpose is the analysis of the vibration of the gear train, it is necessary to model the rigidity of the teeth of the gear.

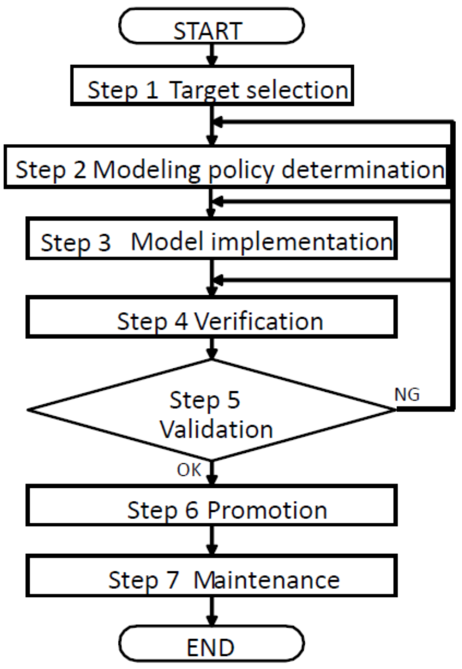


Figure 5. Flowchart for creating proper 1D CAE model.

Step 3 Model implementation: In this step, a model is implemented according to the modeling policy formulated. Modelica-based tools (OpenModelica or other commercial tools) or MATLAB/Simulink are the usually employed tools. Selection of the proper tool is critical in this step. Many Modelica-based 1D CAE systems such as Dymola and SimulationX are commercially available in the market. Each of these systems has its own unique characteristics. It is important that the most appropriate tool is selected according to the design target.

Step 4 Verification: The models constructed are verified in this step. There are two types of verification: operation and accuracy. The operation is verified by evaluating the following questions:

- Does the model involve physical movements of any of the parts during the operation?
- Does the model work properly with multiple design variables within their specified upper and lower limits?

Additionally, we strive to confirm that the implemented model exhibits a stable motion with errors within the specified limits with reference to the accuracy verification.

Step 5 Validation: In this step, the model of the complete subsystem is constructed by connecting the component models. The model is subsequently verified at the subsystem level by comparison with actual measurement values. The results are checked for the accuracy requirements formulated in Step 2. If they do not meet the requirements, then the Steps 2–4 are repeated to refine the model. The outcome is then compared with the actual machines, including the past models or experimental benches, to confirm the accuracy of the model. Finally, the PDCA (plan-do-check-act) cycles are executed to improve the accuracy of the model.

Step 6 Promotion: The models constructed are forwarded to the design department. To encourage the designers to use them, various promotional measures are necessary. These may typically include distribution of a usage manual, formulation of a report explaining the theoretical background of the model, accuracy reports, and other relevant documents. It is important that the designers use the model with confidence.

Step 7 Maintenance: When the model is deployed in the design, new demands emerge, such as expansion of functions, addition of new design parameters, and so on. To respond to the demands, it is essential that a model maintenance system with proper human resources is realized. It is also necessary to establish the rules for the control and reuse of the models.

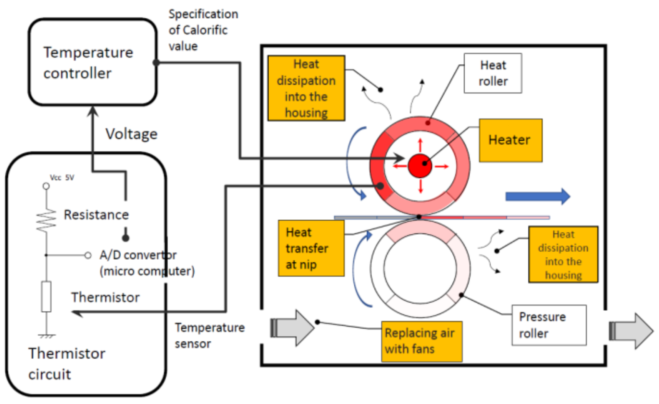


Figure 6. Simplified model of the image fixing unit of a plain paper copying machine.

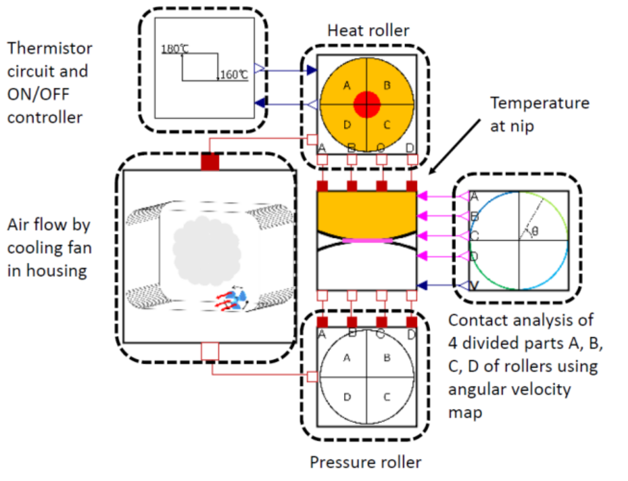

Thermistor circuit and ON/OFF controller
180℃
160℃
Heat roller
Temperature at nip
Air flow by cooling fan in housing
Contact analysis of 4 divided parts A, B, C, D of rollers using angular velocity map
Pressure roller

Figure 7. Temperature control model of the heat roller and the pressure roller.

To evaluate the applicability of the guidelines to the construction of a 1D CAE model, we conducted empirical studies on the modeling of typical components of a mechano-electrical product. In our test case, several components that simplified the functions of the plain paper copying machine were studied by employing OpenModelica (Open Source Modelica Consortium) as the modeling tool. We constructed the model by adhering to the method described in our guidelines as much as possible.

Figure 6 illustrates a block diagram of the implemented 1D CAE model. It shows a simplified mechanism of the image fixing unit of a copying machine (Ricoh 2018). To generate a stable image on a paper, the image fixing unit melts the toner by heating and pressurizing and fixes it on the paper surface. As shown in the figure, the heating and pressurizing are performed when the paper and toner pass between a heat roller and a pressure roller. The portion of the paper in contact with the heat roller and the pressure roller is called as nip. An electric heater is contained inside the heat roller. The temperature is controlled by a thermistor and an electric circuit so that the surface of the heat roller is maintained at a temperature within a limited range.

An analysis model of the temperature control is illustrated in Figure 7. This model consists of five components: a heat roller, a pressure roller, a paper, a thermistor circuit with a temperature controller, and a heat exhausting fan of the housing. The components are modeled by considering the design of the heating and pressurizing mechanism of actual copying machine. The heat generated by the heat roller is transmitted to the surface of the heat roller, the nip, and the pressure roller, in this order. In our model, the heat roller and the pressure roller were divided into four parts, respectively. In the rotation process of the rollers, the combination of the heat roller part and the pressure roller part in contact was calculated to realize the switching of the heat transfer model. In this model, heat transfer between the adjacent part of the heat roller and the pressure roller, and the heat dissipation into the air are also considered.

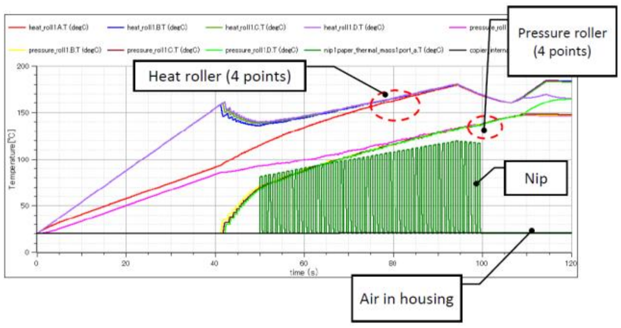


Figure 8. Analysis result of the temperature change at the nip.

Figure 8 shows the analysis results. We considered that the paper conveyance would be repeated from the start time to the end time of the copying process. The roller performs an acceleration motion from the start of the operation until the specified angular velocity is reached. After that, the rotation was continued with a constant angular velocity until the end of the operation. At the end of the operation, the roller performs an equiangular deceleration from the specified angular velocity. In our model, the temperature of the heat roller is correctly controlled, and the changes in the temperature at the nip is accurately reproduced before, during, and after the paper conveyance.

4 Combined Use of Modelica, MATLAB/Simulink and SystemC

In the design of a mechano-electrical product, an analog device was replaced with a digital device for promotional purposes of achieving higher functions of higher quality at low production cost. For similar reasons, the function realization method of the product was changed from a method using hardware to a method using software.

To determine a proper function realization method (analog or digital, hardware or software), the designer must execute the 1D CAE simulation of the product with the mechanical, electrical/controlling, and software components at an early stage of the design. In recent products, use of FPGA (Field-Programmable Gate Array) and custom LSI is emerging as a common trend. Since a longer time is necessary in the development of FPGA and custom LSI, it is necessary to start the function analysis with 1D CAE models earlier. The increased use of IoT technology is further accelerating this trend.

To start the collaboration of the mechano-electrical design and the software design earlier, a suitable method for the combined use of the 1D CAE models of the mechanical, electrical/controlling, and software components is required. For mechanical design, Modelica is emerging as a popular tool for creating 1D CAE models. On the other hand, most electrical as well as controlling engineers use MATLAB/Simulink as a modeling tool. For the software development related to a mechano-electrical product, SystemC is adopted as a standard language for describing a hardware of the electronic circuit to be incorporated in the mechano-electrical product. To realize the 1D CAE simulation of the total mechanism, the method of data sharing between the Modelica models, MATLAB/Simulink models, and SysmteC software must be standardized.

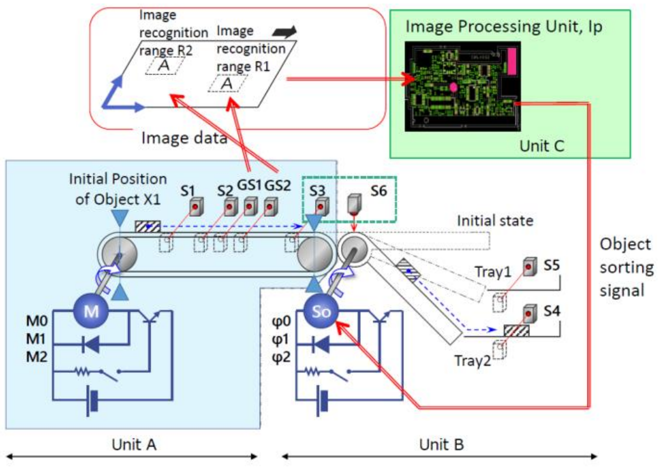


Image recognition range R2
Image recognition range R1
Image data
Image Processing Unit, Ip
Unit C
Initial Position of Object X1
S1
S2
GS1
GS2
S3
S6
Initial state
Object sorting signal
Tray1
S5
S4
Tray2
M0
M1
M2
φ0
φ1
φ2
Unit A
Unit B

Figure 9. A simple model of the paper transferring mechanism of a copying machine.

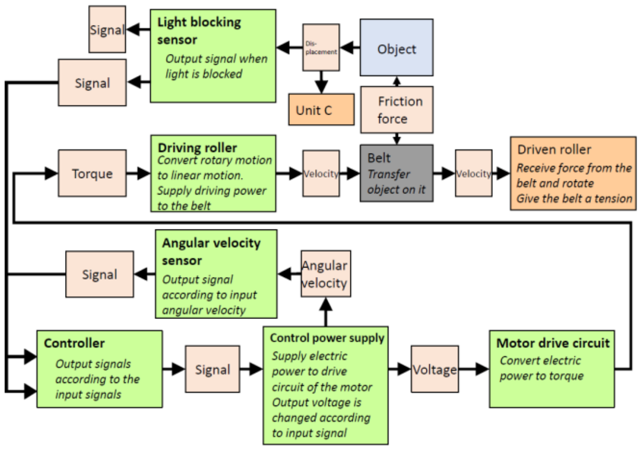

Signal
Light blocking sensor
Output signal when light is blocked
Dis-placement
Object
Unit C
Friction force
Torque
Driving roller
Convert rotary motion to linear motion. Supply driving power to the belt
Velocity
Belt
Transfer object on it
Driven roller
Receive force from the belt and rotate
Give the belt a tension
Angular velocity sensor
Output signal according to input angular velocity
Angular velocity
Controller
Output signals according to the input signals
Control power supply
Supply electric power to drive circuit of the motor
Output voltage is changed according to input signal
Voltage
Motor drive circuit
Convert electric power to torque

Figure 10. Block diagram of unit A (Object transferring mechanism with a belt).

We consider that a method based on the FMI standard is promising for sharing the data between simulation models using different implementation technologies. To clarify the problems in the data sharing using FMI, we performed several experiments implementing the 1D CAE models with Modelica, MATLAB/Simulink, and SystemC, and combined them using FMI for realizing the total simulation. Figure 9 illustrates a test case. This is a simplified model of a paper transferring mechanism of a copying machine. This model has three basic units that are denoted as unit A, unit B, and unit C.

Figure 10 shows a block diagram of unit A. This unit is an object (paper) transferring mechanism with a belt. This unit changes the motion as an object passes in front of the light blocking sensor S1, S2, and S3.

1. When the power is turned on, the driving motor rotates at speed M0.
2. When the object passes in front of the light blocking sensor at S1, the speed of the drive motor is switched from M0 to M1.
3. When the object is further conveyed and passed in front of the light blocking sensor at S2, the speed of the driving motor is switched from M1 to M2.
4. When the object is transported and passed in front of the light blocking sensor S3, the speed of the drive motor is switched from M2 to M0.

The unit C is a mechanism to identify the image of an object (paper). This mechanism has two devices GS1 and GS2 for capturing an image on the object. GS1 captures the image in a region R1 on the object. Another device GS2 obtains the image in a region R2 on the object. The images captured are processed by an image processing unit Ip. This unit outputs a signal according to the captured image. The output signal is used in the following object sorting operation.

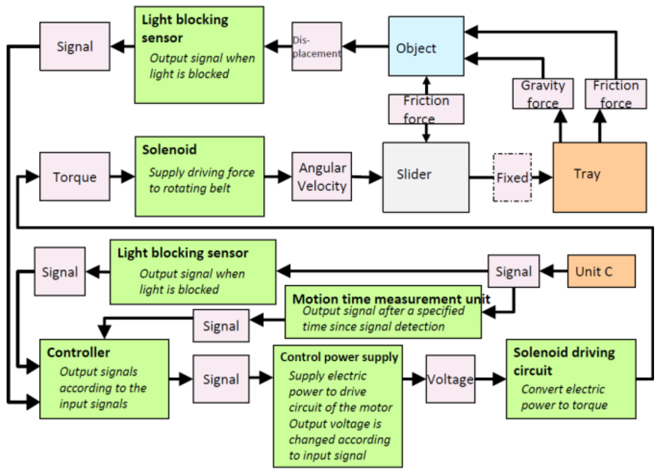

Signal
Light blocking sensor
Output signal when light is blocked
Dis-placement
Object
Friction force
Gravity force
Friction force
Torque
Solenoid
Supply driving force to rotating belt
Angular Velocity
Slider
Fixed
Tray
Light blocking sensor
Output signal when light is blocked
Signal
Unit C
Motion time measurement unit
Output signal after a specified time since signal detection
Controller
Output signals according to the input signals
Control power supply
Supply electric power to drive circuit of the motor
Output voltage is changed according to input signal
Voltage
Solenoid driving circuit
Convert electric power to torque

Figure 11. Block diagram of unit B (Object sorting mechanism).

Figure 11 illustrates a block diagram of unit B. This unit is a mechanism to sort the transferred object. It controls the inclination angle of a bridge according to the signal given by the unit C, and it sorts the object to tray1 or tray2. In the initial state, the bridge is in horizontal orientation.

- When the signal is 1, a solenoid S0 is rotated and the bridge is connected to tray1. When the sorted object is passed in front of the light blocking sensor

S5, the bridge is rotated back to the initial horizontal orientation.

- When the signal is 2, the bridge is connected to tray2. When the sorted object is passed in front of the light blocking sensor S4, the bridge is rotated back to the initial orientation.

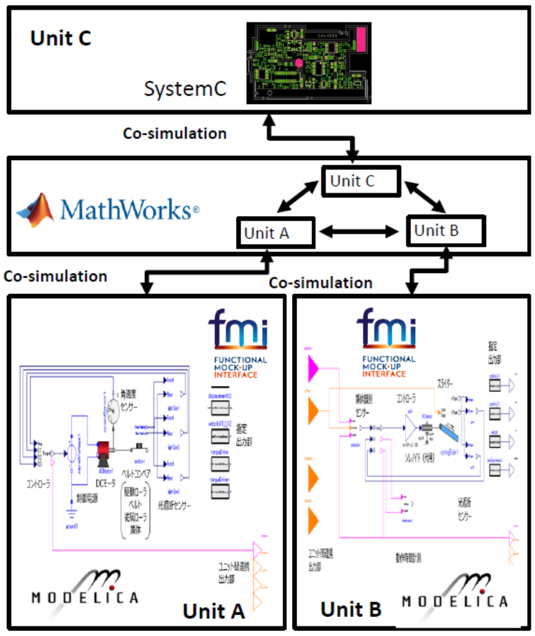


Figure 12. Combined use of Modelica, MATLAB/Simulink, and SystemC for implementing the paper transferring system.

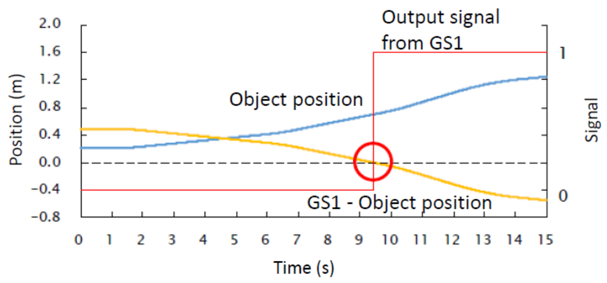


Figure 13. Analysis result of unit A.

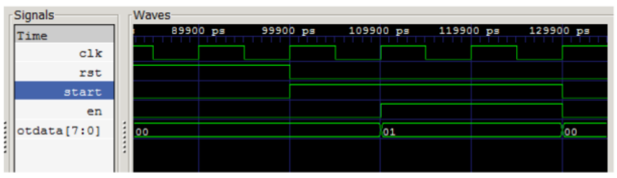


Figure 14. Analysis result of unit C.

In the implementation of the models, Modelica is used for modeling the unit A and unit B. Behavior of the electronic circuit of the image capturing and processing unit (unit C) is implemented using SystemC. The control softwares of the three units are realized using MATLAB/Simulink. Using the export function of FMI, unit A and unit B are incorporated into the control software. Since FMI is not supported by SystemC, the image capturing and processing software is integrated with the control software using the S-function of the MATLAB/Simulink. Figure 12 illustrates the relationships between Modelica, MATLAB/Simulink, and SystemC in our implementation.

By using the combined models, the engineers can analyze the effect of the change of the design parameters of the units on the behavior of the mechanism. The following three issues were detected in the verification result of our model.

1. **Management of the simulation step time:** Figure 13 shows the analysis result of the unit A. The three curves in the graph represent the object displacement (blue curve), distance between image sensor GS1 and object (yellow curve), and signal output from GS1 (red curve). Figure 14 shows the simulation result of the operation of the unit C. As shown in the two graphs, the time step of the simulations is much different between the unit A (time step is in s) and unit C (ns). When the total simulation is executed following the time step of the unit A, the high frequency behavior of the unit C cannot be detected. The simulation of the unit A must be repeated a number of times when the total simulation is executed according to the time step of the unit C. The latter case causes an increase in the computation time and the accumulation of the computation errors. In the co-simulation of the mechanical system (unit A) and the electronic circuit (unit C), the designer must pay attention to the validity of the time step in the simulation. Similar problem was discussed in (Centomo 2016).

2. Signal transmission between the models: In the control process of the paper transferring mechanism, various parameters of the physical quantities are exchanged between the components and the FMUs. In addition to the physical quantities, the signals that trigger the events are also exchanged. For example, the start and end of the operation of the units in our model are triggered by mutual sending and receiving of the signals. In our experiments, the signals were not successfully transmitted between the software components written in SystemC and the controlling unit performed by MATLAB/Simulink. This problem is expected to be resolved by incorporating the SystemC unit into the controlling unit by using FMI.

3. Selection of proper design parameters: The central purpose of the combined simulation is to determine the optimal design parameters by executing various simulations with different design parameters. In the design of the electrical/electronic system, the clock frequency is usually selected as a design parameter in the time domain. On the other hand, the processing time is usually adopted as a design parameter of the mechanical system. In our demonstration of the paper transferring model, the processing time is selected as a common parameter in the time domain. The design parameter of the electrical/electronic system is given by converting the processing time to the time steps counted in the clock frequency. An incorrect selection of the design parameters would necessitate cumbersome parameter conversions and thereby affect the work efficiency in the simulation. Therefore, it is necessary to the select the design parameters carefully.

5 Conclusions

To realize an efficient design of a mechano-electrical product, it is important to utilize the 1D CAE simulations effectively in the early functional and performance consideration stage. In the automobile and aerospace industry, the use of the 1D CAE model is already a common practice. The model is considered to be effective for assisting the functional and performance design of the mechano-electrical products; however, it has not been adopted as a design method for these products in the past.

To promote the use of the 1D CAE model in the mechano-electrical industry, it is necessary to resolve the issues associated with the use of the model as much as possible. In this paper, we propose the guidelines for creating a proper 1D CAE model. In the guidelines, the desirable steps in the modeling process as well as the important points to be noted in each step are mentioned. By following the guidelines systematically, we can minimize the modeling cost.

In the 1D CAE simulation of a mechano-electrical product, Modelica, MATLAB/Simulink, and SystemC are used as the modeling tools. We investigated to consolidate the important points to consider in the combined use of these three tools. Through the development of simplified models of the plain paper copying machine, we found that there are three critical issues in their combined use: 1) management of simulation step time, 2) signal transmission between the models, and 3) selection of proper design parameters.

It is a difficult task to quantify the effectiveness of the developed guidelines and the points learnt in the combined use of Modelica, MATLAB/Simulink, and SystemC, but we strongly believe that these results are helpful in creating the models without mistakes. We plan to distribute our research results to the member companies of the Standardization Committee of New Digital Verification Technology so as to evaluate its applicability thoroughly.

References

Centomo, S., Deantoni, J., De Simone, R., “Using SystemC Cyber Models in an FMI Co-Simulation Environment: Results and Proposed FMI Enhancements,” Proc. 2016 Euromicro Conference on Digital System Design (DSD), 2016. DOI: 10.1109/DSD.2016.86

Fritzson, P., “Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica,” Wiley-IEEE Press, 2011.

Hayakawa, S., et al., “CAE Technology for Mechanical Design,” Technical Report 18, Fuji Xerox Co., Ltd, 2008 (in Japanese)

Hirano, Y., “Toward the model exchange by using FMI (Functional Mockup Interface) JSAE WG Activities,” Proc. FMI FORUM 2018 in Japan, 2018.

Mathworks, “Modeling Guidelines”, https://jp.mathworks.com/help/simulink/modeling-guidelines.html?lang=en.

Modelica Association, “FMI Version 2.0, FMI for Model Exchange and Co-Simulation,” https://fmi-standard.org/downloads/ 2014.

Müller, W., Rosenstiel, W., Ruf, J. (Eds.), “SystemC, Methodologies and Applications”, Springer, 2003.

Open Source Modelica Consortium, “OpenModelica User’s Guide, Release v1.13.0-dev-493-gddc51eb.”

Ricoh Company, Ltd, https://jp.ricoh.com/kouken/science_caravan/QandA/science/qanda2_18.html, reviewed in Nov. 3, 2018. (in Japanese)

Tyagi, A. K., “MATLAB and Simulink for Engineers,” Oxford University Press, 2012.

Yazdani, A. et al. “Modeling Guidelines and a Benchmark for Power System Simulation Studies of Three-Phase Single-Stage Photovoltaic Systems,” Proc. IEEE Transactions on Power Delivery, Vol. 26, Issue 2, April 2011, pp. 1247-1264.

Advanced Modeling of Electric Components in Integrated Energy Systems with the TransiEnt Library

Jan-Peter Heckel Christian Becker

Institute of Electrical Power and Energy Technology, Hamburg University of Technology, Harburger Schloßstraße 20, 21079 Hamburg, Germany, {jan.heckel,c.becker}@tuhh.de

Abstract

Integrated Energy Systems (IES) with the coupling of electricity, gas and heat are assumed to be a suitable concept for future energy systems. For the necessary energy system analysis with respect to dynamics and stability, powerful tools are needed. Such tools should be provided in open toolboxes to make the research more transparent, comprehensive and communicative. It is aimed that scientists collaborate on models for multimodal energy system analysis.

The dynamic simulation is a method that allows to consider transient, non-linear effects and controller design. The TransiEnt Library, developed and established at Hamburg University of Technology (TUHH), offers such a toolbox. Previous versions of the library worked with limited electrical models. In this paper, the extension of the TransiEnt Library with new models, that allow to consider stability effects of the electric grid, is described. To do so, features from Modelica® and its specification are used to create an advanced framework for the electrical part of IES. As a result, IES with interconnected electric grids can be modeled and simulated more precisely.

Keywords: Integrated Energy System, Electric Energy System, Load Flow Calculation, Frequency Stability, Voltage Stability, Renewable Energy

1 Introduction

The TransiEnt Library (Hamburg University of Technology, 2018) is a Modelica® (Modelica Association 2017) library for the simulation of an IES with high share of renewable energies. The complete library is open-source. The TransiEnt Library was developed within the project TransiEnt.EE (BMWi 03ET4003) as a fundamentally new approach for modeling IES with the method of dynamic simulation. TransiEnt.EE was completed in 2017 and the successor project ResilientEE (BMWi 03ET4048) has been started subsequently. The TransiEnt Library is implemented in the simulation environment Dymola® (Dassault Systèmes, 2018). Within the Transient.EE project the main focus of the entire system modeling was mainly put on heat and gas system representation with a reduced scope of the electrical system part. This paper presents the extension of the TransientEE Library by increased modeling and simulation capabilities concerning dynamic and stability phenomena of meshed electrical grids.

The paper starts with an illustration of the political, technical and organizational background for IES. This includes the goals and already existing results of the two projects mentioned above. The new models' purpose and their level of detail is explained. In section 2, the development and implementation of the new electrical models are introduced and explained in detail. Section 3 contains three implemented example models including the new features. The paper closes with a conclusion and an outlook to further research work.

1.1 Energy Transition and Integrated Energy Systems

The motivation of the projects TransiEnt.EE and ResiliEntEE is the persisting transition of the German energy supply known as Energy Transition. It is planned to increase the share of renewable energies in the next decades in order to decrease the carbon dioxide emissions. Furthermore, the nuclear power phase-out was decided after the Fukushima incident in 2011. In the electrical sector, the renewable energies should have a share of at least 80 % in 2050 (German Federal Ministry of Justice und Consumer Protection, 2017).

The fluctuating generation of the renewable energies must be balanced with the volatile consumption. This is only possible by using storage technology. Electric storages are considered, but their power output and storage capacities are limited and costs per kWh energy storage are high. Hence, the idea of IES to couple the sectors electricity, gas and heat is considered. The coupling is performed by primary coupling technologies. These technologies are for example Power-to-Gas (PtG), Combined-Heat-and-Power generation plants (CHP), Combined Cycle Gas Turbine (CCGT), Fuel Cells (FC) but also technologies like gas boilers, gas turbines (GT), electric boilers, known as Power-to-Heat technology. Figure 1 shows an example of an IES with different technologies.

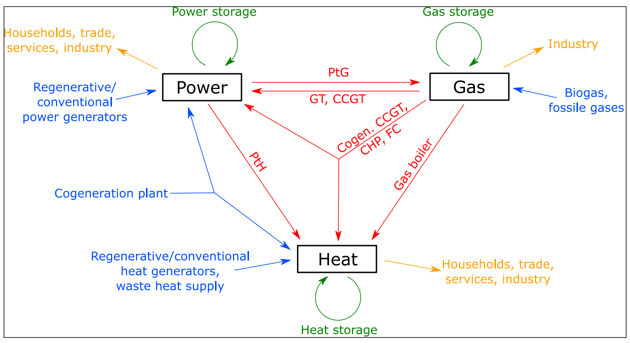

Power storage
Gas storage
Households, trade, services, industry
Industry
Power
Gas
PtG
GT, CCGT
Regenerative/ conventional power generators
Biogas, fossile gases
Cogen, CCGT, CHP, FC
PtH
Gas boiler
Cogeneration plant
Regenerative/conventional heat generators, waste heat supply
Heat
Households, trade, services, industry
Heat storage

Figure 1. The three sectors with examples of primary coupling technology (red), generation technology (blue), storage technology (green) and civil sectors (yellow)

By multimodal sector coupling, the storage possibilities in the non-electrical sectors can be used for compensating the imbalance between renewable energy generation and consumption. Gas and heat grids themselves are assumed as distributed storages because they can buffer failures much longer than the electric grid. Gas and thermal storage technology are considered to have higher energy capacity, power output and lower costs per kWh energy storage than electric storages.

Nevertheless, future IES must be investigated in a systematic way to estimate their opportunities and risks.

1.2 Project TransiEnt.EE

The project TransiEnt.EE investigated innovative possibilities for an efficient energy supply by integrating renewable energies in the existing grid infrastructure. For this, the IES of the city of Hamburg was regarded. The main innovation was the development and use of dynamic simulation means for the entire energy system analysis. Dynamic simulations facilitated to investigate non-linear behavior of components, control strategies and system stability.

It was shown that the sector coupling increases the flexibility of the energy supply. The gained flexibility allows a higher share of renewable energies (Andresen *et al*, 2017).

1.3 Project ResiliEntEE

In the current ResilientEE project, the scope of IES analysis and design is extended to Northern Germany. With this new example region, the topologies of the electric and the gas grid come into the focus of study to a much higher level of detail. The stability of the electric grid coupled with subsystems of the energy sectors and the compliance of the demanded limits for characteristic values are to be investigated.

Finally, the resilience of the IES should be benchmarked. For this benchmark, it is planned to introduce a quantified figure to compare different scenarios. From this analysis, suggestions for further improvement, i. e. an increase of the system's resilience, should be developed. Three predefined scenarios should be regarded within the project: first, a centralized scenario of larger power plants and a centralized control scheme, secondly, a decentralized scenario and thirdly a scenario according to the German grid development plan (*Netzentwicklungsplan*) 2035.

1.4 Purpose of Models

In the frame of the TransiEnt.EE project, the electric grid is modeled as a copper plate. Therefore, the dynamic representation is limited to time-dependent active power balance and system frequency as well as power-frequency-control (Andresen *et al*, 2015). Furthermore, the TransiEnt Library offers models for unmeshed grids without branches. These models limit the possibilities of electric grid modeling. Standard methods for electric power systems like load flow calculations, angle stability and voltage stability examination are not possible with these models. The modeling approach does not allow to compute voltages and reactive power values.

Hence, it is necessary to develop and implement extended model features that fit into the existing framework of the TransiEnt Library. Components from already existing Modelica Libraries for the modeling of electric power systems are not compatible with this framework. Detailed models for gas- and heat-processes already exist in the TransiEnt Library. The extension for the electric grid empowers the TransiEnt Library to become a complete and unique toolbox for universal steady-state and dynamic energy system modeling and analysis.

The use of Modelica has many advantages even for the new electrical models. Equations and system structures can be implemented directly. Connectors permit interfaces between sectors and their components and combine the components mathematically to one model.

The main advantage Modelica offers in this context is its modularity. The modularity allows to replace components easily. New features and models can be added by different editors in a simple way.

The TransiEnt Library is designed for energy system analysis performed by energy engineers and users in the energy and power economy. Hence, the electric component models are required to be easy to use for those with different backgrounds as well.

1.5 Level of Detail

In complex, dynamic models of energy systems, dynamics of the technologies from different energy sectors are coupled in order to cover their interaction. In general, the electric part of the coupled system has higher dynamics than the processes in the gas and heat sector. Consequently, the risk of a stiff problem occurs. To deal with this risk, models are created in different levels of detail for different time scales which can be replaced by each other.

In the ResiliEntEE project, systemic concepts for model reduction have to be used. Furthermore, it is deemed to be reasonable to consider only dynamics with time constants above 1 s in order to avoid stiffness.

2 Implementation of New Models

The new models represent interconnected grids with components such as generators, transmission lines and transformers. The basis for this modeling is the complex, quasi-stationary calculation of alternating current networks. The main part of electric grid modeling is the connector which is described in the following paragraph before the components will be introduced.

2.1 New Connector Complex Power Port

In the electrical part of the energy system, connectors called Power Ports are used to link several components with each other. In the TransiEnt Library, there are three Power Ports with different levels of detail.

- Active Power Port
- Apparent Power Port
- Complex Power Port

The three Power Ports only regard one phase of the three-phase system assuming this to be symmetric. The Active Power Port is the connector developed and used in the TransiEnt.EE project. Because it just contains active power and frequency, this port can only be used for electric grids modeled as copper plates. The Apparent Power Port is capable of modeling grids without branches, arranged as one line. This is made possible by adding voltage and reactive power to the connector. For calculating the partition of power at branches, another electrical quantity is necessary. The Complex Power Port therefore adds the angle of the complex voltage phasor to the connector quantities.

When regarding the Modelica specification, it becomes clear that the Complex Power Port is overdetermined by one quantity. There should not be more than one potential variable per flow variable. In this case, there are two flow variables, the active and the reactive power, and three potential variables, voltage magnitude, voltage angle and the frequency. The two quantities of the complex voltage can be assigned to the two power flows which all together build the complex representation of the electric network. But the frequency is an additional potential variable which cannot be assigned to a flow variable. Physically, the frequency is not an independent quantity. The frequency is part of the models because of the quasi-stationary approach. Additionally, the frequency is part of the Complex Power Port and hence, an available information in all parts of the electric grid in this modeling approach. Within the model approach for the ResiliEntEE project, it is intended to only have one grid frequency. This is a justified simplification for the grid model of Northern Germany as it is strongly meshed. For larger networks, different local frequencies must be considered to take care of power oscillations. In section 2.3, it is discussed how to extend the concept to more than one frequency.

To avoid overdetermined Differential Algebraic Equation Systems (DAE) in interconnected electric grids, the procedure mentioned in the Modelica Specification section 9.4 is followed. With this procedure, the Complex Power Port is defined as follows.

```
// Potential variables ("technical
connection conditions")
  Modelica.SIunits.Angle delta "Voltage
Angle";
  Modelica.SIunits.Voltage v "Voltage of
grid";
  TransiEnt.Basics.Units.Frequency2 f
"Frequency of net";

  // Flow variables ("Transmitted
information")
  flow Modelica.SIunits.ActivePower P
"Active Power in Connector";
  flow Modelica.SIunits.ReactivePower Q
"Reactive Power";
```

The Type `Frequency2` has the function “equalityConstraint” as defined in the specification. The algorithm is:

```
algorithm
  assert(f1-f2<1e-10,"Equal frequencies");
```

With this connector, electric grids can be connected via one unipolar connector. This allows the well-known unipolar representation of electric power systems which facilitates the work with the components for users with different backgrounds.

2.2 Steady-State Components

Stationary busses for single load flow calculations and components for linking busses are available in the TransiEnt Library as steady-state components. Stationary busses are used as boundaries for different quantities from the Complex Power Port. In electrical power engineering, three kinds of stationary busses are distinguished.

- Slack Bus: Voltage magnitude and angle are fixed; 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.
- PQ Bus: Active and reactive power are fixed; other quantities are calculated.

The Slack Bus has the following code that defines it as the root of the connection graph of Complex Power Port’s frequency.

```
Connections.root(epp.f);
```

The PV Bus is defined as `Connections.potentialRoot` for the frequency of the

Complex Power Port. These two functions are part of the Modelica language. Slack and PV busses are separate boundary models. For a PQ Bus, it is possible to either choose a simple boundary model or the load model with voltage and frequency dependency of active and reactive power. The frequency is part of the Complex Power Port due to the frequency dependency of the load.

When performing load flow calculations, the nodal voltages are determined. Because unknown quantities are initialized by zero, all grid busses need to be represented by one of these bus models. Even if no generation or consumption is connected to a bus, a boundary model must be chosen in order to prevent division by zero and to speed up the initialization.

The main linking component is the transmission line. Transmission lines are modeled using the Pi-network representation which is valid for electrically short transmission lines up to a length of approximately 150 km.

For modeling the behavior of a transmission line, the n-port representation is a powerful way to build its equations. When using admittance parameters in form of an admittance matrix $[\underline{Y}]$, the transmission line with its physical parameters Resistance R, Inductance L, Capacitance C and angular frequency ω can be represented by the following equations.

$$\begin{pmatrix} \underline{I_1} \\ \underline{I_2} \end{pmatrix} = [\underline{Y}] \cdot \begin{pmatrix} \underline{V_1} \\ \underline{V_2} \end{pmatrix} \qquad (1)$$

$$[\underline{Y}] = \begin{bmatrix} \frac{1}{R + j\omega L} + \frac{j\omega C}{2} & \frac{-1}{R + j\omega L} \\ \frac{-1}{R + j\omega L} & \frac{1}{R + j\omega L} + \frac{j\omega C}{2} \end{bmatrix} \qquad (2)$$

$$\underline{S_i} = P_i + jQ_i = \underline{V_i} \cdot \underline{I_i}^* \qquad (3)$$

The port *i* has the voltage $\underline{V_i}$, the current $\underline{I_i}$, the complex power $\underline{S_i}$, the active power P_i and the reactive power Q_i. The power quantities are balanced powers. In the TransiEnt Library, the convention is that inflowing powers are positive and outflowing powers are negative. Consequently, generated power is assumed to be negative and consumed power to be positive at electrical connectors. With these equations, it is possible to describe the transmission line in a numerically efficient way that allows modeling of networks with up to hundreds of busses. Transmission lines can be parametrized by given values from literature or customized values. The TransiEnt Library additionally offers transformer models. Hence, electric networks with several voltage levels can be modeled and simulated.

Because of the overdetermined connector mentioned in section 2.1, the linking components must be defined as *Connections.branch* for the frequency of the Complex Power Port for the elimination of the surplus equation.

The load flow calculation function which has been implemented in the TransiEnt Library has been successfully validated with the commercial software tool NEPLAN® (ABB, 2016).

2.3 Dynamic Components

The dominant dynamic components in electric grids that are not fed by inverters are synchronous generators. Their behavior can be modeled in different levels of detail. Regarding the Complex Power Port, the simplest way to model a generator is to couple the active power with the frequency. The voltage is usually controlled and therefore set to a fixed desired value and the reactive power is calculated. The dynamic behavior is represented by the mechanical swing equation.

$$\ddot{\phi} = \dot{\omega} = \frac{\omega_0}{T_A} \cdot \frac{P_{mech} - P_{el}}{S_{rG}} \qquad (4)$$

The generator and the turbine rotate at the angular frequency ω that has the nominal value ω_0. The angle ϕ denotes the machine's rotor wheel angle and the inertia is described by the time constant T_A. The generator which has the rated apparent power S_{rG} is driven by the mechanical power P_{mech} and generates the electric power P_{el}. In the TransiEnt Library, the inertia of the generator and turbine is described with an additional component model for an inertia. Generator models are offered in a slack as well as in a PV bus version. The slack version fixes the voltage angle and therefore the voltage angle reference for the quasi-stationary consideration of the electric grid. The user can switch between both versions by setting a Boolean parameter.

For a higher level of detail, the coupling between the generator voltage magnitude and the active as well as reactive power is considered. In general, synchronous generators are typically modeled according to the Two Axis Method (Arrillaga Arnold 1990, Milano 2010). With the Two Axis Method, it is possible to describe the non-symmetric design of the magnet wheel. Only cylindrical rotor machines, which underlie slow dynamic changes, can be modeled using the synchronous reactance of just the direct axis which corresponds to a single-phase schematic as shown in Figure 2.

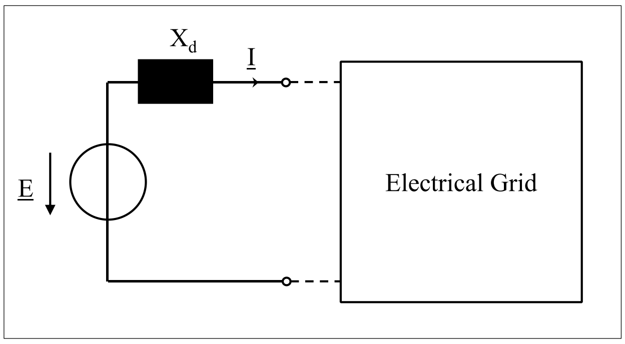


Figure 2. Steady-state model of synchronous generator as electric circuit with alternating voltage source $\underline{E}$ and synchronous reactance X_d.

The Two Axis Method is based on the Park Transformation and not explained in detail (Park, 1929).

Models with this coupling between voltage magnitude and the two forms of electric power need a voltage controller for operation in electric grids. Applied to the model in Figure 2, the voltage controller determines a value for the excitation current which leads to a voltage $\underline{E}$ to obtain the desired nominal voltage at the generator terminals. Models for voltage controllers are combined with excitation system models. These combined models are mainly linear time-invariant (LTI) systems represented as transfer functions in feedback loops. The Institute of Electrical and Electronics Engineers (IEEE) recommends different types of such models (IEEE, 2016). The TransiEnt Library offers DC and AC excitation system models in accordance with the IEEE recommendations.

Although the simulations in the ResiliEntEE project will be done with a single overall system frequency, the TransiEnt Library has the feature to compute different frequencies for generators. It is possible to activate a distinct frequency for every PV bus generator by a Boolean parameter. This feature is implemented by the following equation.

$$\frac{d\,\theta_{pv}}{dt} = \omega_{pv} - \omega_{slack} \qquad (5)$$

The slack generator operates with the angular frequency ω_{slack} while the PV bus generator operates with ω_{pv} and has the polar wheel angle θ_{pv}. The polar wheel angle of the slack bus is set to zero when using more than one frequency. Different frequencies in electric networks allow differently oscillating generators and thus power oscillations in large electric networks. Models for Power System Stabilizers (PSS) to damp power oscillations can be implemented when using this feature. With the consideration of more than one frequency, it becomes necessary to determine a frequency that is used as the overall grid frequency at different places in the network for dynamic loads and other models instead of the slack generator's frequency. This problem has not yet been faced by the TransiEnt Library. Currently, this topic is worked at TU Ilmenau by combining phasor-based and electromagnetic transient (EMT) simulation to a hybrid simulation model which can be assumed as an approach for this problem (Jiang *et al*, 2018).

In existing models from the TransiEnt Library like power plant models, Modelica's modularity allows to define generator models, power boundaries and connectors as replaceable components. This allows the reuse of proven models from the non-electrical sectors. In addition, the replaceable components permit a variation of the model's level of detail.

3 Simulated Examples

To demonstrate the new features of the TransiEnt Library, some example system models are presented in this section.

3.1 Steady-State Example

First, there is a steady-state example for demonstrating the performance of load flow calculations. A test grid for Northern Germany has been created and implemented for performance analyses.

This Modelica model consists of three submodels. One model represents the electric high voltage grid of Mecklenburg-West Pomerania, another one Schleswig-Holstein and Hamburg. The third sub model contains the high voltage grid of Lower Saxony and Bremen. Using the transformer models mentioned in section 2.2, it is possible to represent the three voltage levels 380 kV, 220 kV and 110 kV. The topology is established manually from data published by the Transmission System Operators (TSO) (TenneT TSO GmbH, 2018; 50Hertz Transmission GmbH, 2018). The model is only capable of representing one switching status of the real grid. Figure 3 shows the Schleswig-Holstein and Hamburg part of the model.

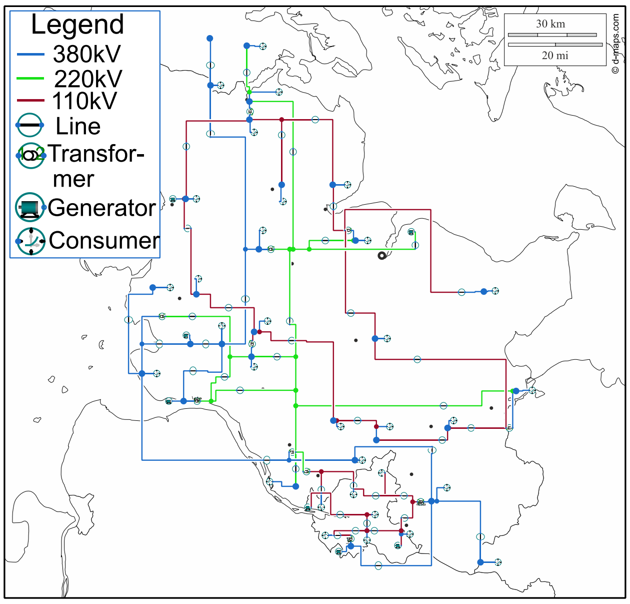


Figure 3. Model of the electric power grid of Schleswig-Holstein and Hamburg in Dymola (d-maps, 2018).

Three scenarios are considered. The first scenario is the hypothetical operation of the Northern German grid as an electric island grid. The second scenario represents power transit from the direction of Denmark into the direction of the Netherlands. The third scenario is the so called *Dunkelflaute* scenario that assumes minimal wind and photovoltaic generation as well has high consumption. Especially, the third scenario shows a high use of the transmission line capacities which can be seen as an indication of congestion. Detailed results can be provided by the authors on request.

3.2 Dynamic Example — Only Electrical

The next model shows how dynamic simulations of electric power systems can be performed with the TransiEnt Library. This example covers only electric and some necessary mechanical components. It is purely theoretical and represents a grid with four generators supplying a load variation in active and reactive power. The example is used as a reference model in the TransiEnt Library for the test of the component models. The model size and the short time horizon allows to choose the most advanced generator model with Two Axis description and differential equations describing the magnetic effects. This does not increase the computational time significantly in comparison to generator models with more simplified equations. Figure 4 shows an excerpt of this simulation model. The turbine is modeled as a first order time delay and a mechanical boundary. The turbine is actuated by a linear system of the same type which represents the measurement delay and the proportional controller. The inertia of the whole rotating system is represented in an inertia model. The generator model is coupled to the electric grid by a transmission line and a transformer. A standard DC voltage controller is used. The bus between transmission line and transformer is initialized by a power boundary.

Figure 5 is an exemplary plot from this simulation and shows the reaction of the overall grid frequency to a load step. It shows a decrease in frequency because only primary control (proportional control) and no secondary control (integral control) is applied in this example. The bus voltages in this example are also variable and an example bus voltage on the 380 kV level shows the time behavior in Figure 6. The voltage controller neither has an integral behavior. Consequently, the nominal voltage

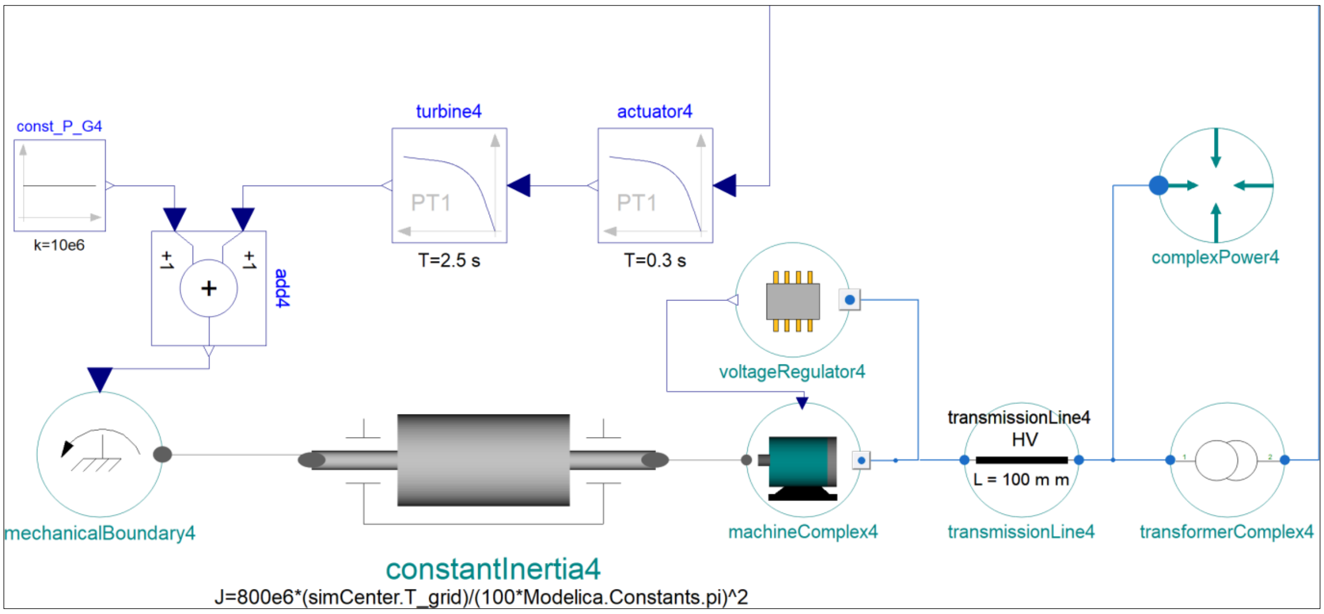


Figure 4. Excerpt of dynamic simulation model with mainly electric components.

cannot be reached in steady-state before and after the load step.

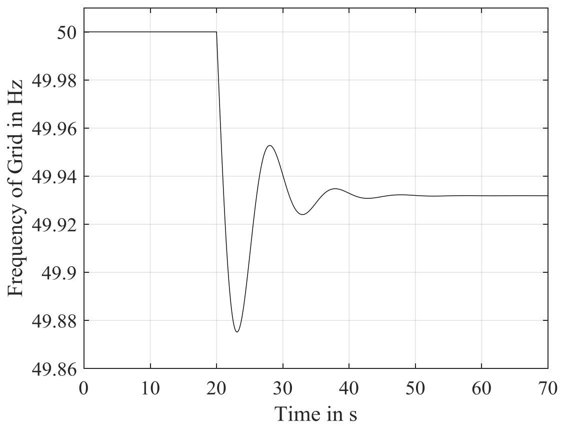


Figure 5. Time plot of grid frequency in dynamic example model.

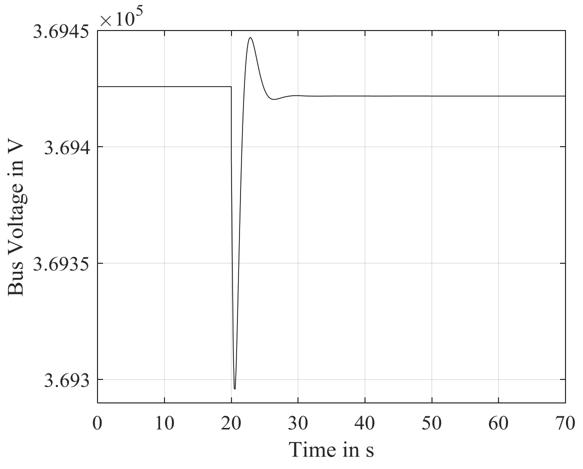


Figure 6. Time plot of bus voltage on the 380 kV level.

3.3 Dynamic Example — IES

The third example should give an outlook to further planned simulations within the ResiliEntEE project. In these simulations, the components of all three sectors electricity, gas and heat constitute one simulation model that is simulated without the drawbacks of Co-simulation. Co-simulations typically need interfaces between different partial simulations. These interfaces handicap physical constraints such as mass and energy conservation and reduce the numerical efficiency.

Figure 7 shows the implemented dynamic system model. The model consists of two conventional power plants, a wind park as a renewable generation plant and a CHP plant. A natural gas grid is also part of the simulation model. For the heat sector, the CHP plant supplies a heat consumer. The feed-in-station feeds hydrogen into the gas grid when an excess of renewable energy generation in the right subpart of the model occurs. A battery storage is used to demonstrate some flexibility and is implemented as a voltage controlled device which sets the bus voltage to a fixed value. This sufficiently models the behavior of a power electronic inverter for the given time resolution.

When performing small parameter-variations, the overall system behavior changes noticeably, showing the high non-linearity of the system model. The model shows that a suitable operation management is necessary for IES. Especially, the storages must be controlled to have a beneficial effect on the electric grid. It can be shown that the battery storage helps to hold the electric grid frequency in the desired band of ± 50 mHz. This is done by a simple frequency control of the storage input and output power with a proportional controller. The behavior can thus be considered as an additional primary control by the battery storage. It can also be shown that the voltage control is important for setting the bus voltages to appropriate values to minimize transmission line losses and to prevent the overload of transmission lines. An insufficient choice leads to an overload of `TransmissionLine` and `TransmissionLine4` in the simulation.

4 Summary and Outlook

In this paper, the extension of the freely available TransiEnt Library for modeling IES with the method of dynamic simulation is shown. Compared to the models used until the end of the TransiEnt.EE research project, the new electrical models allow much more detailed dynamic modeling and analysis of electric power grids. Load flow calculations can be performed. The stability of the electric grid can be analyzed by considering frequency, voltage and angle stability. This is enabled by new models based on a new connector that allows interconnected networks without overdetermined DAE. Numerically efficient transmission line, transformer and generator models are provided in the TransiEnt Library. Generator models allow different levels of detail in dynamic modeling of the electric grid, starting with simple models that only regard active-power-frequency behavior up to models with excitation systems, Two Axis Method based equations and distinct frequencies. The high modularity of Modelica allows the simple adaption of existing models as well as the extension of existing models.

As already mentioned, the TransiEnt Library can be extended within the given framework. As extension, models for On-Load Tap Changer (OLTC), Flexible Alternating Current Transmission Systems (FACTS) and High Voltage Direct Current (HVDC) transmission are possible. The main goal is the simulation of a representative coupled system of Northern Germany. A strategy for the system configuration has already been considered and research for designing this complex simulation model is currently done. The idea is the separation of the complete system model into regional subsystems. In the ResiliEntEE project, this method is called *Superstructure*. Parallel to this, it is investigated which stability phenomena should be regarded in the

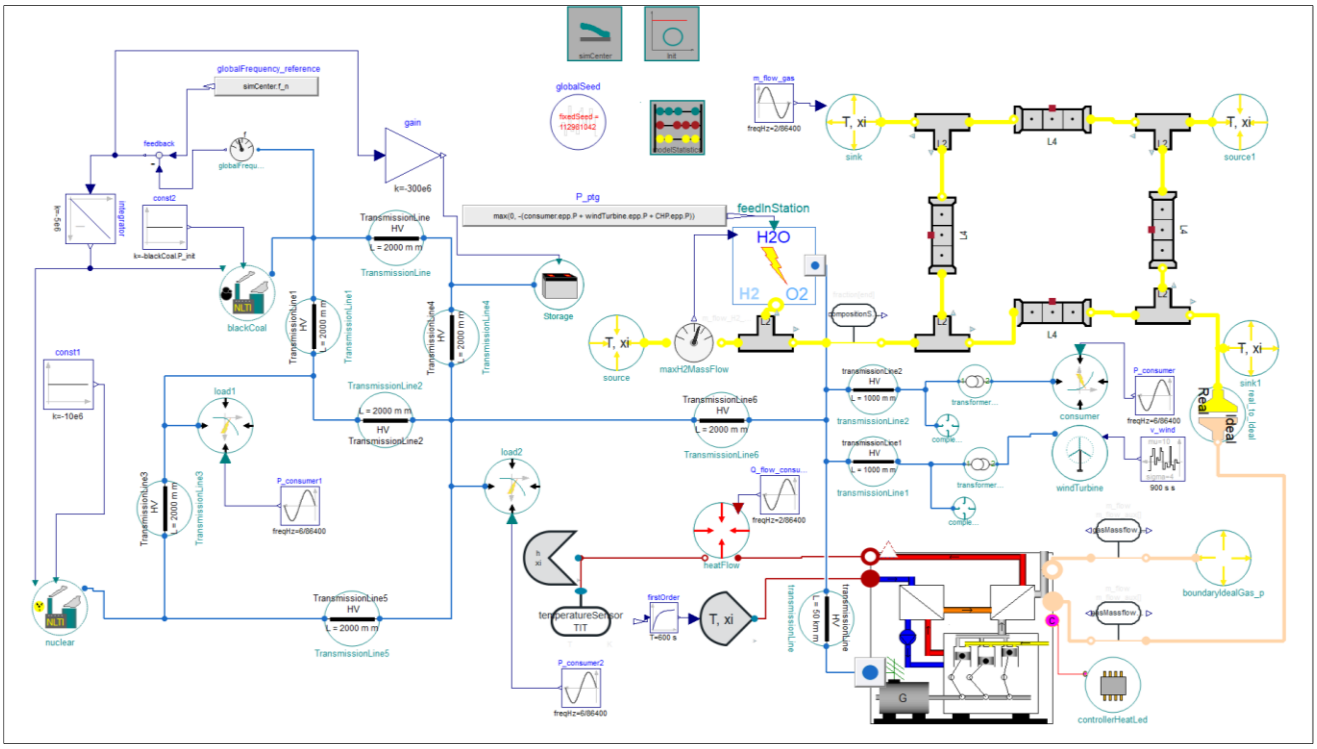


Figure 7. Example model for all three energy sectors within one simulation model in *Dymola*

electric grid for the chosen time and scenario horizon in order to investigate the overall system resilience.

The introduced features will be part of the upcoming release of the TransiEnt Library on the project website (University of Technology Hamburg, 2018).

Acknowledgements

The authors greatly acknowledge the funding from the German Federal Ministry of Economic Affairs and Energy for the project "ResiliEntEE - Resilience of coupled energy networks with a high share of renewable energies" (ResilientEE- Resilienz gekoppelter Energienetze mit hohem Anteil Erneuerbarer Energien, project number: 03ET4048).

References

ABB AG: NEPLAN® (Version V557). Available at: https://www.neplan.ch/neplanproduct/electricity, Zürich, Switzerland, 2016

L. Andresen, P. Dubucq, R. Peniche, G. Ackermann, A. Kather, G. Schmitz: Status of the TransiEnt Library: TransiEnt Simulation of Coupled Energy Networks with High Share of Renewable Energy. *Proceedings of the 11th International Modelica Conference*, Versailles, France, 2015.

L. Andresen, P. Dubucq, R. Peniche, G. Ackermann, A. Kather, G. Schmitz. Transientes Verhalten gekoppelter Energienetze mit hohem Anteil Erneuerbarer Energien. Abschlussbericht. *Technische Informationsbibliothek*, Hannover, Germany, 2017. doi: 10.2314/GBV:1002659345

J. Arrillaga, C. P. Arnold: Computer Analysis of Power Systems. *John Wiley & Sons Ltd*, Chichester, United Kingdom, 1990

d-maps: Schleswig-Holstein (Deutschland) Grenzen (weiß). Available at: https://d-maps.com/carte.php?num_car=6499&lang=de, Trets, France, 2018

Dassault Systèmes: Dymola® (Version 2019). Available at: https://www.3ds.com/de/produkte-und-services/catia/produkte/dymola/, Vélizy-Villacoublay, France, 2018

Hamburg University of Technology: TransiEnt Library. Available at: https://www.tuhh.de/transient-ee/, Hamburg, Germany, 2018

German Federal Ministry of Justice und Consumer Protection: Gesetz für den Ausbau erneuerbarer Energien (EEG). Available at: https://www.gesetze-im-internet.de/eeg_2014/, Berlin, Germany, 2017

IEEE Power and Energy Society: IEEE Recommended Practice for Excitation System Models for Power System Stability Studies. *The Institute of Electrical and Electronics Engineers, Inc.*, New York, United States of America, 2016

T. Jiang, X. Song, S. Schlegel, D. Westermann: Hybrids-Simulation using eMEGASIM and ePHASORSIM for Converter Dominated Distribution Grid. *NEIS Proceedings*, Hamburg, Germany, 2018

F. Milano: Power System Modelling and Scripting. *Springer*, London, United Kingdom, 2010

Modelica Association: Modelica® — A Unified Object-Oriented Language for Systems Modeling — Language Specification. *Version 3.4*, Linköping, Sweden, 2017

R. H. Park: Two Reaction Theory of Synchronous Machines — Part I. *AIEE Transactions*, Vol 48, pp. 716-727, 1929

TenneT TSO GmbH: Statisches Netzmodell. Available at: https://www.tennettso.de/site/Transparenz/veroeffentlichu ngen/statisches-netzmodell/statisches-netzmodell, Bayreuth, Germany, 2018

50Hertz Transmission GmbH: Statisches Netzmodell. Available at: https://www.50hertz.com/de/Anschluss-und-Zugang/Europaeischer-Stromhandel-und-Engpassmanagement/Statisches-Netzmodell, Berlin, Germany, 2018

Robust and accurate co-simulation master algorithms applied to FMI slaves with discontinuous signals using FMI 2.0 features

Andreas Nicolai[1] Anne Paepcke[1] Hauke Hirsch[1]

[1]Faculty of Architecture, Technische Universität Dresden, Germany, andreas.nicolai@tu-dresden.de

Abstract

Error control in system simulation using co-simulation techniques is a task for the employed simulation master. With the availability of the FMI standard version 2.0 and rollback capabilities of simulation slaves, master algorithms can be implemented with support of error controlled integration. Particularly, for automated integration tools, the problem-specific dynamic adjustment of communication interval lengths becomes a necessity to obtain reliable co-simulation results while maintaining calculation efficiency. The article discusses various master algorithms and time step adjustment strategies using a test case with discontinuous input/output signals. As expected, fixed-step Gauss-Jacobi and Gauss-Seidel algorithms are found to be generally unsuited for the task. Iteration-based time step adjustment rules are an improvement, yet cannot recognize discontinuities resulting from time-event. Since the traditional Richardson/step-doubling error estimate also fails to recognize discontinuous signal changes, a slope-based modified Richardson-test is introduced and successfully applied. Finally, it is concluded that a suitable master algorithm for such problems is the non-iterating Gauss-Seidel with modified Richardson communication interval adjustment.

Keywords: FMI, co-simulation, master algorithm, error control, adaptive

1 Introduction to FMI Co-Simulation

The FMI standard is an established simulation runtime coupling standard, implemented by many simulation tools, already. The standard is maintained by the Modelica Association and defines two operation modes; model exchange (single central integration core) and co-simulation (each slave has its own time integration engine). In this article we discuss the co-simulation approach. In this simulation coupling procedure, a simulation master requests simulation slaves to integrate a part of the simulation time interval, and communicate output variables to the master. Hence, this part is termed communication interval.

In the version 2.0 of the standard, a new feature was introduced that allows the master to trigger slaves to store their current state and restore it later. This permits the solver to rollback the state of an FMU to a previous state and repeat an already calculated communication interval. With that capability a co-simulation master has now several different options at obtaining a fully coupled/implicit solution of the coupled problem.

Typical choices for such co-simulation master algorithms have already been investiged (Clauß et al., 2017; Schierz et al., 2012). The authors also show that other new features of the FMI standard, like input/output extrapolation, positively influence simulation performance and stability. Further, communication step adjustment strategies were discussed that take into account the mathematical characteristics of the coupled simulation slaves.

In this article a special subset of problems will be analyzed: co-simulation slaves with discontinuous real signals.

1.1 Reference implementation MASTERSIM

The standard master algorithms discussed in aforementioned publications were implemented in an open-source reference implementation of a co-simulation master. The software MASTERSIM[1] has been used successfully in the scope of building energy performance and system simulation. In particular, the master was used for the co-simulation of a Modelica control model coupled to the building energy simulation model NANDRAD (Nicolai and Paepcke, 2017), whereby the building energy system FMU implemented already FMI v. 2.0 features (Nicolai and Paepcke, 2016).

The implemented algorithms are, however, of generic nature and can be applied to a wide range of physical problem domains. In many of such domains, discontinuous signals and variables appear and shall be treated correctly by the co-simulation master. With *correctly* we mean, that the simulation results are accurate within a requested/accepted tolerance band.

In order to analyze behavior of systems with discontinuous signals, we will use a dedicated test case to observe effects and results related to the chosen co-simulation master algorithms.

2 Test Case and Reference FMUs

The investigated test case was designed by C. Clauß (Clauß and Majetta, 2016):

[1]https://sourceforge.net/p/mastersim

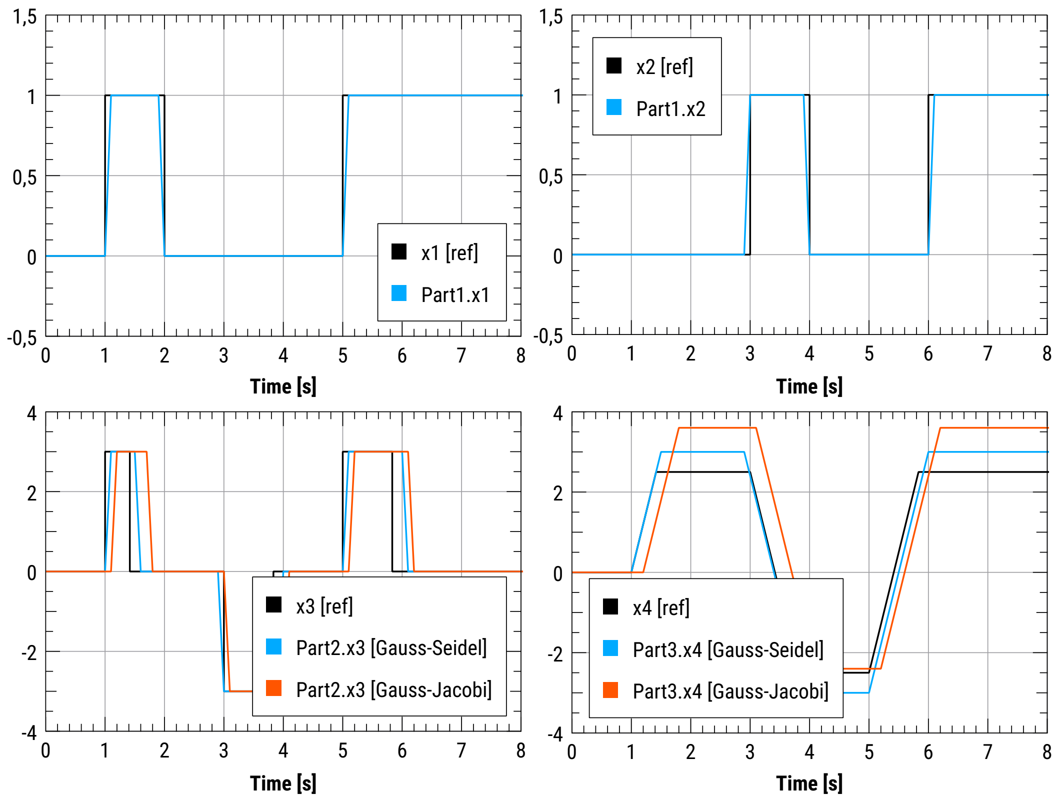


Figure 1. Non-iterating, constant step, step size 0.1 s; Reference solution (black), GAUSS-JACOBI (red), GAUSS-SEIDEL (blue)

$$x_1 = \begin{cases} 0 & t < 1 \quad \text{or} \quad 2 \le t < 5 \\ 1 & else \end{cases} \tag{1}$$

$$x_2 = \begin{cases} 0 & t < 3 \quad \text{or} \quad 4 \le t < 6 \\ 1 & else \end{cases} \tag{2}$$

$$x_3 = \begin{cases} 3 & x_1 = 1 \quad \text{and} \quad x_2 < 0.01 \quad \text{and} \quad x_4 < 2.5 \\ -3 & x_1 < 0.001 \quad \text{and} \quad x_2 > 0 \quad \text{and} \quad x_4 > -2.5 \\ 0 & else \end{cases} \tag{3}$$

$$\dot{x}_4 = 2x_3 \tag{4}$$

The solution shall be obtained for the variables x_1, x_2, x_3, x_4 in the time interval $t \in [0, 10]$, with $x_4(0) = 0$. Note, that the conditions for variable x_3 are not formulated as comparisons with 1 or 0 for the variables x_1 and x_2, but instead allow for small tolerances that may appear through use of difference-quotient approximations in Newton algorithms[2].

2.1 Reference Solution

The problem has an exact solution, with time and state events at: $t = 1 + 2.5/6$, $t = 3 + 5/6$ and $t = 5 + 5/6$.

[2]Since the problem is linear, Newton algorithms are not meaningful and are not discussed in this article.

2.2 FMI Co-Simulation Scenario

The problem shall be split into 3 parts (corresponding to one FMU, each), where the first implements equations (1) and (2), the second implements eqn. (3) and the third implements eqn. (4). Only part 2 and 3 are coupled in a cycle. Part 1 shall be always evalulated first. In iterative algorithms for the cycle, part 2 shall be evaluated before part 3. Only part 3 needs to implement time integration, whereby an analytic solution is available.

2.3 FMI Slave Generation

Initially, the FMI slaves were generated by writing the equations in Modelica and exporting co-simulation slaves. However, small deviations from the requested solution appeared. For example, when FMU 1 was requested to take 20 communication steps with 0.1 s each, the end time point of the last interval passed to the FMU was 2.0000000000000004 s. However, the Modelica-exported FMU still returned $x_1 = 1$, either due to internal rounding errors or because the variable x_1 was evaluated at the beginning of the interval [1.9...2.0]. This was in disagreement with the formulated problem, so a direct implementation of the FMI slaves in C/C++ code was used instead, see also source code from of the MASTERSIM test suite (Nicolai, 2018b).

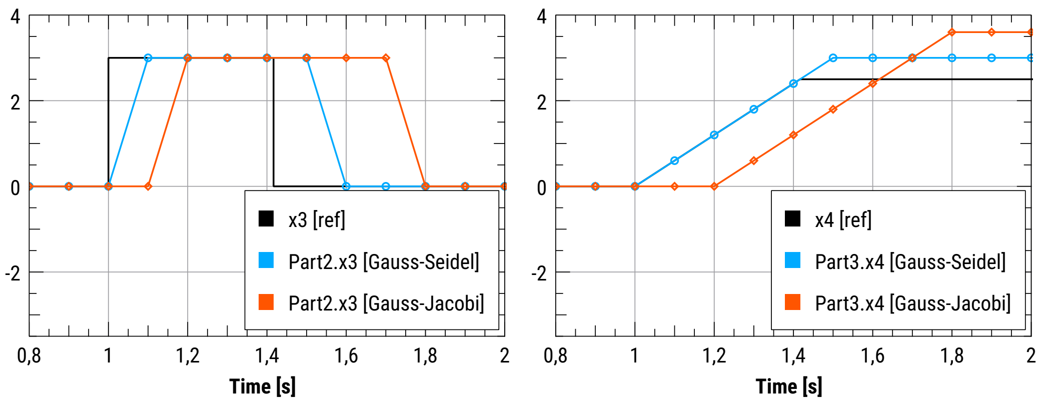


Figure 2. Non-iterating, constant step, step size 0.1 s, enlarged view; Reference solution (black), GAUSS-JACOBI (red), GAUSS-SEIDEL (blue)

3 Master Algorithms without Error Estimation

3.1 Constant-step Master Algorithm without Iteration (FMI v1)

Two popular examples for non-iterative constant-step co-simulation master algorithms are GAUSS-JACOBI and GAUSS-SEIDEL, both applicable for FMI slaves implementing only version 1 of the interface specification. With GAUSS-JACOBI, all FMUs are requested to integrate a time interval $[t,t+h]$ using input signals from time t. With GAUSS-SEIDEL, the FMUs are evaluated in sequence, where results from a previous FMU evaluation (output signals at $t+h$) are passed as inputs to a later FMU. Figure 1 shows results obtained when using constant communication interval lengths of 0.1 s. Clearly, the information propagation in the GAUSS-SEIDEL method leads to better results compared to GAUSS-JACOBI.

Results of FMU part 1 (variables x_1 and x_2) are identical for both methods, since they only depend on time. The differences in variables x_3 and x_4 are clearly visible. While in GAUSS-SEIDEL a change in variable x_1 or x_2 is already noted in the same communication interval, with GAUSS-JACOBI the information takes one more interval to reach FMU part 2 (variable x_3) and a further interval to influence variable x_4 (see Figure 2).

Of course, the accuracy of the solution can be increased by using smaller communication interval lengths, at the expense of computational (and mostly wasted) power.

3.2 Iterative Algorithm with Constant Steps

Once an FMU implements rollback capabilities, the co-simulation master can use iterative algorithms. It can be expected, that use of an iterative algorithm ensures information to propagate into all coupled FMUs in a cycle within the *same communication interval*. This implies, that the same communication interval is (re-)computed multiple times.

For iterative methods, a criterion for stopping the iteration is needed. In MASTERSIM the weighted-root-mean-square norm (5) is used for all exchanged real input/output variables. Variables of other types are ignored in the test.

$$|y|_{WRMS} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(\frac{y_{new,i}-y_{old,i}}{|y_{new,i}|\,r_{tol}+a_{tol}}\right)^2} \tag{5}$$

Convergence is reached when $|y|_{WRMS} \leq 1$, with r_{tol} and a_{tol} being relative and absolute tolerances, respectively. In order to avoid stalling of iteration, a maximum number of iterations is specified. Once this limit is exceeded, the simulation continues with the results from the last iteration, which may be potentially inaccurate.

Indeed, an iterative algorithm used in conjunction with GAUSS-SEIDEL (and tolerances $r_{tol} = a_{tol} = 10^{-5}$), improves the results as shown in Figure 3.

3.3 Iterative Algorithm with Adaptive Communication Step Size

When encountering a state event[3], the GAUSS-SEIDEL method does not converge. This observation can be used to adapt the communication step size and thus detect the state event with a discontinuous first derivative reliably. Whenever an iteration does not converge within the given maximum number of iterations, the communication step is reduced and the step is reattempted. If the iteration converges, (in this test case with linear coupling usually after one repeated step,) the step size is enlarged. This procedure successfully captures the state event, i.e. the correlated variables x_3 and x_4 (Figure 4).

The factors, by which the communication step is reduced or enlarged, determine to a large extend the overall simulation effort. Also, a lower limit for the time step has to be specified, below which a step is accepted even without convergence (limit to fall-back to non-adaptive variant). Without such a criterion, the simulation will get stuck at the first state event by continuously reducing the

[3]A state event describes a discontinuity in a variable or its derivative resulting from a condition that depends on the value of the variable itself. In contrast to that, a time event indicates a similar discontinuity/-condition that exclusively depends on the simulation time.

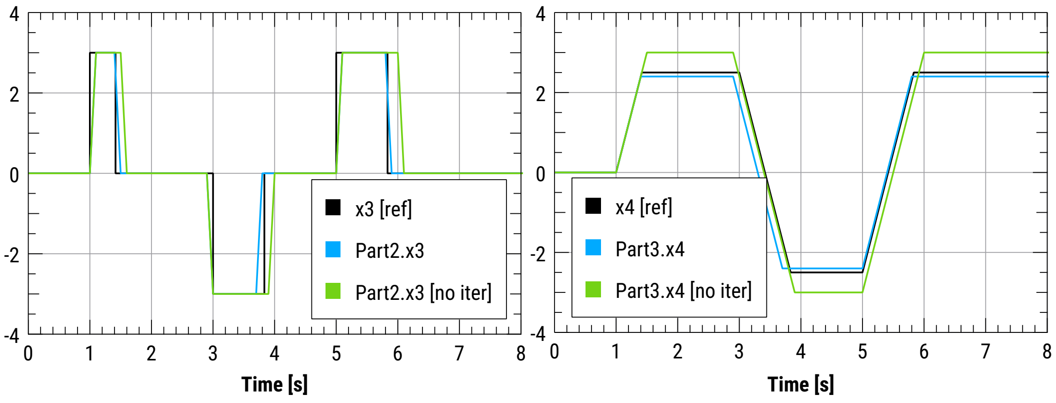


Figure 3. GAUSS-SEIDEL, iterating, max 2 iterationens, step size 0.1 s (blue), non-iterating GAUSS-SEIDEL (green)

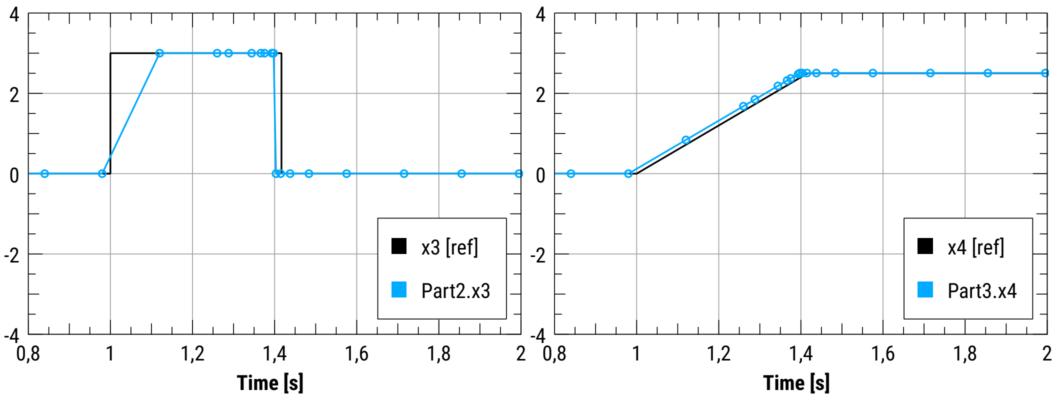


Figure 4. GAUSS-SEIDEL, iterating, max 2 iterations, variable step sizes (max. 0.14 s, min 0.005 s)

time step until rounding errors prevent further progress. Also, an upper limit of the time step is needed, to avoid overshooting over significant events. This limit is problem specific and for building simulation models, for example, the time step shall not exceed 30 min, so that hourly climate data is correctly considered. The heuristic parameters for this test case were set as follows: $f_{red} = 0.2, f_{exp} = 2, h_{fallback} = 0.005\,s, h_{max} = 0.14\,s$.

Consider Figure 4, where the occurance of the first state event is shown. We can observe the following:

- The time delay of the curves x_3 and x_4 results from the delayed jump of variable x_1 (occurs at time $t = 1.12s$).
- $t = 1.344s$ is reached with maximum step size of $0.14s$, the next step with same step size leads to $x_4 > 2.5$ and result in a convergence error.
- Hence, the step will be reduced by a factor of 5 ($\Delta t = 0.0224s$) and $t = 1.366s$ is reached; with $x_3 = 3$ and $x_4 < 2.5$ same as previous step and successful convergence, and step size is enlarged (doubled) to $\Delta t = 0.0448s$ afterwards.
- For the next interval, $0.0448s$ is again too long, a reduction of step size to $\Delta t = 0.00896s$ brings the solution even closer to the point of state event. Once again the step will be doubled after iteration has converged.
- The interval $1.375s$ to $1.393s$ will be completed with exactly one iteration; conditions $x_3 = 3$ and $x_4 < 2.5$ remain unchanged.
- In the next interval, the state event is reached. First the step size is reduced to $\Delta t = 0.007168s$. With this time step, the state event is surpassed and the step is again reduced to $\Delta t = 0.0014336s$. Now, the step size is below the assigned lower limit of $0.005s$, and the step is processed/accepted without iteration.
- Afterwards the step is doubled to $\Delta t = 0.0028672s$, which is still below the fallback limit, and results are again accepted without iteration.
- Now, that the discontinuity has been surpassed, in the next intervals only one iteration is needed and the step sizes are doubled after the end of each interval (see Figure 5 for variation of communication step sizes).

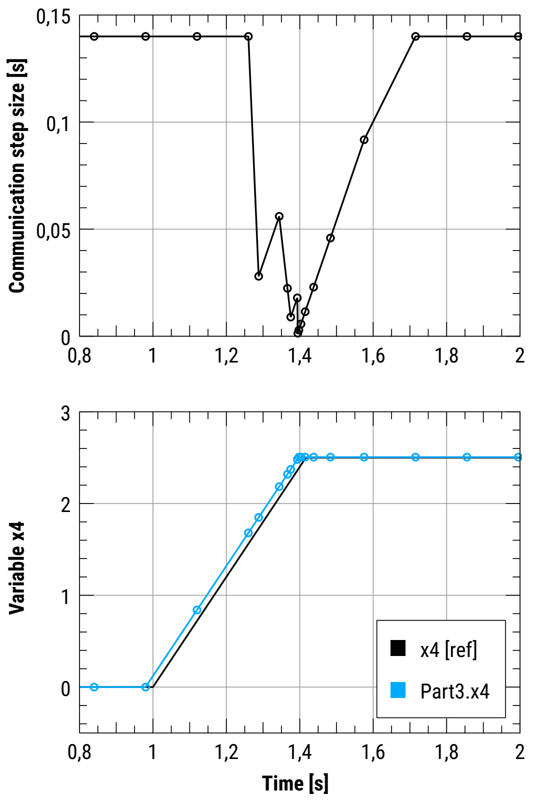


Figure 5. Accepted step sizes and resultant variable x_4 for iterative GAUSS-SEIDEL method.

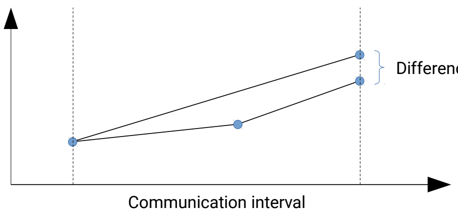


Figure 6. Illustration of Richardson/Step-doubling error test

4 Algorithm with Error Control

4.1 Error Estimate

While the previous algorithm did resolve the state event, it is not able to recognize discontinuities in variables that only depend on time (in this case, variables x_1 and x_2). Utilizing the rollback functionality of FMI v2.0, a straightforward approch is the use of Richardson-extrapolation (step-doubling) error test (Shampine, 1985). In the basic variant of this method, the step is executed first in full length, followed by a rollback to begin of the last step and execution of two steps with half communication length. The difference between the final results is an indicator for the integration error (Figure 6).

However, this error estimation fails when discontinuities appear within the interval, as in the case of variable x_1. In this case, the results of the full and the half-step are identical, and consequently the error cannot be detected.

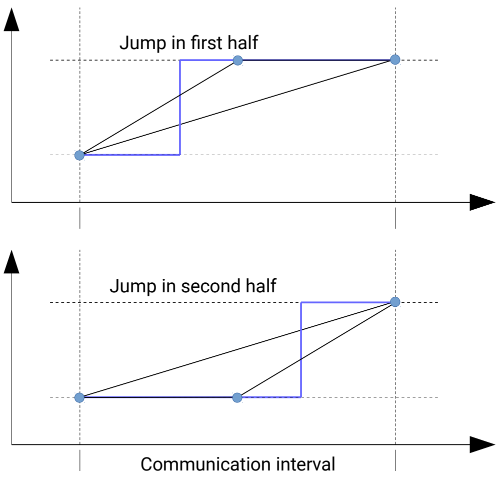


Figure 7. Inability of step-doubling/Richardson error test to recognize discontinuities in the solution; blue curve shows correct solution, the points show the results after FMU evaluation

Figure 7 illustrates the problem for both cases, when the discontinuity arises either before or after half of the interval.

A solution to this problem is to construct the error estimate based both on function results *and* slopes of the intervals, eqns. (6) and (7), respectively. Derivative information provided by FMUs themselves (a feature rarely supported, yet) is of no use, since derivatives at the end of the interval may well be the same for the full and a half interval (see Figure 7, bottom). Hence, the derivatives of the approximated solutions are constructed by difference-quotients:

$$\varepsilon_{Richardson} = \left| y(t+h)_{h/2} - y(t+h)_h \right| \tag{6}$$

$$\begin{aligned}\varepsilon_{slope} &= h\left(\dot{y}(t+h)_h - \dot{y}(t+h)_{h/2}\right)\\ &= h\left(\frac{y(t+h)_h - y(t)}{h} - \frac{y(t+h)_{h/2} - y\left(t+\frac{h}{2}\right)}{h/2}\right)\end{aligned} \tag{7}$$

The value $y(t+h)_{h/2}$ is obtained after executing two half-steps and $y(t+h)_h$ is the value obtained with the one full step. The derivatives $\dot{y}(t+h)_h$ and $\dot{y}(t+h)_{h/2}$ at $t+h$ are approximated by backward finite differences using the full step and the second half-step, respectively. For the final error test, the worst-case of both error estimates is taken to determine the communication step size.

The derivation and formal analysis of the robustness of this error estimate is beyond the scope of this article. However, since the construction of the error estimate requires a full step and two half steps just as the Richardson-test, it is meaningful to always evaluate both error tests individually and take the more critical one. Hereby, the master algorithm will detect any discontinuity and adjust communication step sizes, accordingly.

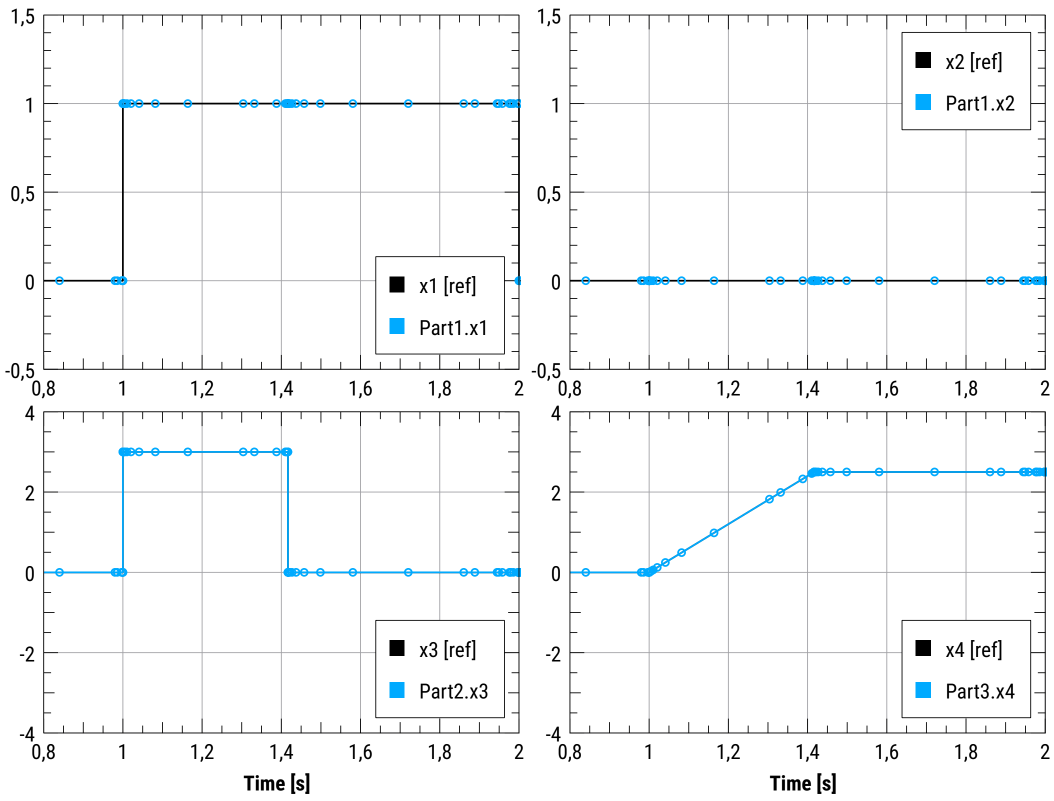


Figure 8. Results obtained with step-doubling/Richardson-error estimate and slope-difference error estimate

4.2 Results with Step Adjustment based on Richardson and Slope-Difference Error Estimates

The communication step is now adjusted based on the aforementioned error estimates, whereby the relative and absolute tolerances are set to 10^{-5}, each. The lower limit of a communication step is set to be $10^{-5}\,s$. This limit is also necessary for error controlled integration, since it is not possible to surpass a discontinuity otherwise. While implementing a formal root-finding procedure within the FMI co-simulation setup is possible with the FMI 2.0 rollback functionality, it will hardly be efficient. Thus, there must be a mechanism implemented to disable the error check, once time steps become too small.

The computed results (see Figure 8) appear to be almost identical to the exact solution. Around the occurance of the first time and state events many evaluations close to the event's locations become visible. This is due to reduced step sizes because of error test failures.

After passing a discontinuity/event the communcation step is quickly enlarged again (with max. factor 2). The resulting characteristic spikes (see Figure 9) in the evolution of the communication step sizes are representative for linear or mostly linear problems with discontinuities.

The algorithm can reproduce the exact solution sufficiently well while maintaining the tolerance limit of max. 10^{-5} (see Figure 10). Apparently, the lower step size limit ($10^{-5}s$) was selected sufficiently small to avoid overshooting the tolerance band.

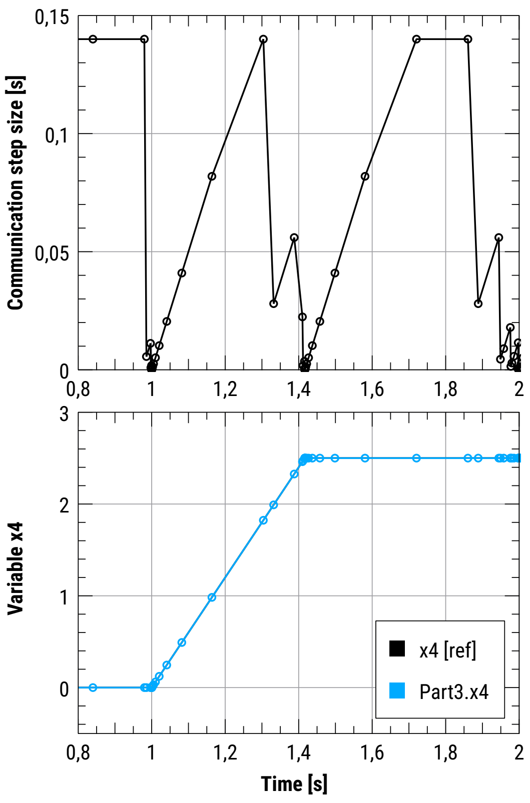


Figure 9. Accepted step sizes and resultant variable x_4 for the variant with error estimates.

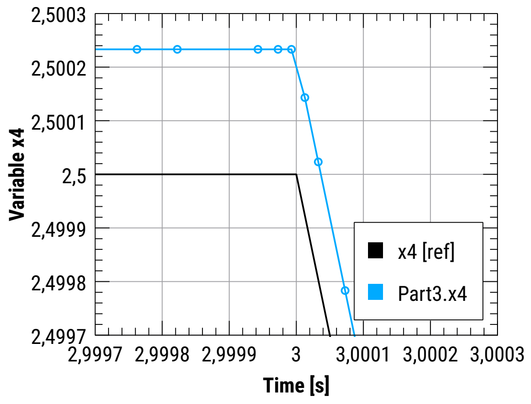


Figure 10. Remaining error in computed solution due to lower communication step limit

Further, the results are nearly the same for the iterative and non-iterative GAUSS-SEIDEL variants. The iterative variant typically requires two evaluations of each FMU per step. The simulation effort for the non-iterative variant is halved compared therewith.

5 Conclusions

Problems related to GAUSS-SEIDEL iteration methods in conjunction with discontinuities are well known for many decades, and presented stabilization techniques (iteration limit, lower time step limit, etc.) are state-of-the-art. Interestingly, though, these problems appear anew with the introduction of FMI for co-simulation, since authors/providers of individual FMUs may not have the global view necessary to identify such problems (or do not pay attention to potential numerical problems arising in the coupled simulation). Also, many straight-forward implementations of control models will generate discontinuous real signals, which should be handled in a robust way by co-simulation masters.

Without the FMI v. 2.0 state rollback feature, a sufficiently accurate solution of such problems cannot be guaranteed. Users of a co-simulation master would have to estimate a sufficiently small communication step, likely resulting in poor simulation performance. Still, the risk of unwanted errors remains.

In the presented case, a constant step size of 10^{-5}s would have been necessary for FMU v. 1 algorithms, resulting in 10^6 FMU evaluations. With the step adapting iterative GAUSS-SEIDEL algorithm, only 2639 FMU evaluations were needed. Hence, we can only recommend to implement FMI interface version 2.0 with rollback functionality (see open-source project FMICodeGenerator (Nicolai, 2018a) to assist in generating a minimal code base of such FMUs).

Implementing an error test procedure is generally advisable. However, when dealing with FMUs emitting discontinuous signals, a step-doubling error test method should be complemented with the presented slope-based check. Otherwise, the co-simulation master will not be able to reliably detect discontinuities arising from time-events. For the presented test case, the variant without iteration and included error test yields the best performance, i.e. smallest number of FMU evaluations while maintaining accuracy. A more in-depth analysis of the various algorithms tested and the respective FMU evaluation counts can be found in the german test case publication (Nicolai, 2018c).

One has to keep in mind that the upcoming FMI 3.0 standard will address such issues to some extend. It will allow FMUs to detect discontinuities and return prematurely to the master and by this notify the master about the exact location of the discontinuity. However, similarly to the introduction of the FMI 2.0 standard, it will take several years until simulation slaves and co-simulation masters implement support for such FMI 3.0 features. Until then, the proposed algorithm and error detection method will be an effective way of ensuring accuracy.

Lastly, the test case shows how much influence the choice of master algorithm and heuristic parameters may have on the results. Also, during development of the MASTERSIM code, many bugs appeared that may involuntarily cause the coupled simulation to generate wrong results, despite the fact that the stand-alone tests done with test FMUs provided on fmi-standard.org ran successfully. We conclude, that co-simulation masters should additionally be tested with coupling scenarios and tested against provided, algorithm-specific results.

6 Acknowledgements

We gratefully acknowledge the support and funding received from the German Federal Ministry for Economic Affairs and Energy in the research projects *EnOB: EnTool-CoSim* (#03ET1215A) and *SimQuality - Entwicklung von Qualitätsstandards für die energetische Gebäude- und Quartierssimulation als Planungswerkzeug* (#03ET1570F).

References

Christoph Clauß and Kristin Majetta. FMI Co-Simulation Test Scenarios. personal communication, 2016.

Christoph Clauß, Kristin Majetta, and Richard Meyer. Application of Richardson Extrapolation to the Co-Simulation of FMUs from Building Simulation. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic,*, pages 79–88, 2017. doi:10.3384/ecp1713279.

Andreas Nicolai. FMI Code Generator project. Github Repository, 2018a. URL `https://github.com/ghorwin/FMICodeGenerator`.

Andreas Nicolai. Open-Source Co-Simulation Master MASTERSIM, 2018b. URL `http://bauklimatik-dresden.de/mastersim`.

Andreas Nicolai. Co-Simulations-Masteralgorithmen - Analyse und Details der Implementierung am Beispiel des Masterprogramms MASTERSIM. *Qucosa*, 2018c. doi:urn:nbn:de:bsz:14-qucosa2-319735. URL http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa2-319735.

Andreas Nicolai and Anne Paepcke. Transformation der Gebäudeenergiesimulation NANDRAD mit variablem Zeitschrittlöser in eine Co-Simulation. In *Proceedings of the BauSIM 2016 in Dresden*, 2016.

Andreas Nicolai and Anne Paepcke. Co-Simulation between detailed building energy performance simulation and Modelica HVAC component models. In *Proceedings of the Modelica Conference 2017 in Dresden (DOI:10.3384/ecp17132)*, 2017.

Tom Schierz, Martin Arnold, and Christoph Clauß. Cosimulation with communication step size control in an FMI compatible master algorithm. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, 2012.

L. F. Shampine. Local error estimation by doubling. *Computing*, 34(2):179–190, Jun 1985. ISSN 1436-5057. doi:10.1007/BF02259844. URL https://doi.org/10.1007/BF02259844.

Development of a General-purpose Analytical Tool for Evaluating Dynamic Characteristics of Thermal Energy Systems

Yutaka Watanabe[1] Toru Takahashi[1]

[1]Energy Engineering Research Laboratory, Central Research Institute of Electric Power Industry (CRIEPI), Japan, {yutaka,toru-tak}@criepi.or.jp

Abstract

We present an original tool for analyzing the dynamics of a wide range of thermal energy systems using Modelica, developed by the Central Research Institute of Electric Power Industry in Japan. This tool was originally developed to analyze thermal power generation systems and has been validated against several sets of operational data so far. This paper reports the extension of the tool to the calculation of the thermophysical properties of not only water/steam and air/gas but also various refrigerants implemented by the ExternalMedia library. This addition expands the range of energy systems that can be analyzed with the tool. To validate the simulation data, a comparison between experimental and simulation data targeting a CO_2 heat-pump loop facility was drawn.

Keywords: Modelica, energy system, heat pump

1 Introduction

A great deal of renewable energy sources (REs) must be introduced to electric power grids to reduce CO_2 emissions in the future. However, the fluctuating and unpredictable power output of REs like photovoltaic generation and wind power generation can destabilize an electric power grid. Therefore, it is necessary to establish a realistic approach to compensate for REs load fluctuation. Thermal Power Generation (TPG) systems, which have high efficiency and excellent flexibility, is promising as a power source that handles load fluctuation. If the flexibility of TPG systems can be improved furthermore, it would be possible to introduce a large amount of REs in the electric power grid while reducing CO_2 emissions. In both the design phase and during the operation of such systems, dynamic simulation is an important for determining the limits of flexibility and improving the operability of TPG systems.

The Central Research Institute of Electric Power Industry (CRIEPI) in Japan has developed a dynamic analytical tool based on Modelica for evaluating the dynamic characteristics of a new and an existing TPG systems (Takahashi et al., 2016; Watanabe et al., 2017). The usefulness of this tool has been demonstrated by applying it to evaluate dynamic characteristics and improve the operability of TPG systems (Watanabe et al., 2017).

Meanwhile, another option that adjusts the power demand at the customer-side such as automatic demand response and virtual power plant schemes have been considered to cope with load fluctuation of REs in Japan. Therefore, a dynamic analysis tool will be also needed to evaluate the performance of demand side resources such as cogeneration systems, heat pumps and an air-conditioners that are affected by load-changing operations. Novel customer-side systems will be also needed to facilitate demand management.

In this study, we aimed at improving our original tool more generically to analyze both TPG systems and customer-side equipment systems. Initially, efficiently adding and handling new working fluids such as fluorocarbon-based refrigerants and natural refrigerants posed developmental bottle necks; however, there are already useful open source libraries developed like the External Media library (Casella and Richter, 2008) for the calculation of thermophysical properties, and it was shown that this library could be combined with other libraries (Quoilin et al., 2014; Casella et al., 2013) and proved to be useful. For efficient implementation, using these libraries to incorporate in our tool is also an effective option, but there was a problem about model connection and management between these libraries and our existing tool. Therefore, the External Library was directly implemented and a new package handling a refrigerant fluids was added in our original tool. These changes improve the modeling tool so that it could be used to analyze the dynamic characteristics of various types of energy systems more generically, such as a complex energy system combining existing TPG systems and heat pump systems.

In this paper, the outline of our new developed tool was shown firstly. And then, as a case study for testing the newly added part, a simplified dynamic model of a heat pump system was constructed using this tool. The model results are then compared with experimental data of the CO_2 heat-pump loop facility at CRIEPI for validation.

2 Outline of Analytical Tool

A tool for analyzing TPG systems dynamically was developed at CRIEPI based on the Modelica language using Dymola environment (Takahashi et al., 2016; Watanabe et al., 2017). In this tool, various component models (e.g. compressor, gas turbine, combustor, heat exchanger, steam drum, etc) mainly for analyzing TPG systems are already in place along with a node model for calculating the pressure and enthalpy between components, a valve model calculating flows between components, and a control model (e.g., a proportional–integral PI controller). The component models are subjected to mass and energy conservation equations based on physical laws. The models are packaged in a way that makes it easy to construct a dynamic system model.

The original analytical tool could not analyze the dynamic characteristics of equipment that uses either synthetic (e.g., chlorofluorocarbons) or natural (e.g., CO_2 and ammonia) refrigerant as the working medium. Therefore, to model customer-side equipment such as heat pumps or air conditioners, the function that calculates thermophysical properties of the working fluid needs to be extended.

Fig. 1 outlies the newly extended dynamic analytical tool, which is based on the previous tool for TPG systems and now incorporates the ExternalMedia library (Casella et al., 2008; Trapp et al., 2014) for the calculation of thermophysical properties. By implementing the ExternalMedia library, the range of working fluids that can be handled by this tool is extended. The ExternalMedia library is an open-source library and includes interface functions to return calculation results of the thermophysical property value using external software to the analysis model. The ExternalMedia library links to REFPROP of the National Institute of Standards and Technology (Lemmon et al., 2010), CoolProp (Bell et al., 2014), and FluidProp (Colonna and Stelt, 2004) as reference external databese. In the ExternalMedia library, the thermophysical value is defined as a variable with the same form as the ThermodynamicState variable as it is defined in Modelica.Media. Therefore, the library is relatively easy to inconnect with each model element.

Table 1 lists the packages included in the dynamic-characteristics analytical tool. In addition to the group of packages for TPG systems, the new version of the tool includes the “REFRIGERANT MODEL” package to represent customer-side equipment model and uses the ExternalMedia library for calculating the physical properties of the working fluids. The “REFRIGERANT MODEL” package contains basic and simple modes of dynamic-characteristics of a compressors, heat exchangers, piping, and valves and so on.

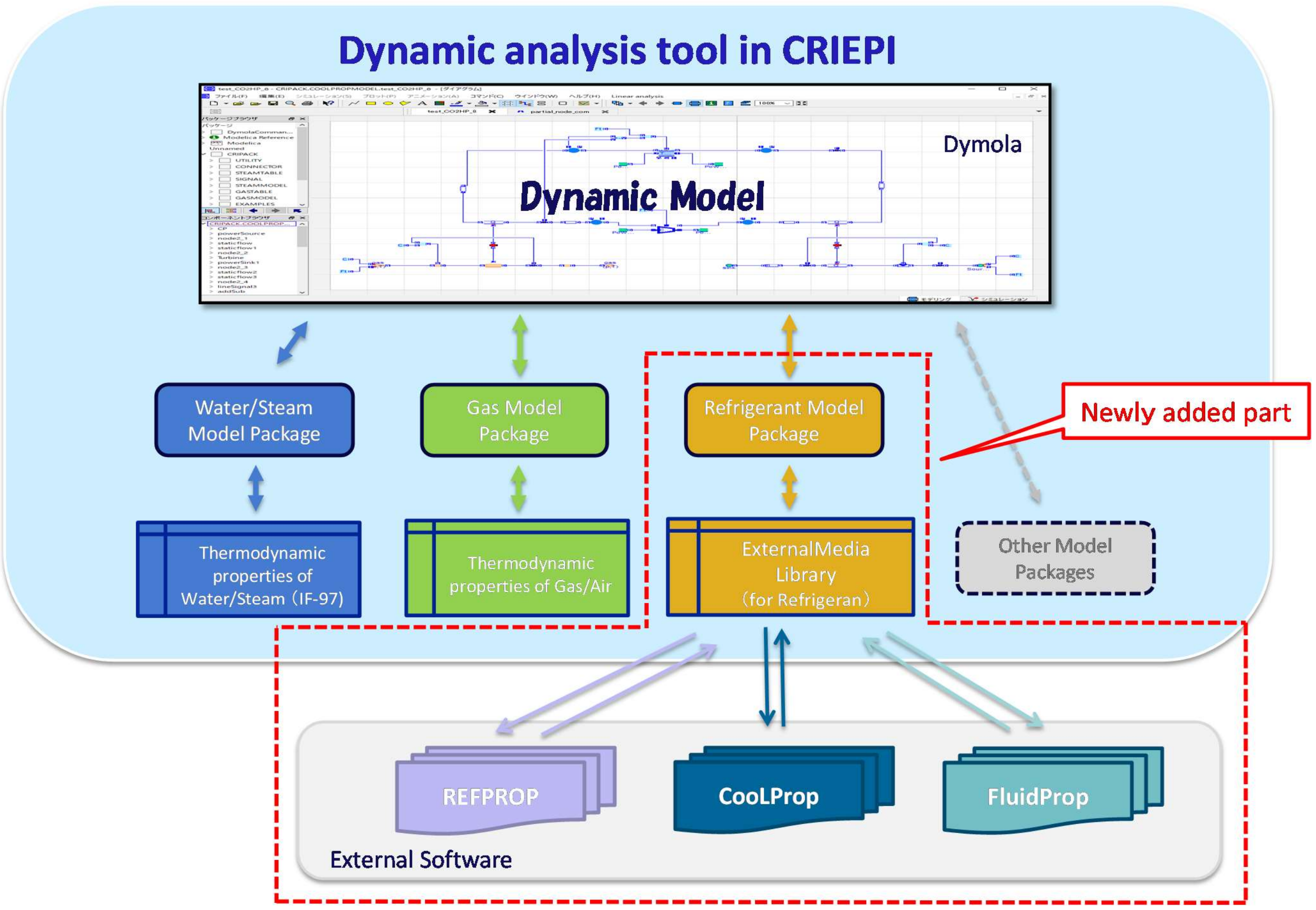

Dynamic analysis tool in CRIEPI
Dymola
Dynamic Model
Water/Steam Model Package
Gas Model Package
Refrigerant Model Package
Newly added part
Thermodynamic properties of Water/Steam (IF-97)
Thermodynamic properties of Gas/Air
ExternalMedia Library (for Refrigeran)
Other Model Packages
REFPROP
CoolProp
FluidProp
External Software

Figure 1. Schematic of the developed tool.

Table 1. Outline of packages in the analytical tool.

Type	**Name**	**Outline**	**Model Example**
Equipment model	STEAMMODEL	Models for fluid dynamics analysis of water and steam instruments	Volume element, Pipe, Valve, Steam turbine, Drum, Heat exchangers, Pump, etc.
	GASMODEL	Models for dynamic analysis of equipment using fluids containing gases other than water vapor	Volume element, Pipe, Valve, Compressor, Combustors, Turbine, Heat exchanger, etc.
	REFRIGERANT MODEL	**Models for dynamic analysis of Freon refrigerant or natural refrigerants**	**Volume element, Pipe, Valve, Compressor, Turbine, Heat exchanger, etc.**
	SIGNAL	Models for signals and control	Step signal model, ramp signal model, PI controller model, etc.
Functions of thermal properties of working fluid	STEAMTABLE	Functions for thermodynamic properties of water and steam based on International Association for the Properties of Water and Steam 1997	Calculation functions of physical properties such as enthalpy, pressure, and specific volume concerning water/steam
	GASTABLE	Functions for thermodynamic properties of gas. Based on IGTC-83 paper (Matsunaga, 1983) and Chemical Properties Hand Book, McGraw-Hill (Yaws, 1999) , create functions necessary for dynamic analysis	Calculation functions of enthalpy, pressure, specific volume, etc. Physical property value concerning gases such as CO_2, O_2, and H_2O and mixed gas
	ExternalMedia (Casella and Richter, 2008)	**Functions for accessing NIST REFPROP, CoolProp, FluidProp and calculating physical property values necessary for dynamic characteristic analysis.**	**Calculation functions of physical properties of the refrigerant provided by external software.**
Models and functions required for model creation or internal computation	CONNECTOR	Models that regulates connection of a device model and a control model	Composition of the working fluid, physical property value, and real number
	UTILITY	General functions for calculation	Number sorting

※Text in bold and red describes components added in the present study

3 Case Study

In this section, the validity and applicability of the tool are verified by comparing results obtained with the dynamic model constructed with the present tool with data obtained from actual machine operation.

3.1 Outline of CO_2 Heat-pump Loop

The target system is the CO_2 heat-pump loop test facility at CRIEPI (Saikawa et al., 1998). Fig. 3 shows a photograph of the experimental apparatus, and Fig. 4 shows a schematic of the equipment. The apparatus includes a compressor, gas coolers, an electro-motion expansion valve, and evaporators. The compressor is an oil-free reciprocating model driven by a variable-speed inverter driven motor and has two pistons and cylinders. The temperature and flow rate of the fluid in each heat exchanger are controlled automatically.

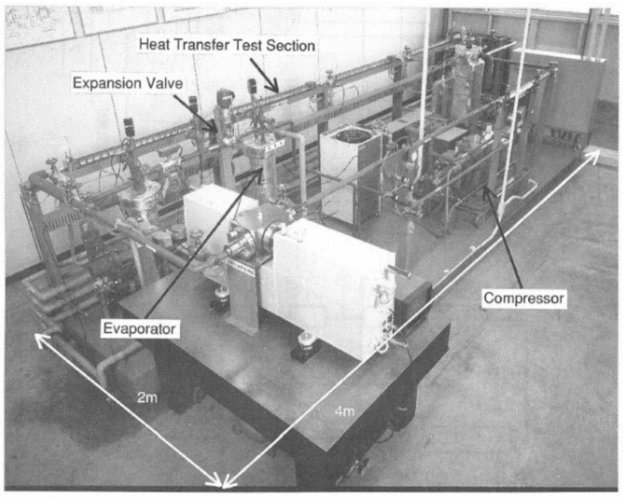


Figure 3. Photograph of the CO_2 heat-pump loop.

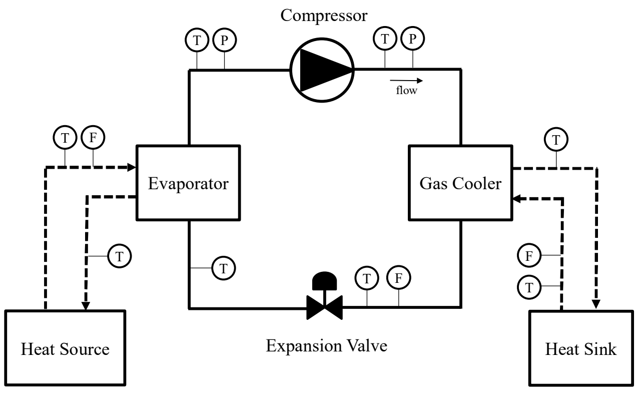


Figure 4. Schematic of the CO_2 heat-pump loop.

Table 2. Specifications of the CO_2 heat-pump loop.

Cycle	Single-stage compression cycle
Power input to compressor	<3 kW (variable by inverter)
Compressor	Reciprocating two oil-free cylinders (D48 × L70 [mm])
Heating capacity	4 - 7 kW
Working pressure	Low pressure: 3 - 4 MPa High pressure: 8 - 12 MPa
Heat source	Brine
Heat sink	Water

3.2 Dynamic Model of CO_2 Heat-pump

A system model of this facility was constructed using the extended analysis tool. Fig. 5 shows an outline of the dynamic model of the CO_2 heat-pump loop facility. In construction of the dynamic model, the refrigerant circuit is used in the "REFRIGERATORMODEL" package, the water side circuit of the heat exchanger is used in the "STEAMMODEL" package, and the controller model is used in the "SIGNAL" package. The calculation equations of the main component models are as follows.

Compressor model:

The flow of the working fluid and the output temperature considering the adiabatic efficiency are calculated with Eqs. (1) and (2), respectively, and the compressor power is calculated as Eq. (3):

$$F = f(N_{comp}), \tag{1}$$

$$h_{out} = h_{in} + \frac{(h_{ad} - h_{in})}{\eta}, \tag{2}$$

$$W_{comp} = F \cdot (h_{out} - h_{in}). \tag{3}$$

Gas cooler and Evaporator model:

The mass and energy equations for the refrigerant side pipe and the water side pipe are represented as Eqs. (4)-(7) (e.g. (Quoilin,2011)). The energy-balance and heat transfer equations for the metal wall on the heat transfer side are represented as Eqs. (8)-(10). These models are also divided vertically and into sections *i* and *j*.

Refrigerant side:

$$V_{r,i}\left(\frac{\partial \rho_{r,i}}{\partial p}\frac{dp_r}{dt} + \frac{\partial \rho_{r,i}}{\partial h}\frac{dh_{r,i}}{dt}\right) = F_{r,i-1} - F_{r,i} \tag{4}$$

$$V_{r,i}\left(-\frac{dp_r}{dt} + \rho_{r,i}\frac{dh_{r,i}}{dt}\right) = F_{r,i-1}(h_{r,i-1} - h_{r,i}) + Q_{r,i} \tag{5}$$

Water side:

$$V_{w,j}\left(\frac{\partial \rho_{w,j}}{\partial p}\frac{dp_w}{dt} + \frac{\partial \rho_{w,j}}{\partial h}\frac{dh_{w,j}}{dt}\right) = F_{w,j+1} - F_{w,j} \tag{6}$$

$$V_{w,j}\left(-\frac{dp_w}{dt} + \rho_{w,j}\frac{dh_{w,j}}{dt}\right) = F_{w,j+1}(h_{w,j+1} - h_{w,j}) - Q_{w,j} \tag{7}$$

Metal wall:

$$cp_{m,i}M_{m,i}\frac{d}{dt}T_{m,i} = Q_{r,i} - Q_{w,i} \tag{8}$$

$$Q_{r,i} = K_{r,i} \cdot (T_{m,i} - T_{r,i}) \tag{9}$$

$$Q_{w,i} = K_{w,i} \cdot (T_{w,i} - T_{m,i}) \tag{10}$$

Expansion valve model:

The flow rate is calculated with Eq. (11). The coefficient varies depending on the valve-opening signal:

$$F = C(u) \cdot \left(\sqrt{\rho_{in} \cdot (P_{in} - P_{out})}\right). \tag{11}$$

In this model, the controller model (PI controller) is used to control (i) the opening rate of the expansion valve in response to the difference between the inlet and outlet temperatures of the evaporator and (ii) the flow rate of the water in the heat exchangers.

3.3 Simulation Conditions

To assess the validity of the dynamic CO_2 heat-pump model, the calculation values were compared with previously reported experimental data from the CO_2 heat-pump loop (Saikawa et al., 1998). Fig. 6 shows the scenario used to assess the validity of the dynamic model. In this case, the inverter frequency of the compressor was varied, which changes the flow rate of the CO_2 working fluid is changing.

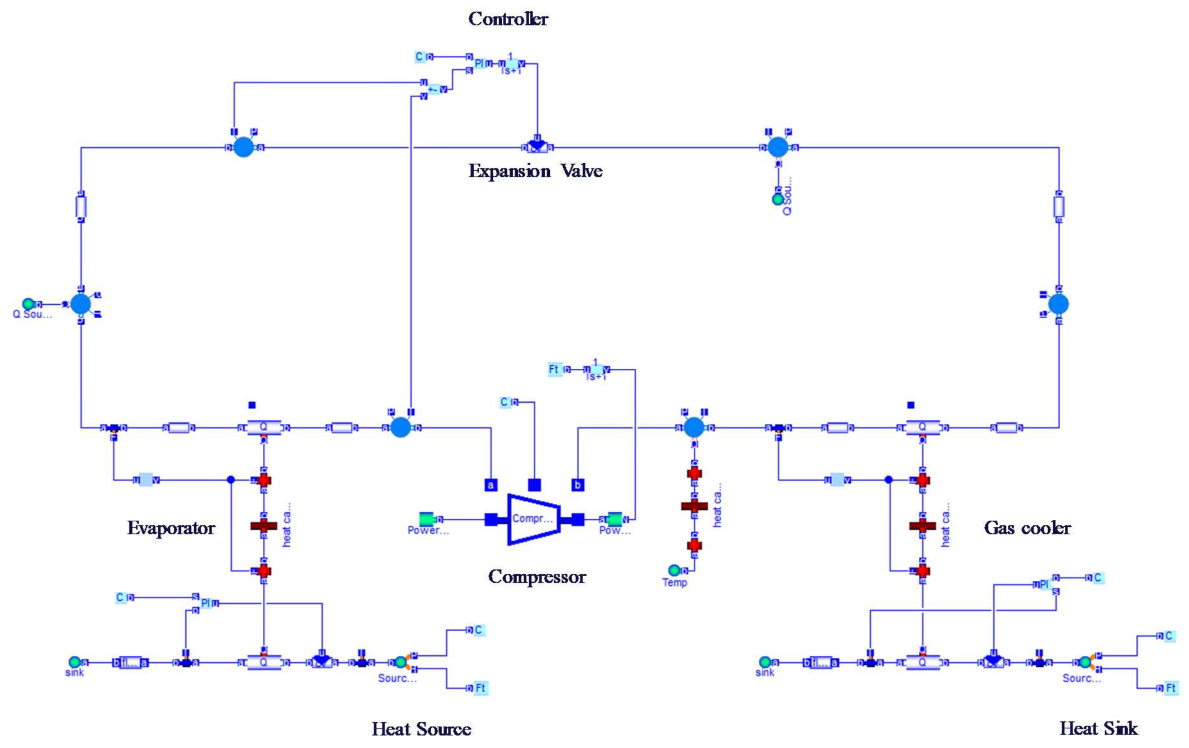


Figure 5. Dynamic model of CO_2 heat-pump-loop facility.

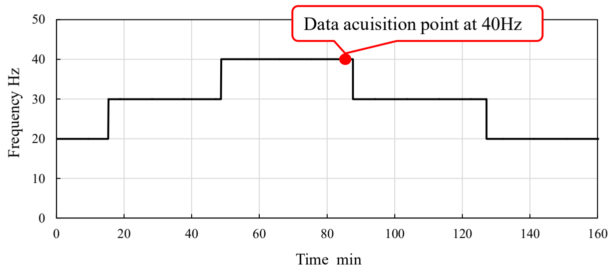
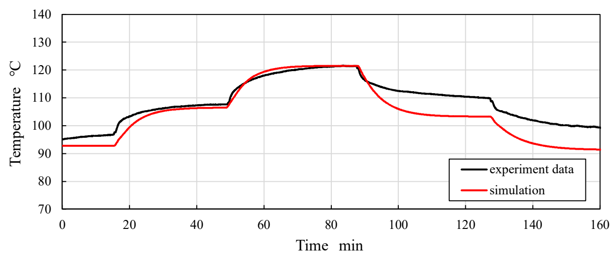


Figure 6. Test scenario for validation: the inverter frequency of the compressor is changing.

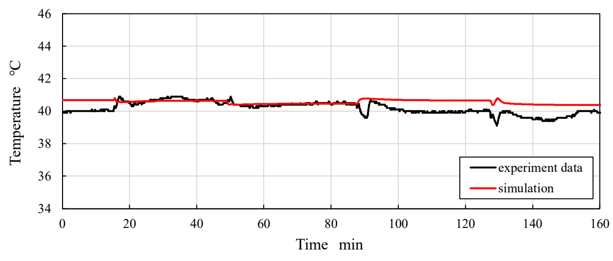
The unmeasured model parameters in the dynamic model, such as the adiabatic efficiency of the compressor and heat transfer coefficient of the heat exchanger, the heat and mass balance calculation using measured operational data of 40Hz (seen in Fig.6) was carried out using the EnergyWin software (Koda and Takahashi, 1999), and these estimated values are set as the initial conditions. The performance parameters of each component set as a function depending on flow rate using the calculation results from steady-state operational data at 20, 30 and 40 Hz. The volume and weight of each equipment and the heat capacity of the heat exchangers were set with reference to the design specifications.

3.4 Results and Discussion

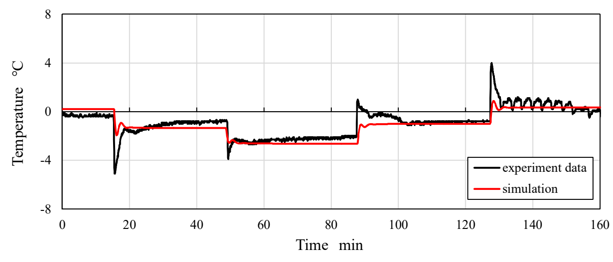
Figs. 7–9 compare the experimental data with the simulation results for the CO_2 heat-pump loop. As shown, the dynamic simulation reproduces the behavior of the operational process. In Figs. 7 and 8, the experiment data and simulation results from the compressor outlet disagree when the frequency decreases. The simplification of the compressor model likely decreased the simulation accuracy, and the model leaves out some phenomena such as volume elements and heat loss. Therefore the model of the compressor characteristics should be prepared with more detail. In Figs. 9(a) and (b), the simulated flow rate shows greater error when the frequency decreases, likely because of the above-mentioned error of the compressor outlet temperature. In addition, the delay factor of the water side-circuit volume element is not considered in the model. Although the inlet temperatures of water from the heat source and heat sink fluctuate slightly in experimental data, the setting parameter of constant inlet water temperature also seems to affect the quantitative results of the simulation, except when the compressor is driven at 40 Hz. The delay in the measuring instruments can be explains this error because the responding speed of this system is quickly. Regarding the quantitative difference when the compressor is under a partial load, the state quantity of working fluid in the heat exchanger and the change of the heat transfer coefficient following the phase change needs to be modeled in greater detail.



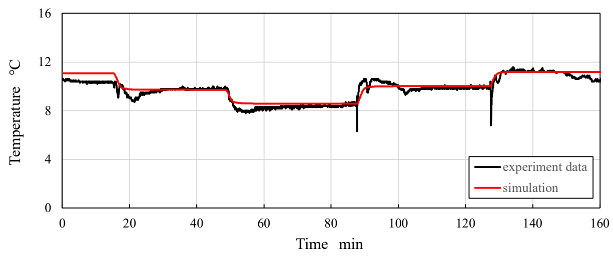
(a) Compressor outlet



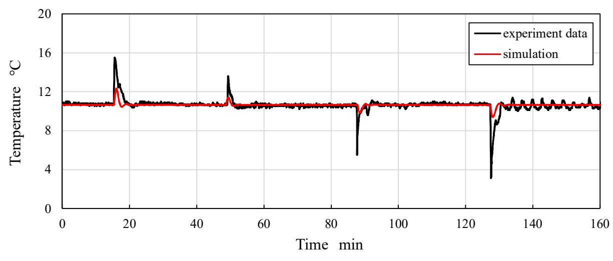
(b) Expansion valve inlet



(c) Evaporator inlet

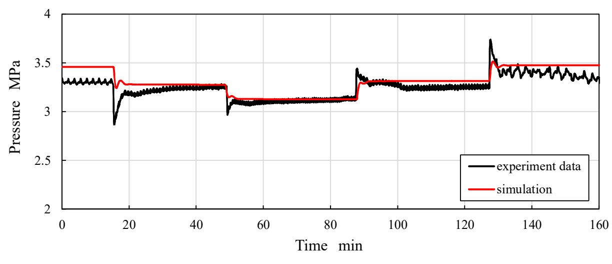


(d) Evaporator outlet

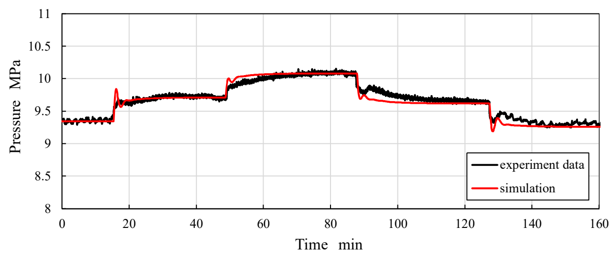


(e) Difference in evaporator inlet/outlet temperature

Figure 7. Results of temperature behavior.

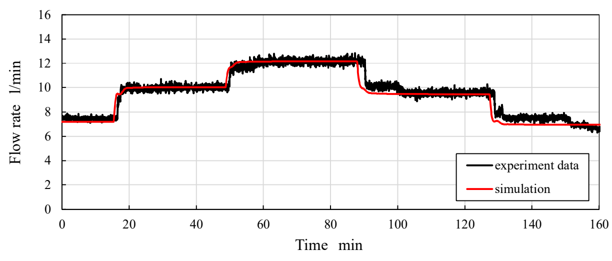


(a) Compressor inlet

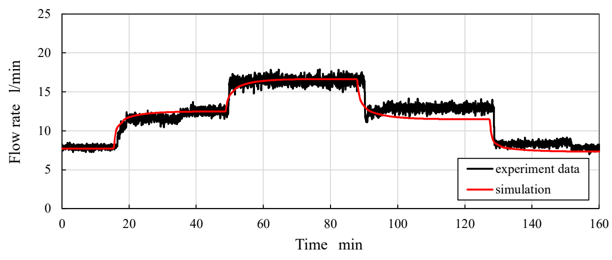


(b) Compressor outlet

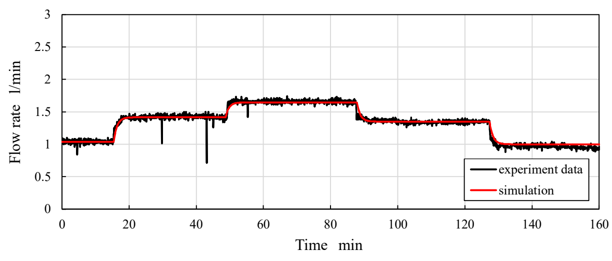
Figure 8. Results of pressure behavior.



(a) Heat source



(b) Heat sink



(c) CO_2 working fluid

Figure 9. Results of flow rate behavior.

These results suggest that the accuracy of some model elements needs to be improved, but it is useful for evaluating the operating performance at the system level.

4 Summary and Future Work

A new function for modeling refrigerant working fluids was added to the CRIEPI's Modelica dynamic-characteristics analytical tool by integrating the ExternalMedia library. The innovations discussed above yield an analytical tool that can be used to more general-purpose energy systems. The validity of the tool was assessed via comparison with experimental data measured from a hot-water supply system with CO_2 refrigerant. The model accuracy of some elements of the system needs further improvement, though sufficiently accurate results for constructing a dynamic model were obtained. In future work, we plan to assess the validity of the tool for other refrigerant types and systems, develop more advanced tools, and challenge the operability evaluation for complex systems that comprise devices with various working fluids. In addition, to promote application to real machines, data calibration and data assimilation as a function to utilize actual operational data will be carried out and models will be developed for the purpose.

NOMENCLATURE

F	Flow rate [kg/s]
h	Specific enthalpy [J/kg]
P	Pressure [Pa]
M	Mass [kg]
V	Volume [m3/kg]
Q	Heat flow [W]
T	Temperature [K]
K	Global Heat transfer coefficient [W/K]
ρ	Density [kg/m3]
W	Power [W]
C	Flow coefficient [-]
u	Input
N	Rotation signal
η	Adiabatic efficiency [-]

Subscript

r	Refrigerant
m	Metal
w	Water
comp	Compressor
ad	Adiabatic change

Acknowledgments

We thank the CRIEPI heat-pump research group for providing us with the experimental data discussed above.

References

Eiichi Koda and Toru Takahashi. Development of general purpose software to analyze the steady state of power generation systems, Energy Conversion and Management journal, Vol.43, pp.264-268, 1999.

Yutaka Watanabe, Toru Takahashi and Masashi Nakamoto. Dynamic Simulation of Startup Characteristics for the Advanced Humid Air Turbine System. Proceedings of ASME Turbo Expo 2017, GT2017-64699, 2017.

Toru Takahashi, Masashi Nakamoto and Yutaka Watanabe. Construction of dynamic analysis tool for thermal power systems, CRIEPI Research Report, M15005, 2016.

Michiyuki Saikawa, Katsumi Hashimoto, Hiromi Hasegawa, Tetsushiro Iwatsubo. Study on Efficiency and Control Method of CO2 Heat Pump, CRIEPI Research Report, W98004, 1998.

Francesco Casella and Christoph Richter. ExternalMedia: a Library for Easy Re-Use of External Fluid Property Code in Modelica, Proceedings of 6th International Modelica Conference, pp.157-161, 2008.

Carsten Trapp, Francesco Casella, Teus can der Stelt and Piero Colonna. Use of External Fluid Property Code in Modelica for Modelling of a Pre-combustion CO2 Capture process Involving Multi-Component, Two-phase Fluids, Proceedings of 10th International Modelica Conference, pp.1047-1056, 2014.

Eric W. Lemmon, Marcia L. Huber and Mark O. McLinden. NIST Standard Reference Database 23 Reference Fluid Thermodynamics and Transport Properties-REFPROP. National Institute of Standards and Technology, Standard Reference Data Program, 2010.

Ian H. Bell, Jorrit Wronski, Sylvain Quoilin and Vincent Lemort. Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp. Industrial Engineering Chemistry Research, 53(6), pp.2498-2508, 2014.

Piero Colonna and Teus van der Stelt. FluidProp: a program for the estimation of thermos physical properties of fluids. Energy Tecnology Section, Delft University of Technology, 2014.

Naoki Matsunaga, Tomohiko Hoshino and Akira Nagashima.ga. Critical Assessment of Thermophysical Properties Data of Combustion Gases for Calculating the Performance of Gas Turbine. Proceedings of International Gas Turbine Conference, pp. 321-328, 1983.

Carl Yaws. Chemical Properties Handbook. McGraw-Hill Education, 1999.

Sylvain Quoilin, Adriano Desideri, Jorrit Wronski, Ian Bell and Vincent Lemort. ThermoCycle: A Modelica library for the simulation of thermodynamic systems. Proceedings of the 10th International Modelica Conference 2014.

Sylvain Quoilin. Sustainable Energy Conversion Through the Use of Organic Rankine Cyclesfor Waste Heat Recovery and Solar Applications. PhD thesis, University of Liege, Belgium, 2011.

Francesco Casella, Tiemo Mathijssen Piero Colonna and Jos van Buijtenen. Dynamic modeling of organic rankine cycle power systems. Journal of Engineering for Gas Turbines and Power, 135(4), 042310, 2013.

Daccosim NG: co-simulation made simpler and faster

José Évora Gómez[1] José Juan Hernández Cabrera[2] Jean-Philippe Tavella[3] Stéphane Vialle[4]
Enrique Kremers[5] Loïc Frayssinet[6]

[1]Monentia SL, Spain, jose.evora@monentia.com
[2]SIANI, Spain, josejuanhernandez@siani.es
[3]EDF Lab Paris-Saclay, France, jean-philippe.tavella@edf.fr
[4]CentraleSupélec - University Paris-Saclay & LRI, France, {Stephane.Vialle}@centralesupelec.fr
[5]EIFER, Germany, enrique.kremers@eifer.org
[6]CETHIL - BHEE, France, loic.frayssinet@insa-lyon.fr

Abstract

This paper introduces the last evolution of Daccosim co-simulation environment, with Daccosim NG developed in 2018. Main features of Daccosim NG are described: enhanced Graphic User Interface and Command-Line Interface, algorithm and mechanism of co-simulation, co-execution shell, software architecture designed for both centralised and distributed architectures, aggregation of a co-simulation graph into a Matryoshka FMU, and declarative language to design large scale co-simulation graphs. A new industrial use case in simulation of energetic systems is also introduced, and first performances of Daccosim NG on multi-core architectures are analysed.

Keywords: co-simulation tool, multithreaded execution, master algorithm, FMI standard, energy system, runtime performance

1 Introduction

The study of Smart Grids, which are intelligent energy systems enhanced by additional communication means and modern IT features, requires a complex analysis of many components considering different aspects. These aspects are amongst others, the demand, production (including renewable), stability of the power grid and flexibility assessment. This is the case for Electricité de France (EDF) and the European Institute For Energy Research (EIFER), where Smart Grids and, more in general, Multi-Energy System analysis are performed through simulations representing the power grids considering multiple aspects. To this end, EDF and EIFER are working in the development of simulation models.

For instance, there are teams working in the modelling and simulation of customers by representing how devices consume energy at their homes: fridges, stoves, washing machines, etc. The analysis of the energy demand of these devices also requires to study thermal dynamics, since many of these devices produce heat or cold. Besides of thermal dynamics, the sociotechnical behaviour of the customers must also be represented as they are the ones who operate the devices. There are also teams developing models for representing thermal gains and loses for houses, buildings, districts, etc. Other teams are dedicated to optimise the grid operation with massive renewable energy and storage units.

Some examples of these kinds of business models are ThermoSysPro, BuildSysPro, PlantSysPro, TelSysPro and EPSL. ThermoSysPro (Hefni et al., 2011) is a library devoted to the modelling and simulation of power plants and energy systems. BuildSysPro (Plessis et al., 2014) is designed to be used in several contexts including building physics research, global performance evaluation, technology development and impact assessment. PlantSysPro is devoted to industrial processes like hot water system. TelSysPro is a new Modelica library able to model the impact of telecommunication networks on complex systems from failure/repair rate of components and stochastic latency.

These teams develop their models using the tool that is the most appropriate according to their work habit or affiliation. There are many tools or programming languages that can be used for developing these models: Anylogic (Borshchev, 2013), Dymola (Elmqvist et al., 1996), Matlab (Guide, 1998), Java (Gosling et al., 2014), Python (Rossum and al., 2007), etc. So, it happens very often that teams want to collaborate by making their models interoperable with others. This is challenging since models are developed in different tools. At this level, the interoperability challenge is double: syntactic and semantic (Hernandez et al., 2016).

The syntax challenge consists in being able to technically communicate models that are developed in different tools. For instance, this problem is equivalent to two people trying to speak when they do not have a common language. The semantic problem has several axis when talking about data exchange between two models: meaning of the words, units that are used, data types, etc. The most common semantic problem in models communication is to have different words to express the refer to the same concept.

The syntactic problem is addressed in FMI (Blochwitz et al., 2011). FMI, the Functional Mock-Up Interface, is a tool-independent standard that supports both model exchange and co-simulation of dynamic models using a combination of XML-files and compiled C-code. In this way, every model that is exported following this standard can be inter-operated with other exported models (FMU - Functional Mock-Up Unit). In the case of co-simulation, an FMU contains a definition of the model expressed in a standard format using XML and some binaries depending on the platforms to which the FMU is compatible with. These binaries are dynamic libraries that can be loaded by a Master Algorithm (MA) and have a standard interface that the MA knows. In this context, FMUs are considered as slave components that are commanded by MAs. A MA is a piece of software that coordinates the execution of several FMUs (slaves). This coordination mainly regards the data exchange between the different FMU models and their scheduling (the way time is advanced).

The construction of the MA to engineer co-simulations is the main challenge this paper addresses. To this end, we present a new version of Daccosim (Distributed Architecture for Controlled CO-SIMulation) (Galtier et al., 2015; Tavella et al., 2016). This new version is called New Generation (NG). Daccosim NG is a tool oriented to facilitate the construction of co-simulations. To this end, the MA behaviour can be easily defined using a very simple interface. Through the Daccosim graphical user interface (GUI), the user can drag-and-drop the different FMUs that are desired to use and, through arrows, the data exchanges among FMUs are defined. Data exchanges are made from an FMU output with a given name to an FMU input which may have a different name, addressing in this way the semantic interoperability challenge of having different wording to refer concepts of the reality. Additionally, Daccosim NG provides mechanisms to deal with other semantic problems as data types or units mismatch.

The new version, NG, is based on the same concept as previous Daccosim versions but re-written from scratch to get an industrialized code knowing that the first Daccosim experience was achieved without professional software developers. A new dramatically simplified installation procedure, a redesigned smarter interface and better performances are the strongest points of this new version.

This paper firstly introduces some co-simulation use cases that have been addressed using Daccosim NG. Then, Dacossim NG is presented showing its capabilities and explaining how co-simulations are executed. After presenting Daccosim NG, the novelties with respect to the old versions are highlighted. The performances of the new version are then compared with Daccosim 2017 and the conclusions and roadmap are discussed.

2 Co-simulation use cases

2.1 Available use-cases in Daccosim NG

From 2018, several demonstration cases are supplied in Daccosim NG deliveries. Some are trivial examples while others are business-oriented use cases. All the referenced FMUs are license-free and have been built using the most recent version of Dymola both for 32-bit or 64-bit machines. In addition all the Modelica source models are supplied for a better understanding.

These demonstration use cases can be downloaded from the Daccosim website (Evora et al., b) regardless Daccosim NG releases. They are organised in three folders named 1-coinit-only, 2-academic, and 3-industrial.

Cases located in 1-coinit-only are co-initialization examples where only a starting point is calculated (no time integration). They illustrate how a system composed with two or more coupled FMUs can be initialised solving algebraic loops between FMUs using a Newton-Raphson algorithm. Incidentally, some cases also show how operators can be used as objects dropped in co-simulation graphs in addition to FMUs (section 3.3). One of these cases is more deeply detailed in section 3.2.1.

Cases located in 2-academic are academic examples illustrating different non-stiff and stiff cases, sometimes including internal events in FMUs. Among other interesting cases, we can emphasise on:

- A case defining a co-simulation graph with FMUs exported from ControlBuild and Papyrus coupled with Dymola FMUs (non-Modelica source models are not supplied)
- Theoretical cases illustrating the capability to define thousands of connections between two FMUs or to instantiate hundreds of times the same FMU
- Two other cases showing how a stochastic behaviour implemented in Modelica models can be useful at a system level

Lastly, 3-industrial folder is dedicated to industrial cases. At the time being, only one case is included in this folder but we intend to enrich it next with for example a distributed power flow.

The supplied use case is related to district heating and cooling energy in buildings. The system represents a district composed with 23 buildings (with only 2 adjoining walls) with inter-building long-wave radiation coupling and solar flux pre-processed per facade to account for shadings and reflections. These FMUs have been built with public components from the EDF BuildSysPro library (Plessis et al., 2014). It represents one of the four variants of a business case more deeply described in section 2.2.

2.2 Building heating and cooling power load at district scale use cases

The heating and cooling energy consumption of buildings is a critical target for current energy issues as its contribution to the overall energy consumption and related greenhouse gases emissions is dominant, while its saving potential is high (IPCC, 2014). Furthermore, district scale implementation offers advantage for the integration of renewable energy sources, particularly in buildings, and notably via Smart Grids.

Therefore, the modelling of the building heating and cooling power load at district scale is essential. However, it faces two main challenges: the computational cost, that becoming prohibitive when using detailed model, decreasing the temporal scale and increasing the spatial scale; and the lack of data at the district scale. To cope with these constraints, the level of detail of the models has to be adapted.

In order to quantify the adaptation suitability, the platform MoDEM (standing from Modular District Energy Model (Frayssinet, 2018)) has specifically been developed. This platform is able to generate building energy model at district scale, with different level of detail, automatically from geometrical data. The district scale model is made of Modelica building models[1], that are coupled (common wall and long-wave radiative heat exchanges), depending on the modelling variant, and co-simulated with Daccosim NG after being converted in FMUs.

The use cases correspond to a district of Paris, France, made of 23 buildings (with 2 adjoining walls)[2] for the following variants:

1. Model–the most detailed–considering inter-building long-wave radiation coupling and pre-processed solar fluxes computed per facade to account for shadings and reflections.
2. Model (1) but with a lower discretisation of the conductive heat problem (fewer equations).
3. Model (2) but without long-wave coupling and and specific solar fluxes (less external resources and no connection between FMUs, excepted for the adjoining buildings).
4. Model (3) but with a simplified model for the conductive heat problem (less equations).

These models were simulated for one year and a month, with a constant time step of 900 s.

The present models focus on heating and cooling but the FMI offers further opportunities to couple these models with energy system, occupant behaviour and energy network models, toward integrated district energy model.

2.3 Urban energy planning use cases

Urban planning use cases are dedicated to the simulation of the multi-energy system of one or more districts, up to a whole town or city, consisting usually in several hundreds or thousands of buildings. This use case is considered as a prospective evolution and application for Daccosim NG, and will thus not be directly analysed in this work. However, even if the time resolution of these models is rather low in comparison to the previous ones (1h-15min), it is a use case that has high requirements for scalability and thus parallelisation, as it replicates the number of buildings of the use case 2.2, "Building heating and cooling power load at district scale use cases", by a factor of 10-500. Therefore it has been identified as relevant, and the requirements and lessons learned by the prototypes being done in this direction on the Anylogic platform by EIFER are gently provided as inputs to drive and inspire the Daccosim NG development to support such applications.

[1]Using the BuildSysPro library.

[2]More information about the characteristics of the district can be found in previous reference.

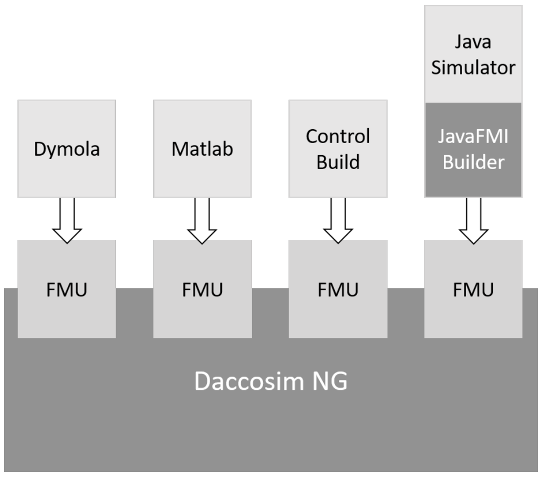


Figure 1. Daccosim interoperability example

3 Daccosim NG

Daccosim NG is an environment to develop co-simulation use cases supported by JavaFMI, a suite of tools for interoperability using the FMI standard (Evora et al., 2013). Daccosim allows the design, development and execution of co-simulation graphs, providing mechanisms to represent co-simulation graphs.

Daccosim NG is able to integrate different simulators exported as executable FMUs from various FMI-compatible tools. An exported FMU is a simulator contained in a FMU file, according to what is understood in the FMI standard. This way, any simulation developed in any programming language and deployed in any computer could be imported in Daccosim NG.

In figure 1, an example of a co-simulation integrating simulators from different sources is shown. On the upper part of the figure, it can be seen how Dymola, Matlab or ControlBuild simulators are integrated as FMUs. Same way Java or C++ codes can be exported as FMU thanks to the JavaFMI Builder tool (Evora et al., 2013).

Daccosim NG can be used through a graphical user interface (GUI) or a command-line interface (CLI). In subsequent sections, it is described how co-simulation graphs are defined, initialised and executed. Besides, some features are presented: the GUI, the CLI, FMI exposition, the matryoshka FMUs construction, the Daccosim graph declarative language. In a user's guide available with the tool distribution (Evora et al., a), more detailed information about the usage of Daccosim NG is available.

3.1 Co-simulation graph design

A co-simulation graph is composed of nodes and arrows that connect nodes. A connection defines which output variables of a source node are connected to which input variables of a target node. There are different types of nodes that can be included in a co-simulation graph:

FMU, operators, external inputs or external outputs:

- FMU: this node represents an FMU and it holds the file path, the variables used as input and output, and the initial values for variables and parameters.
- Operators: there are four operators: adder, multiplier, offset and gain. These operators allow to make calculations using outputs of other nodes providing the result in an output to be used. The adder and multiplier have two or more inputs and one output. The offset has only a fixed value and one input that are summed. The gain is defined with a fixed value that will multiply a given input. All of these nodes can work with Reals, Integers and Booleans.
- External inputs/outputs: these nodes allow to provide fixed values as input for other nodes (external input) or to store values provided by an output (external output). Both kind of nodes can have several variables. For instance, an external input will hold several outputs that can be used for other nodes.

Once nodes are defined, arrows can be established to define how variables are exchanged among them.

3.2 Co-simulation algorithm

Once the co-simulation graph is defined, the execution of the co-simulation can be done. The execution consists in the following steps: loading, co-initialisation (section 3.2.1), co-execution and exporting the results. The Daccosim engine executes each step of this method in parallel so that all cores of a machine are used, improving the performance (section 6).

In listings 1 the co-simulation algorithm is described. The procedure starts by opening the file in which the co-simulation is defined and loading the graph in memory. After opening this file and processing it, every FMU that is used is also loaded. In this way, co-simulation graph is ready for next steps: co-initialisation and co-simulation.

Listing 1. Co-simulation algorithm

```
public void execute() {
  loadGraph();
  coInit();
  export(currentTime);
  while (currentTime < stopTime) {
    currentTime += doStep();
    export(currentTime);
  }
  terminate();
}
```

After the co-initialisation process is executed (check section 3.2.1 for more information about this process), the initial values of the variables selected by the user for exportation are written into the output file for being analysed afterwards. Then the co-execution process starts. For this, the doStep method is called as many times as necessary until the stop time of the simulation is reached. The way the co-execution works is further described in section 3.2.2. After each successfully performed step, the values to be exported are once more written in the output file. Once the stop time is reached, the simulation is finished by terminating all FMUs and closing the exportation file.

3.2.1 Co-initialisation

One of the difficulties of the kind of co-simulations we are considering is the setting of consistent system-wide initial values for all the components. The Daccosim co-initialisation algorithm starts by building a global dependency directed graph for the connected variables of the FMUs. It uses the connections established by the user to find external dependencies between the outputs from source FMUs and the inputs from sink FMUs.

The key idea is that a topological sorting of the directed acyclic graph (DAG) naturally gives the order in which the variables must be initialised. Therefore, this led to study how to convert a generic directed graph into a DAG. The solution found is to build the graph of strongly connected components (SCC) corresponding to cyclic dependencies. The resulting graph in which each SCC has been contracted into a single vertex is a DAG. We use Tarjan's SCC algorithm (Tarjan, 1972) (used in many Modelica tools) to identify each SCC in the dependency graph (runs in linear time). Following the order obtained with a topological sorting on the contracted SCC graph:

1. for nodes which were not contracted, simply propagate their values
2. for nodes which were contracted (they correspond to cyclic dependencies), we solve the initialisation problem using an iterative algorithm called JNRA (Jacobian based Newton-Raphson Algorithm) inspired by traditional Newton-Raphson algorithms often used for electric load flow computation.

The example in figure 2 illustrates the co-initialisation of a system composed with two equations and two unknowns:

- equation1 model calculates x_2 from x_1 according to the equation: $2x_1^2 + 5x_2 = 42$
- equation2 model calculates x_1 from x_2 according to the equation: $x_1 - 6x_2 = 4$.

As it can be seen in figure 2, equation1.x2 depends on equation2.x1 while equation2.x2 depends on equation1.x2. In dotted lines it can be seen an algebraic loop where modifications on equation1.x1 affects equation1.x2 and modifications in equation2.x2 affects equation2.x1. Then, the co-initialisation procedure will then compute this graph to provide a consistent initial value to all variables. To do this, it will detect one SCC and, after several iterations, x1 and x2 will reach following values (x1 = 4.56, x2 = 0.09).

3.2.2 Co-execution

In Daccosim, it is possible to choose how the time is advanced when executing the co-simulation graph. There are two main categories of time steppers: constant and variable.

The constant stepper advances the simulation using a fixed step size. That is, when the simulation is stepped,

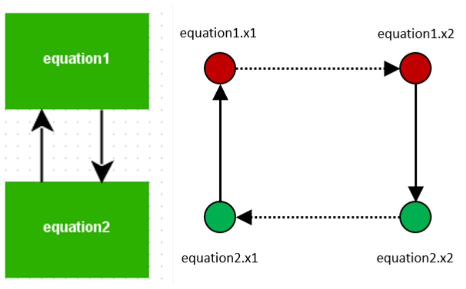


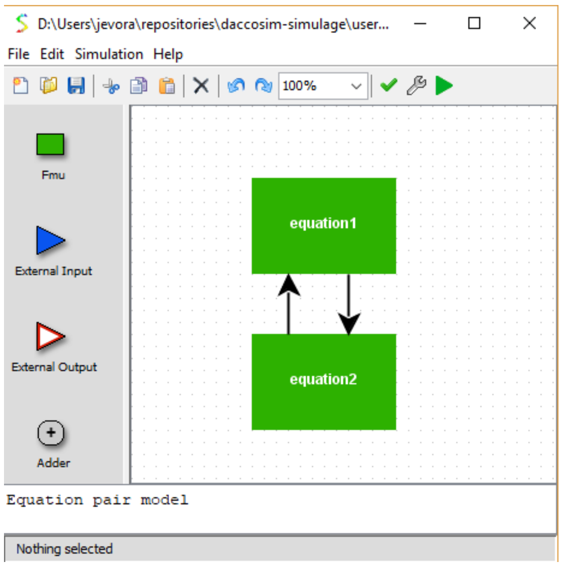
Figure 2. Calculation graph and dependency graph

the time that will be advanced is every time the same one. Constant stepper may not present a good computation/accuracy ratio: the choice of a small value for the step size results in a large number of computation steps, while a large value might fail to capture some variations in the simulated variables.

Depending on FMU capabilities, Daccosim NG also implements several variable stepping strategies. After the simulation of step i, each FMU examines its outputs and estimates how far they are from the exact value. Daccosim NG implements two algorithms which do that: one is based on the Euler's method and a second one is based on Adams Bashforth's method. Their principle is to store the values of the derivatives at consecutive communication points to infer an estimation at the next iteration. If the error is found to be tolerable, the engine will propose to perform the next step with a bigger step size. Otherwise, the last step would be cancelled and redone with a smaller step size value. The rollback is made possible since version 2.0 of FMI which introduced the notion of FMU state, allowing the serialisation of the FMU state before performing a simulation step, and the restoration of the saved state if necessary.

Suppose FMU A provides inputs for FMU B and initial step value is 10. At $t10$, it is decided that the step must be redone with a step size of 6 and it does not send its outputs to B. B, on the other hand, is satisfied with its outputs and only awaits updated inputs from A to perform its next step (from t10 to t20). When A reaches t6, it could send its outputs to B but they would not make much sense since B already advanced to t10 (and the next available outputs from A could be time stamped t12, which is not satisfactory either). To avoid this situation, all the FMUs adopt the same pace and they will all redo the cancelled step with the same new (smaller) step size. Conversely, if all the FMUs agree on a bigger step size, it will be used for the next steps.

It is also intended to implement another variable stepping method based on the concept of state quantization used in the Quantized State Systems (QSS) methods which are non-stiff QSS solvers of different orders described by Ernesto Kofman in many publications (Kofman and Junco, 2001; Kofman et al., 2001; Kofman, 2002). QSS adaptations for FMI standard are being designed in one work package of the French national project Modeliscale (2018-2020) leaded by Dassault Systèmes and whose


D:\Users\jevora\repositories\daccosim-simulage\user...
File Edit Simulation Help
100%
Fmu
External Input
External Output
Adder
equation1
equation2

Equation pair model

Nothing selected

Figure 3. Daccosim NG GUI

goal is to provide the ability to model and simulate with Modelica and FMI the behavior of very large energy systems.

3.3 Editor

This module provides a GUI to facilitate the design, development and execution of co-simulation graphs. The editor can be downloaded from (Evora et al., b). The editor is distributed in three different formats: exe files for running in Windows (32 & 64 bits exe files) and a jar file for execution in any operating system. All of these version require to have a JVM installed in the system. In any of these three formats, the editor can be launched by just double-clicking it (it does not require any installation or configuration).

In the GUI (figure 3), aside from the menu and the toolbar with the options to deal with the co-simulation graph, the palette and the canvas are the main components supporting the definition of co-simulation graphs (Evora et al., a). The palette contains all the possible nodes that can be set in the co-simulation graph (note that for space constraints, not all blocks are visible in the figure). These nodes can be dragged and dropped into the canvas.

Then, by dragging out from the centre of a node (source node), an arrow can be created when dropping the mouse in the target node. The variables that are to be exchanged in that arrow can be configured by double-clicking it or in the contextual menu: properties option. Each kind of node has also its own configuration which can be accessed in the same way: double-click on the node. In the case of an FMU node, the label of the node and the initial values of the FMU variables can be configured. For external inputs and external outputs, the variables to be connected to other nodes can be defined and their initial value set. In the case of the operators, the data type to work with (either Real, Integer or Boolean) and in the case of the adder and multiplier, the amount of inputs to receive can be chosen. In the case of the offset and the gain, the fixed value that will be added or multiplied to the input can be set.

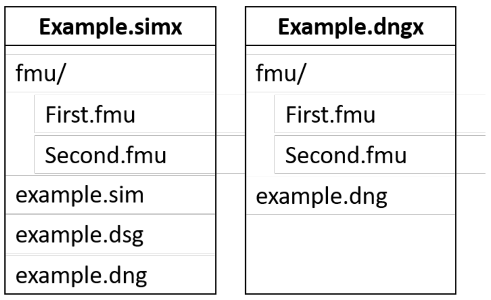


Figure 4. simx and dngx files

Below the canvas and the palette, there is a blank empty box in which the user can write a message to describe what purpose is the model and any other extra information. And below this box, there is a bar state in which some information is contextually displayed to interactions made by the user.

Up in the toolbar, commands are defined in menus to create a new co-simulation graph, to open it (see section 4.1 to know more about file formats) or to save it. It is also possible to cut, copy or paste parts of the graph and to undo or redo some of the actions made. Then, once the graph is defined, it can be validated, configured (start and stop times, variables to export, etc) and executed.

For more information about how to use the editor, please read the user's guide (Evora et al., a).

4 Other Features

4.1 Shell

This module wraps the core and provides a command line interface (CLI) to run co-simulation graphs (Evora et al., a). The shell can be downloaded from (Evora et al., b). This utility allows users to develop script files to run co-simulation graphs in a batch mode. The main argument to provide to this CLI is the path to the file in which the co-simulation graph is stored. With Daccosim NG there are two main file formats:

- simx: this is an archive file (zip) containing a folder named fmu with the fmu files used in the co-simulation graph as well as the sim, dng and dsg files (figure 4). The sim, dng and dsg files contain representations of the co-simulation graph in different formats. sim file contains the graph including the visualisation information to be presented in the Editor. dng contains the graph in the declarative language (section 4.4). dsg is a serialisation of the graph in json format. This is the one that is effectively used as a graph representation for starting the co-simulation execution.
- dngx: this archive has the same structure and content than the simx but sim and dsg files are not present (figure 4). It allows to create a runnable file in which the graph is defined using the declarative language (section 4.4). The idea of this format is to allow other tools or users with a text editor to create a co-simulation graph compatible with Daccosim NG by describing it in a dng file. This is especially interesting to develop large-scale co-simulation graphs.

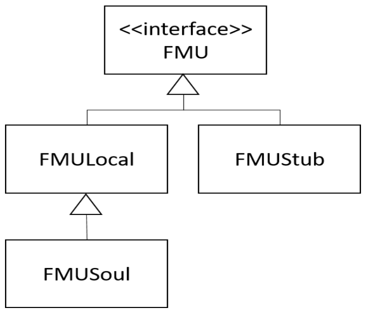


Figure 5. The FMU interface is used by the execution engine. This way the engine works independently from where FMUs are being executed

This CLI has also the possibility of parameterizing some of the features of the execution. The idea is to make simpler the GUI (editor) by avoiding this kind of parameterization and let for advanced users to play with them using the CLI. For instance, one of the parameters that can be changed in CLI is to run the co-simulation in singlethread or multithread. In the editor, this option is not available and all co-simulations are run in multithread by default.

4.2 Designed for distributed executions

FMU nodes are normally performing costly processes each time they are called in the doStep method (Blochwitz et al., 2011). For this reason, the execution engine is also prepared to be run in a distributed environment allowing the execution of large-scale co-simulation scenarios. This is made thanks to the use of abstraction mechanisms so that the execution engine does not need to be aware of where each FMU is being physically executed. To this end, every time the engine interacts with an FMU, it uses an abstracted interface. Based on this interface, there are three implementations: FMULocal, FMUStub and FMUSoul. First one uses the FMU files from the filesystem (normal case in a single machine). Second one uses a connection to a Java Message Service (JMS) to interact with an FMU that is being remotely executed. Third, and last one, is the representation in the remote computer of the FMU. This receives the queries from the FMUStub and acts accordingly (figure 5).

In figure 6, the communication between different machines running a distributed simulation is exemplified. In this example, there are three machines. In the first one, an instance of Daccosim core is responsible for coordinating a distributed execution. The execution engine of this instance uses the FMU interface to communicate with the three FMUs to be coordinated. Two of them are being executed remotely and one locally. However, as the engine only depends on the interface, these details of where they are being executed are not important for its execution. Whenever a command is asked by the engine, the FMUStub will communicate to the FMUSoul to perform

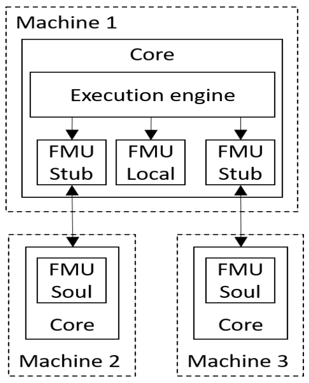


Figure 6. The communication of the execution engine with the remote fmu (soul) is made through the stub

the command providing the answer back to the engine. Note that, despite it is not represented, communications are made through a JMS. The use of JMS gives flexibility to design different distribution architectures to support large scale co-simulations.

4.3 Matryoshka

This feature(Galtier et al., 2017) allows to wrap a co-simulation graph into an FMU for being used either in other co-simulation environments like Dymola or in Daccosim NG itself (Evora et al., a). The co-simulation graph that is stored inside the FMU is seen as a single FMU when opened by other tools. Every time a master algorithm uses this FMU for any purpose, the Matryoshka FMU will dispatch the command to the corresponding inner FMU or FMUs.

For instance, if the master commands a simulation advancement through a doStep to the Matryoshka FMU, this FMU will perform the doStep for all the FMUs contained and will exchange the values between the FMUs as described in the co-simulation graph. At the same time, it is possible to embed one Matryoshka FMU inside another co-simulation graph and export this graph into a Matryoshka FMU having in this way several levels of FMUs embedded. Exported FMUs will be beneficial (Galtier et al., 2017):

- FMU can be imported into any FMI compliant simulation tool also able to handle non-FMI components with which Daccosim NG is not able to directly interact.
- Taking advantage of Daccosim NG efficient, multi-threaded, step-size control solution helps simulating faster larger models within traditional monothreaded simulation tools.
- Initialization of complex graphs is taken care of within the Matryoshka thanks to Daccosim NG generalized co-initialization algorithm.
- A complex simulation graph can be reused directly without having to re-write anything and with no risk of disclosing industrial and intellectual property.
- The co-simulation process can be finely tuned: when typically a solver only uses one accuracy objective for the whole model, Daccosim NG allows the user to define different tolerance values for every output and internal variable of each FMU.

Besides, new features have been developed:

- Added information in the modelDescription file about dependencies of external outputs to external inputs.
- Continuous inputs extrapolation and output derivatives provision.

These improvements have been accompanied by a significant FMU size reduction of the order of 3MB. They will be completed next year with two new features: improvement in the performance when loading multiple instances and the capability to make rollbacks.

4.4 Declarative language

The declarative language implemented in Daccosim NG allows the user to define a co-simulation graph on a text editor or to automatically generate it through a program (Evora et al., a). To do so, a domain-specific language has been designed to simply define a co-simulation graph. Enjoying a feedback from user experiences, this language has been dramatically simplified regarding the previous version named DSL in Daccosim 2017. As it can be seen in listing 2, this language is very simple and can be easily understood. Its purpose is to create very wide graphs that cannot be modelled in the GUI, in which there are hundreds of interconnected nodes exchanging thousands of variables. This feature allows pre-processing tools to develop compatible models to be executed in Daccosim NG. Note a textual form of a co-simulation is automatically generated from the GUI once a valid graph is defined. Conversely, a valid textual form of a co-simulation can be opened in the GUI and the corresponding graph is automatically drawn.

Listing 2. Declarative language example

```
FMU equation1 "fmu/equation1win3264.fmu"
Output equation1 x2 Real
Input equation1 x1 Real
FMU equation2 "fmu/equation2win3264.fmu"
Output equation2 x1 Real
Input equation2 x2 Real
Connection equation1.x2 equation2.x2
Connection equation2.x1 equation1.x1
CoInit 100 1.0E-5
ConstantStepper 1.0
Simulation 0.0 10.0
```

Listing 2 expresses the co-simulation graph defined in figure 2 using this language. There are two FMUs declarations followed by the label and the path to the file. Then the outputs and inputs to be connected are described for each of the FMUs and the connections defined. Finally, the co-init, stepper method (constant step with step size 1.0) and simulation start and stop times are defined.

5 Why NG?

In this last year and a half, we have rebuilt Daccosim from the scratch. To this end, we have gotten rid of some strong dependencies that made difficult the evolution and use of the software. Prior version of Daccosim (2017) required the installation of the Eclipse IDE (Eclipse, 2007) in a specific version with specific plugins. Then, the source code of the project had to be imported into it and compiled to be run. This was a tedious process that made harder to the users their initial steps in Daccosim. In the new version, just by downloading it and making a double click, the user can start working in the design of a co-simulation graph using a very comfortable GUI.

Since the GUI is detached from Eclipse IDE, the interface is not contaminated by the style in which Eclipse displays buttons, views, etc. This GUI is tailor made to focus on the design of the co-simulation graph and its execution. This way, the result is a very clean interface with two main components, palette and canvas, in which the graph is designed. The projects view has also been deleted as Daccosim NG can be opened as many times as necessary holding a project on each instance.

The performance has also been improved in many aspects. Previous Daccosim versions used the co-simulation graph designed by the user to generate tailor made Java code for executing the graph. For this reason, co-simulations made by the user were conceived as ”projects”, as they had the files containing the graph definition, the fmus and the generated code for execution. In this new version, co-simulation graphs are read an interpreted so that no Java code is generated to execute a specific graph. This makes the process much faster and lighter, specially for wide co-simulations. This way, all the information that concerns a co-simulation is stored in a single simx file (section 4.1). The performance in runtime is also compared in the following section 6.

From the point of view of the development, the most important achievements are the removal of strong dependencies (Eclipse and its plugins) and the code maintainability which will make easier the correction of bugs and the evolution of the software.

Finally, to make easier the experience of the user, a daccosim-windows-installer has been built for a complete and simple installation on windows 32-bit or 64-bit machines. This is available at (Evora et al., b).

6 Performance on parallel machines

6.1 Comparing previous and new Daccosim

In order to exhibit the performances of Daccosim NG, we run the 4 variants of the case building energy system with 23 FMUs described in section 2.2. The runs have been done on the same 4-core Windows machine both with Daccosim 2017 (the previous version of Daccosim) and with the new Daccosim NG (also called Daccosim 2018).

In order to get reliable results (summarized in table 1), each variant has been done 3 times with strictly the same co-simulation conditions (start time, stop time, stepping parameters,...). Based on the average of gotten time durations, a speedup has been calculated ($Speedup = T_{2017}/T_{2018}$) showing the performance improvement of Daccosim NG (2018).

Variant	Model 1	Model 2	Model 3	Model 4
Dac. 2017	6150 s	1666 s	1660 s	2075 s
Dac. 2018	5217 s	1574 s	1562 s	1515 s
Speedup	1.18	1.06	1.06	1.37

Table 1. Performances of Daccosim 2018 *vs* Dacossim 2017

6.2 Benchmark on dual-processor Linux machines

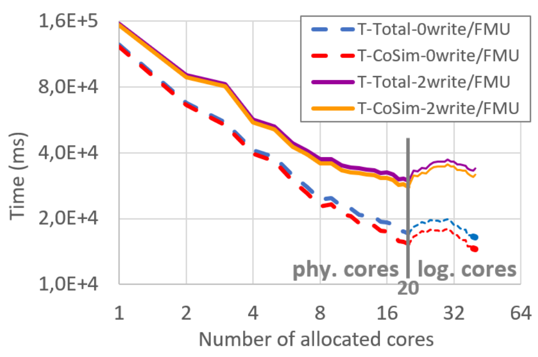
In order to evaluate the performances of Daccosim NG on a parallel multi-core machine, we needed a test application with a large number of FMUs to spread on computing cores, and with a significant total amount of computations and disk IO. But most of our business use cases involve FMUs handling resource files (e.g. temperature time series), and unhappily when exported from Dymola Linux these FMUs are not working correctly unlike with export from Dymola Windows. We go on investigations to identify the issue either at the Dymola side or as a side effect in Daccosim NG. For this reason, we fell back to the demonstration case *multiFMU* supplied beside the Daccosim NG distribution which was originally composed of 1000 instances of the well-known *stairBouncingBall* model (Kofman, 2004). In order to mix logical instances and physical FMUs, we have duplicated the original FMU 10 times to get 10 different FMUs and then we have instantiates each of them 100 times. However, all these FMUs model independent balls, and do not consume inter-FMU communication times. About IO, we have defined two variants, the first one without any FMU output saved on disk and the second one with 2 outputs per FMU written after each step integration.

This test case has been run on a dual 10-core Intel Xeon Silver 4114 at 2.2 GHz (*Skylake* architecture), with 96 GBytes of RAM. This machine is part of a PC cluster of CentraleSupelec, managed with the OAR[3] environnement. OAR allows to allocate an entire PC or only a required number of its cores, and runs all threads of an application only on the allocated cores. We used this mechanism to test our application from 1 up to 20 physical cores, and then up to 40 logical cores on our dual 10-core Xeon machine.

When running our test case for 5000 steps on one core of our test machine, the variant saving 2 outputs per FMU generates an output file of 110 *MBytes* and elapses on 160 s We have chosen this configuration with easy to measure execution times (not too long but significant times).

Figure 7 left shows the co-simulation execution times as a function of the number of allocated cores, in logarith-

[3] s://oar.imag.fr/



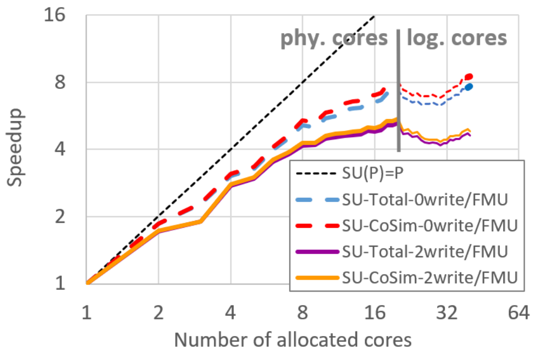


Figure 7. Execution times and speedup of *multiFMU* Daccosim NG benchmark on *dual 10-core Xeon Silver 4114* machine

mic scale. A straight line with a -1 slope would mean a perfect decrease of the execution time. Full lines illustrate performances of the benchmark with 2 FMU output writing per time step, while dashed lines are related to co-simulation runs without any IO (no FMU output was saved on disk). We can observe a very regular and very good decrease of co-simulation and total execution times from 1 up to 10 physical cores, and a little bit less good decrease from 10 up to 20 physical cores when using the second CPU of the machine. As expected, execution time is lower and exhibit better decrease when no FMU output are written on disk (no IO). When writing all FMU outputs on disk at each time step, execution time is higher but still exhibits a significant and almost regular decrease.

Beyond the 20 physical cores of our machine, the threads are distributed also on the *logical* cores. As each physical core hosts two logical cores, when allocating $20+n$ cores (with OAR), n physical cores host two threads and $20-n$ host only one thread. Beyond 20 cores this load unbalance leads to an execution time increase, as illustrated on figure 7 left. However, when allocating 40 cores (all virtual ones) load balancing is achieved again and performances appear a little bit better than on 20 physical cores when no FMU output is written on disk (dashed lines). At the opposite, when writing 2 outputs per FMU on disk (full lines) it appears better to use only the 20 physical cores. These IO remain sequential and partially overlapped with the computations (depending on the OS), but disturb parallel computations.

A single-threaded version of Daccosim NG (running on one core), has exhibited execution times very close to our multithreaded version run on one core. Then we can define the speedup achieved by Daccosim NG running on several cores: $SU(p) = T_{single}/T_{multi}(p) \approx T_{multi}(1)/T_{multi}(p)$. Figure 7 right shows this speedup, and we get:

$$\begin{aligned} SU^{max}_{0write/FMU} &= SU_{0write/FMU}(40) = 7.7 \\ SU^{max}_{2write/FMU} &= SU_{2write/FMU}(20) = 5.2 \end{aligned}$$

Considering only the experiments with FMU output saving (more realistic use case), experiments have shown the execution time of the multithreaded implementation of Daccosim NG has *scaled* on our benchmark. An almost regular decrease of the execution time has been measured up to all physical cores of our dual 10-core Xeon machine. Moreover a significant speedup close to 5.2 has been achieved compared to a sequential execution. The current multithreaded implementation of Daccosim NG appears ready to be the kernel of a distributed version on PC clusters and clouds.

7 Conclusion and roadmap

In 2018, Daccosim NG is more robust, faster and simpler to use than the previous version that was simply a proof of concept for EDF to make sure that co-simulation is helpful for the simulation of wide energetic systems.

To further improve it, the Daccosim NG team intends to implement before the end of 2018 a new major version including some Matryoshka evolutions (section 4.3) and QSS-inspired variable stepping implementation (section 3.2.2). In addition, as EDF is participating to the Modelica Association Project FMI, Daccosim NG is candidate to implement the new hybrid co-simulation feature under discussion in the WG "clock hybrid co-simulation" in order to accurately detect internal FMU events from a proposal pushed by EDF in 2017.

We will also think about an implementation of the System Structure and Parameterization Standard (SSP)(Köhler et al., 2016) as a future standard way to export/import co-simulation graphs in Daccosim NG. Additionally, and as mentioned in section 2.3, urban energy planning requires large scale simulations of hundreds of buildings, which can deliver valuable simulation results taken as decision aid for large infrastructure investments by municipalities. The following challenges arise in this context:

1. Multiple layers: different energy vectors addressed (heating, cooling, electricity, gas).
2. Bottom up simulation: large number of buildings that on its own have an individual behaviour.
3. Connection through networks: energy flows are distributed via networks which have to be included and are a model in itself to couple other models.
4. Data uncertainty and unavailability: in early planning stages, many parameters have not yet been fixed, and furthermore, projections over several decades allow for important assumptions in the development of environmental parameters (future evo-

lution of energy or fuel prices, etc).

The representation of such a complex system requires and efficient coupling of different models, to avoid constructing unmanageable complicated model couplings. Daccosim NG shows excellent prerequisites to support these kind of simulations, especially on points 1 & 2. A use case in which Dymola and Anylogic energy system models are co-simulated, is thus envisaged to proof the feasibility of Daccosim NG towards requirement 3 & 4.

Finally, to go further than previous experiments in 2017 (Vialle et al., 2017), we also intend to co-simulate again very wide complex systems to illustrate the new constraints EDF has to cope with in the context of the energy transition and the renewal of the energy market landscape. This will be done as soon as the problem we have encountered with Dymola FMUs exported from Linux is solved.

Acknowledgement. This work was partly supported by the Paris-Saclay region through the FUI research project "Modeliscale".

References

T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mauß, M. Monteiro, T. Neidhold, D. Neumerkel, and al. The functional mockup interface for tool independent exchange of simulation models. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*, 2011.

A. Borshchev. *The big book of simulation modeling: multimethod modeling with AnyLogic 6*. AnyLogic North America Chicago, 2013.

IDE Eclipse. Eclipse foundation, 2007.

H. Elmqvist, D. Brück, and M. Otter. Dymola-user's manual. *Dynasim AB, Research Park Ideon, Lund, Sweden*, 1996.

J. Evora, J-Ph. Tavella, JJ. Hernandez, and S. Vialle. *Daccosim NG User's Guide*. EDF, Monentia, CentraleSupelec, a.

J. Evora, J-Ph. Tavella, JJ. Hernandez, S. Vialle, and E. Kremers. Daccosim ng web page, b. URL `https://bitbucket.org/simulage/daccosim`.

J. Evora, JJ. Hernandez, and O. Roncal. Javafmi. *URL https://bitbucket. org/siani/javafmi*, 2013.

L. Frayssinet. *Adapting building heating and cooling power need models at the district scale*. PhD thesis, INSA de Lyon, 2018.

V. Galtier, S. Vialle, C. Dad, J-Ph. Tavella, J-Ph. Lam-Yee-Mui, and G. Plessis. Fmi-based distributed multi-simulation with daccosim. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. Society for Computer Simulation International, 2015.

V. Galtier, M. Ianotto, M. Caujolle, R. Corniglion, J-Ph. Tavella, J.E. Gómez, JJ. Hernandez, V. Reinbold, and E. Kremers. Experimenting with matryoshka co-simulation: Building parallel and hierarchical fmus. In *12th International Modelica Conference*, 2017.

J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley. The java language specification, java se 8 edition (java series), 2014.

MATLAB User's Guide. The mathworks. *Inc., Natick, MA*, 5, 1998.

B. El Hefni, D. Bouskela, and G. Lebreton. Dynamic modelling of a combined cycle power plant with thermosyspro. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical University; Dresden; Germany*, 2011.

JJ. Hernandez, J. Evora, and J-Ph. Tavella. Semantic interoperability in co-simulation: use cases and requirements. *European Simulation and Modelling Conference 2016 at Las Palmas de Gran Canaria, Spain*, 2016.

IPCC. *Climate change 2014: mitigation of climate change: Working Group III contribution to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, 2014.

E. Kofman. A second-order approximation for devs simulation of continuous systems. *Simulation*, 78(2), 2002.

E. Kofman. Discrete event simulation of hybrid systems. *SIAM J. Sci. Comput.*, 25(5), May 2004. ISSN 1064-8275.

E. Kofman and S. Junco. Quantized-state systems: a devs approach for continuous system simulation. *Transactions of The Society for Modeling and Simulation International*, 18 (3), 2001.

E. Kofman, J. S. Lee, and B. P. Zeigler. Devs representation of differential equation systems. review of recent advances. *Proceedings of ESS'01*, 2001.

Jochen Köhler, Hans-Martin Heinkel, Pierre Mai, Jürgen Krasser, Markus Deppe, and Mikio Nagasawa. Modelica-association-project "system structure and parameterization"– early insights. In *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*, number 124, pages 35–42, 2016.

G. Plessis, A. Kaemmerlen, and A. Lindsay. 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*, 2014.

G. Van Rossum and al. Python programming language. In *USENIX Annual Technical Conference*, volume 41, 2007.

R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2), 1972.

J-Ph. Tavella, M. Caujolle, S. Vialle, C. Dad, Ch. Tan, G. Plessis, M. Schumann, A. Cuccuru, and S. Revol. Toward an accurate and fast hybrid multi-simulation with the fmi-cs standard. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*. IEEE, 2016.

S. Vialle, J-Ph. Tavella, C. Dad, R. Corniglion, M. Caujolle, and V. Reinbold. Scaling FMI-CS Based Multi-Simulation Beyond Thousand FMUs on Infiniband Cluster. In Modelica Association, editor, *12th International Modelica Conference 2017*, Czech Republic, May 2017.

Dynamic Parameter Sensitivities: Summary of Computation Methods for Continuous-time Modelica Models

Atiyah Elsheikh

Mathemodica.com , Egypt & Germany , Atiyah.Elsheikh@mathemodica.com

Abstract

Applications of Sensitivity Analysis (SA) encouraged several Modelica platforms to independently provide facilities for externally computing Dynamic Parameter Sensitivities (DPS). FMI specifies an optional function call for evaluating directional derivatives. On the other hand, mathematical foundation for uniform representation of DPS at the Modelica language level has been established. This has resulted in a platform-independent approach demonstrated through example libraries. The paper neutrally hints that already conducted efforts may converge to the integration of language facilities for DPS without neglecting to mention many mathematical difficulties. Surprisingly, many of what could be thought to be algorithmic obstacles have intuitive solutions along a minimalist implementation approach.

Keywords: algorithmic differentiation, parameter sensitivities, sensitivity analysis

1 Introduction

1.1 Motivation to DPS

In (Wiechert et al., 2010) one reads:

> Simulation tools not only perform numerical solutions based on the system equations but also assist the modeler in systems analysis. Doubtlessly the most important systems analysis tool is SA ... SA is required for parameter fitting, statistical regression analysis, experimental design, and metabolic control theory.

Analogously, common guidelines for model-based studies recommend SA to be performed in order to assist the validity of the conclusions, for example demanded from:

1. Impact Assessment Guide, European Commission
2. Guidance on the development, evaluation and application of environmental models, US Environmental Protection Agency

Hence, facilities for SA are crucial for modelers if offered by a modeling language or a simulation platform.

While many methods of SA are based on numerical approaches (Saltelli et al., 2004) their applicability on typically large-scale Modelica models is questionable. As an attractive alternative, analytical derivatives of model outputs w.r.t. model inputs can be exploited. Derivative-based approaches usually lead to superior results in terms of accuracy and computational complexity e.g., derivative-based global SA methods (Kucherenko and Iooss, 2016). In Modelica-based terminologies, Dynamic Parameter Sensitivities (DPS) are sought.

1.2 Applications of DPS

Despite of many applications of SA DPS enable, there are not so many works conducted within the Modelica community exploiting DPS. One possible reason for that (up to the author knowledge) there is no comprehensive literature focusing on general applications of DPS. Instead, their applications are splintered among thousands of books and articles many of which belong to Chemical and Bio-Engineering domains. In conjunction with this paper, (Elsheikh and Kucherenko, 2019) provides a new classification and initial summary of applications of DPS. In this technical report[1], three families of application samples are categorized:

1. *Modeling-oriented applications:*
   Applications benefiting from the presence of DPS at the model level without the need of mathematically sophisticated post processing or leaving the GUI of the simulation platform such as control coefficients (Fell, 1992), local SA, parameter sweeping studies, model simplification and error analysis
2. *statistical-oriented applications:*
   Statistical tools benefiting from DPS for improved efficiency in terms of accuracy and computational complexity such as regression analysis, global SA, uncertainty analysis and identifiability analysis
3. *optimization-oriented applications:*
   high-level optimization problems with objective

[1]The technical report is subject to continuous modification from the author. Interested readers are welcome to contribute.

functions expressed in terms of DPS such as experimental design, optimal design and parameter estimation in conjunction with identifiability analysis

Figure 1 demonstrates an application of a DPS-enabled simulation. The underlying model is capable of additionally computing DPS. Consequently it was possible to apply parameter sweeping studies using generic models within the PSTools library.

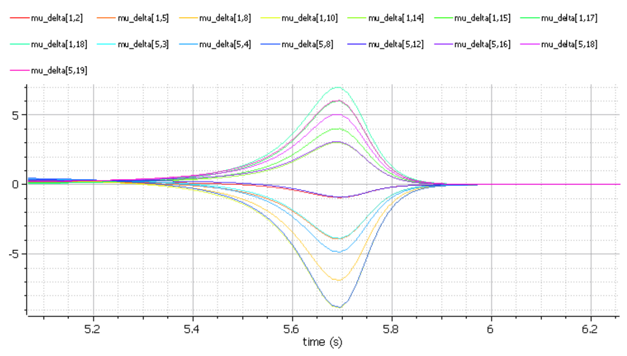


Figure 1. Parameter sweeping study based on one DPS-enabled simulation with two parameters. Instead of running multiple simulations, Taylor series expansion is exploited

1.3 Methodologies for Computing DPS

Assume that a given continuous-time Modelica model equivalently corresponds to an explicit ODE:

$$\dot{x} = f(x,q,t) = 0 \quad , \quad x_0(q,t_0) = x_0(q) \qquad (1)$$

where:

- t : time
- $q \in R^{n_q}$: set of model parameters
- $x(q,t) \in R^{n_x}$: set of state variables

The chosen parameters (say $p \in R^{n_p}$) w. r. t. which derivatives are sought are referred to as *active parameters*. This work is summarizing some of the available methods for computing DPS

$$\frac{\partial x}{\partial p}(p,t) \in R^{n_x \times n_p}$$

These methods are:

1. Finite Difference (FD) methods (Section 2): straightforward to implement with Modelica but subject to serious accuracy issues
2. Specialized solvers (Section 3): provided by specific simulation environments usually accessible by external scripting capabilities
3. Equation-based Algorithmic Differentiation (AD) (Section 4): a platform-independent approach allowing DPS to be present at the language level but requires additional implementation efforts

Section 5 discusses a potential enhancement to the Modelica language by allowing the operator *der* to take a second argument as a model parameter. The discussion includes some expected obstacles with a minimalist implementation approach highlighted in Section 6 and an outlook given in Section 7.

2 Finite Difference Methods

2.1 Implementation

A straightforward way to compute DPS is realizable by a first-order FD method. Assuming that simulation of Equation (7) leads to the approximated solution:

$$x_i(p,t_k) = x_{i,k}(p) \qquad (2)$$

$$\text{for} \quad i \in \{1,2,...,n_x\} \quad \& \quad k = 0,1,2,..$$

DPS are evaluated by post-processing additional n_p simulations each with one slightly modified active parameter:

$$\frac{\partial x_i}{\partial p_j}(t_k,p) \quad \approx \quad \frac{x_{i,k}\left(p+e_j\delta_j\right) - x_{i,k}\left(p\right)}{\delta_j} \qquad (3)$$

where:

$$\delta_j = \xi_j \cdot p_j, \quad \xi_j : \text{perturbation factor}$$

and $e_j \in R^{n_p}$: the j-th unit vector. However, utilizing Modelica capabilities, DPS can be implemented via one simulation as follows:

```
model FDModel
 parameter Real zeta = 0.01 "perturbation";
 parameter Real p1=3.0,p2=0.3,p3=...;

 MyModel M (p1=p1,p2=p2,...);
 MyModel M1(p1=p1*zeta+p1,
            p2=p2,
            ...);
 MyModel M2(p1=p1,
            p2=p2*zeta+p2,
            ...);
 ...
 Real dxdp[nx,np] "DPS";
 ...
equation
 dxdp[1,1] = (M1.x1-M.x1) / (p1*zeta);
 dxdp[1,2] = (M2.x1-M.x1) / (p2*zeta);
 ...
 dxdp[2,1] = (M1.x2-M.x2) / (p1*zeta);
 dxdp[2,2] = (M2.x2-M.x2) / (p2*zeta);
 ...
end FDModel;
```

2.2 Performance

Theoretically, for a Modelica model with say n_x nontrivial equations and n_p active parameters, the previous implementation results in a model with $n_p(n_x+1)$ nontrivial equations. The current computation paradigms for Modelica simulation environments (at least for the published ones) imply that the above model translates to one single block of an ODE / a DAE system of equations. This causes performance drawbacks when considering large number of active parameters. One may rather attempt to consider fewer numbers of active parameters and additional models. However, by extending BLT-based speed-up techniques (Cellier, 1991) to ODEs / DAEs level rather than only algebraic equations, advanced compiler strategies may allow independent simulation of smaller blocks of ODEs / DAEs, even in parallel. Hence, run-time performance of FD-methods should not be the main concern but the accuracy. As mathematically justified in (Elsheikh and Wiechert, 2012), the study shows that FD is non-reliable in the context of large-scale nonlinear dynamics. Hence FD, in the way implemented in this subsection, should not be the chosen approach for computing DPS.

2.3 Applying FD within the PSTools Library

Employing FD for computing DPS can still be exploited only as a first experimental step, in particular if advanced FD strategies and more accurate formulas are employed. This is also useful to validate other approaches for computing DPS. The PSTools library provides modelers capabilities for computing DPS of arbitrary Modelica models. Given a Modelica model *M*, the first step is to parameterize the model under consideration as follows[2]:

```
model ParM
  extends PSTools.Utilities.Parameterized(
    NP = 2,
    _P = {0.4, 0.5 / 3600},
    PNAME = {"p1", "p2"},
    NX = 4,
    _X = {_M.x1, _M.x2},
    XNAME = {"x1", "x2"});
protected
    M _M(p1 = _P[1], p2 = _P[2]);
end ParM;
```

In this way, the active parameters and the significant variable of the model *M* are specified. Alternatively, exploiting enumeration classes is also possible:

```
model ParM
  extends
    PSTools.Utilities.EnumParameterized(
      redeclare type ActivePars =
                        enumeration(p1,p2),
      _P = {0.4, 0.5 / 3600},
      redeclare type SignificantVars =
                        enumeration(x1,x2));
protected
  M _M(p1 = _P[ActivePars.p1],
       p2 = _P[ActivePars.p2]);

equation

  _X[SignificantVars.x1]  = _M.x1;
  _X[SignificantVars.x2]  = _M.x2;

end ParM;
```

[2]other styles other than declaring the model under consideration in a protected section are also realizable

Once a model under consideration is a subclass of *PSTools.Utilities.Parameterized*, it is straightforward to employ advanced capabilities within the PSTools library for computing DPS as follows:

```
model FDParM
  PSTools.PS.FD.CD2
    PS(redeclare replaceable model
         ParModel =  ParM);

  Real g_x1_p1, g_x1_p2, g_x2_p1, g_x2_p2;

equation

  g_x1_p1 = PS.g_x[1,1];
  g_x1_p2 = PS.g_x[1,2];
  g_x2_p1 = PS.g_x[2,1];
  g_x2_p2 = PS.g_x[2,2];

end FDParM;
```

In this way DPS are computed using a central difference formula of order two. Figure 2 demonstrates the trajectories of DPS for some chosen significant variables w.r.t. some active parameters for a Modelica model in the Bio-engineering domain with 10 significant variables and 9 active parameters resulting in 90 trajectories for DPS.

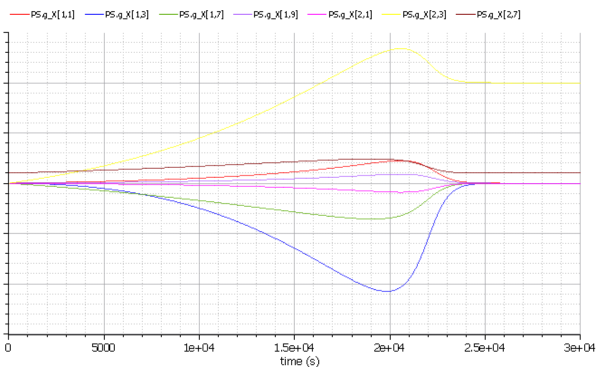


Figure 2. Some trajectories of DPS of a Model from Bio-Engineering domain

3 Specialized Solvers

3.1 Approach

There are specialized numerical solvers, e.g. Sundials package (Hindmarsh et al., 2005) or DSPACK (Petzold et al., 2006) that can be employed for computing DPS. This is done by numerical integration of the sensitivity system composed of the ODE (1) together with sensitivity subsystems:

$$\dot{x}_p = f_x x_p + f_p \quad , \quad x_p(t_0) = \partial x_0(p)/\partial p \tag{4}$$

obtained by differentiating ODE (1) w.r.t. *p*. OpenModelica and Wolfram SystemModeler exploit such solvers with the help of external scripting capabilities. Dymola can be coupled with the DSPack solver as shown in (Wolf et al., 2008). FMI2.0 specifies a functionality for evaluating partial derivatives of an FMU (*fmi2GetDirectionalDerivative*). Due to apparent implementation overhead this functionality is optional.

A common way to compute the required partial derivatives f_x and f_p is to employ classical AD techniques (Naumann, 2012) (cf. `www.autodiff.org`). The simpler way to accomplish that is via a computational approach for forward differentiation, cf. next Subsection. Alternatively, advanced heuristics based on FD schemes can be utilized. Instead of direct numerical integration of a system of size $n_x(n_p+1)$, couple of methods exist with improved performance efficiency (Maly and Petzold, 1996; Feehery et al., 1997). These methods attempt to exploit the linearized form of a sensitivity system and the structure of its extended Jacobian.

3.2 Computation of Partial Derivatives

Given the equation system f from (1), an AD algorithm in forward mode is capable of computing the s.c. tangent linear model y corresponding to directional derivatives of f w.r.t. a user-defined input. Formally, y is computed as:

$$y(u,s) = \frac{\partial f}{\partial u} \cdot s \tag{5}$$

where $u \in \{x,p\}$ and $s \in R^{|u|}$ is a seed vector. Several works related to evaluation of the Jacobian (i.e. $f_x(t) \in R^{n_x \times n_x}$) via AD in Modelica compilers have been reported (Olsson et al., 2005; Andersson et al., 2010; Braun et al., 2011). Given that $x = (x_1, x_2, \ldots, x_{n_x})^T$, the term f_x is accumulated by letting s ranges over the Cartesian basis of R^{n_x} as follows:

$$\begin{aligned} f_x = \Big[y(x,e_i) \Big]_{i=1,2,\ldots,n_x} &= \left[\frac{\partial f}{\partial x} \cdot e_i \right]_{i=1,2,\ldots,n_x} \\ &= \left[\frac{\partial f}{\partial x_i} \right]_{i=1,2,\ldots,n_x} \end{aligned} \tag{6}$$

where e_i is the i-th unit vector in R^n and each iteration i computes the i-th column in f_x. If f is expressed as a functional unit within a program P, AD produces another program P' that includes evaluation of f together with y. This implies that the program P' needs to get repeatedly evaluated for n_x (n_p) times in order to compute f_x (f_p) with potentially common intermediate computations being repeatedly performed.

A possibility to reduce such excessive evaluations is to exploit sparsity patterns of object-oriented Modelica models. In this way, many directional derivatives can be simultaneously computed by compressing many unit vectors into the seed vector s (Braun et al., 2012). However, even with this technique, it is not a wonder that straightforward symbolic differentiation outperforms AD w.r.t. runtime performance as reported in (Åkesson et al., 2012). In the case of $n_p \gg n_x$, excessive computational efforts can be avoided by following a backward AD approach as demonstrated in (Braun et al., 2017). (Hannemann-Tamas et al., 2012) shows another approach for computing first and second-order DPS using adjoint sensitivity analysis applied to a flattened Modelica model.

4 Equation-based AD for Modelica

So far demonstrated reliable approaches for computing DPS are limited to external solutions (FMI) or non-unified platform-dependent additional services. In contrary, a platform-independent approach to realize DPS is to exploit equation-based AD technique with which DPS is modeled using Modelica syntax. Given a Modelica model (or a library) DPS are modeled by systematically extending every base component by another component additionally including entities and equations for the sensitivity system. Then, a top-level model is slightly changed by specifying the set of active parameters.

4.1 Example: The ADGenKinetics Library

The ADGenKinetics library (Elsheikh, 2012) an algorithmically differentiated library capable of modeling the dynamics of biochemical reaction networks together with DPS, the major connection mechanism:

```
connector ChemicalPort
   "reaction connector"
  Units.Concentration c "concentration";
  flow Units.VolumetricReactionRate r
                     "reaction rate";
end ChemicalPort;
```

is extended within another separate package *Derivatives* as follows:

```
connector ChemicalPort
  extends Interfaces.ChemicalPort;
  outer parameter Integer NG
          "dimension of the gradients";
  Real g_c[NG] "gradients of c";
  flow Real g_r[NG] "gradients of r";
end ChemicalPort;
```

The parameter *NG* specifies the number of active parameters that is first to be specified at a top-level model. The array *g_c* (*g_r*) is a derivative object describing the derivatives of the concentration (the reaction rate) w.r.t. active parameters. Similarly a component providing basic interfaces for arbitrary chemical substances:

```
partial model BasicNode
  "Basic declarations of any Metabolite"
  extends
     Interfaces.dynamic.NodeConnections;
  parameter Units.Concentration c_0 = 0
         "initial concentration";
  Units.Concentration c(start = c_0)
         "substance concentration";
  Units.VolumetricReactionRate r_net
         "net reaction rate";
equation
  r_net = rc.r;
  rc.c = c;
  mc.c = c;
end BasicNode;
```

is extended as follows:

```
partial model BasicNode
  "Basic declarations of any Metabolite"
  extends Derivatives.Interfaces.
            dynamic.NodeConnections;
  extends NodeElements.dynamic.BasicNode;
  outer parameter Integer NG
                "# of gradients";
  parameter Real g_c_0[NG] = zeros(NG)
                "gradients of c_0";
  Real g_c[NG](start = g_c_0)
                "gradients of c";
  Real g_r_net[NG](start = zeros(NG))
                "gradients of r_net";
equation
  g_r_net[:] = rc.g_r[:];
  rc.g_c[:] = g_c[:];
  mc.g_c[:] = g_c[:];
end BasicNode;
```

The extended equations are obtained by differentiating the original equations w.r.t. arbitrary parameters. Note that the differentiated component is now extending the differentiated model of *NodeConnections*. To each variable and parameter, a corresponding derivative object is associated, e.g. *g_c_0*. In this way, derivatives w.r.t. start values can be also represented. Obtaining derivative formulas as well as the naming conventions of intermediate variables are comprehensively explained in (Elsheikh, 2015). The algorithm is also employed to manually derive partial derivatives of complex formulas.

4.2 A Top-level Model

A top-level model is slightly modified from:

```
model Spirallusdyn "An abstraction TCA
    cycle"
  import NodeElements.dynamic.*;
  import Reactions.convenience.dynamic.*;
  ..
  Node A;
  RevKinetic v1
    (NS = 1, NP = 1,
     Vfwdmax = 3.0, Vbwdmax = 1.0,
     ...);
  ...
equation
  ...
  connect(A.rc, v1.rc_S[1]);
  connect(v1.rc_P[1], B.rc);
  ...
end Spirallusdyn;
%
```

to:

```
model spirallusdynAll "Parameter
    sensitivities"
 import Derivatives.NodeElements.dynamic.*;
 import Derivatives.Reactions.
          convenience.dynamic.*;
 import Derivatives.Functions.*;
 // instead of
 // import NodeElements.dynamic.*;
 // ...

 // additional declaration
 inner parameter Integer NG = 24;
 ...
 // the exisiting components
 Node A;
 RevKinetic v1
    (NS = 1, NP = 1,
    Vfwdmax = 3.0,
        // declare the 4-th active
           parameter
        g_Vfwdmax = unitVector(4, NG),
    Vbwdmax = 1.0,
        // declare the 5-th active
           parameter
        g_Vbwdmax = unitVector(5, NG),
    ...);
  ...
equation
  // connection equations remain the same
  ...
  connect(A.rc, v1.rc_S[1]);
  connect(v1.rc_P[1], B.rc);
  ...
end  spirallusdynAll;
```

in order to additionally simulate DPS, cf. (Elsheikh, 2012) Section 5 for simulation results. Imported differentiated types are employed and input Jacobian is additionally declared. Methodological details on the equation-based AD method are demonstrated in (Elsheikh, 2014) illustrated on a simple part of the MSL, the ADMSL library. Regardless of being platform-independent uniform approach, the main advantage of this method is that DPS are already a part of the model, profitable to many applications highlighted in (Elsheikh and Kucherenko, 2019).

5 Whispering about *der(x,p)*

5.1 *der(x,p)* !?

The established methodology for equation-based AD of Modelica models / libraries inherits an implicit but intuitive proposal: *der(x,p)*. This allows the operator *der* to take a second argument as a parameter to represent DPS. The equation-based AD approach can be viewed as an equivalent prototype implementation of such an enhancement. More or less, *der(x,p)* may provide all equivalent activities a modeler currently needs to explicitly implement DPS at model level. For instance, a demonstration may look as follows:

```
model MyModel
  MyComp1  C1;
  MyComp2  C2;
  Real dxdp;
  Real dydp;
  Real dydq;
equation
  dxdp = der(C1.x,C2.p);
  dydp = der(C2.y,C2.p);
  dydq = der(C2.y,C1.q);
end MyModel;
```

The above model is equivalently realizable by equation-based AD at the model level.

5.2 What speaks for *der(x,p)?*

The demonstrated efforts for computing DPS of Modelica models reveal that they unintentionally converge to provide DPS-capabilities at the language level:

1. If a Modelica simulation environment will eventually support (or are already supporting) DPS for FMI, then it makes sense also to consider DPS at Modelica level due to potentially duplicated efforts
2. Why should a Modelica modeler need to externally employ FMI with additional overhead for reasons FMI have not been apparently established for?
3. Representing DPS at model level is intuitive and realizable for significant subset of Modelica models, cf. Section 6
4. Regardless of a significant set of applications of DPS where its advantageous to have DPS at the model level (Elsheikh and Kucherenko, 2019), DPS can be physically part of the model (Fell, 1992)

The last argument is referring to the s.c. control coefficients corresponding to scaled DPS allowing comparative studies among parameters generally applicable to arbitrary physical domains.

5.3 What speaks against *der(x,p)?*

While the demonstrated algorithmically differentiated libraries correspond to continuous-time physical models, probably most of Modelica applications cover discrete phenomena. It should be clear how to treat *der(x,p)* if x is not continuous. Formally speaking: assume that the given Modelica model corresponds to an explicit hybrid ODE:

$$\dot{x} = f(x,z,m,q,t) = 0 \quad , \quad x_0(p,t_0) = x_0(p) \qquad (7)$$

where x, p and t as given in Equation (1) and

- $m(q,t) \in R^{n_m}$ set of discrete variables with constant values between events
- $z(q,t) \in R^{n_z}$ set of event indicators where $z_i(t_j)$ changes sign implies that for a small $\varepsilon > 0$:

  $$m_k(t_j - \varepsilon) \neq m_k(t_j + \varepsilon),$$

  $$\lim_{t\to t_j^+} x_s(t_j) \neq \lim_{t\to t_j^-} x_s(t_j) \text{ or}$$

  $$\lim_{t\to t_j^+} \dot{x}_r(t_j) \neq \lim_{t\to t_j^-} \dot{x}_r(t_j)$$

  for some $k \in \{1,2,\ldots,n_m\}$, $s \in \{1,\ldots,n_x\}$ and $r \in \{1,\ldots,n_x\}$

Similarly, assume that p are the active parameters. Some justifiable questions, among others, on the validity of this study on hybrid systems would include:

1. When does a unique solution for the trajectories of sensitivities exist?
2. How to handle sensitivities when x jumps between events?
3. What if a parameter p_k is the time of event, i.e. $\partial p_k/\partial t_j = 1$ with some $z_i(t_j) = 0$
4. Generally, what if z is a function of parameters, i.e. $z = z(p,t)$?
5. How to handle the case where a parameter may even influence the presence of an event?
6. How to treat the influence of parameter values on event indicators z or even discrete variables m?

Definitely some of the above questions need to be adequately addressed. On the other hand, there are plenty of studies on SA of hybrid systems in literature e.g. (Galán et al., 1999; Barton and Lee, 2002; Saccon et al., 2014) some of which could be inspiring for addressing these questions.

6 Evaluation, Compilation and Simulation of Sensitivity Systems

Two AD approaches for generating a sensitivity system out of *der(x,p)* specification can be imagined. The classical one is to generate the required derivatives needed by specialized solvers for DPS, possibly using classical AD tools as demonstrated in Section 3.2. Another naive but attractive alternative is to self generate the sensitivity system using symbolic differentiation capabilities, if not equation-based AD. Afterwards the generated sensitivity system is subject to the standard optimization techniques of Modelica compilers (Murota, 1987; Cellier, 1991). In many cases, compilation complexity does not need to be overwhelmed due to the dimension of sensitivity system, as revealed in Section 6.2.

6.1 Equation-based AD

In the following, an advanced equation-based AD approach following a forward-differentiation scheme is demonstrated which

1. evaluates a sensitivity system in one single shot
2. overcomes the repetitive computations drawback
3. requires no additional memory for derivative objects of intermediate computations

This is the same approach that has been applied to compute partial derivatives of library components demonstrated in Section 4. Technically speaking w.r.t. the above mentioned first argument and partially the second argument, symbolic differentiation results in similar conclusions. The same applies to classical AD if the seed vector s from Eq. (5) becomes a seed matrix and assigned to the identity matrix I_{n_x} for computing the Jacobian. However both methods were not satisfiable for computing partial derivatives at Modelica language level (Elsheikh et al., 2008).

On the other hand, equation-based AD combines the advantages of both approaches by employing algorithmic capabilities borrowed from equation-based compilation techniques for simplifying common sub-expressions. In (Elsheikh, 2015) it is shown that equation-based AD does not associate derivative objects for intermediate computation resulting in efficient memory management of large equation systems of complex formulas. The proposed approach directly derives the sensitivity system by computing the rhs of Eq. (4) in one single shot as follows:

$$f_x\, S_1 + f_p\, S_2 \tag{8}$$

$$\text{where } S_1 = \frac{\partial x}{\partial v} \in R^{n_x \times q} \quad , \quad S_2 = \frac{\partial p}{\partial w} \in R^{n_p \times q}$$

S_1 & S_2 are seed matrices (vectors if $q = 1$) that are set using the auxiliary vectors $v, w \in R^q$ in the following way:

1. only f_x required: let $v = x \in R^{n_x}$, $w = \phi_{n_x} \in R^{n_x}$ where ϕ_{n_x} is a dummy vector with which $\partial \dot{x}/\partial \phi_{n_x} = 0 \in R^{n_x \times n_x}$ and $\partial p/\partial \phi_{n_x} = 0 \in R^{n_p \times n_x}$
2. only F_p required: let $w = p \in R^m$, $u = v = \phi_m \in R^m$
3. sensitivity subsystem required: set $v = w = p \in R^{n_p}$

Despite the large size of sensitivity systems, they can be compactly represented. Assuming that $x = (x_1, x_2, \ldots, x_{n_x})^T$, $p = (p_1, p_2, \ldots, p_{n_p})^T$ and $f = (f_1, f_2, .., f_{n_x})^T$ with

$$f_i : R^{n_x + n_p + 1} \to R$$

$$\text{s.t. } f_i \equiv f_i(x,p,t) = e_i^T \cdot f(x,p,t) \ , \ i \in \{1, 2, \ldots, n_x\}$$

corresponds to the i-th equation in f, the corresponding differentiated equation w.r.t. a parameter p_j is derived from Equation (8) as follows:

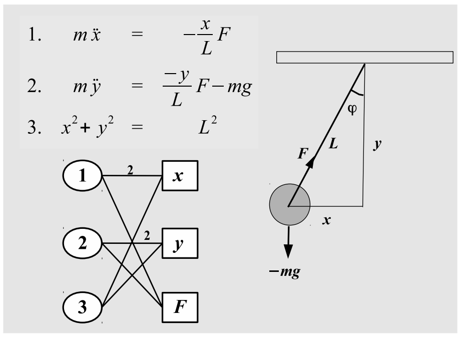
$$\frac{\partial \dot{x}_i}{\partial p_j}(x,p,t) = \sum_{k=1}^{n} \frac{\partial f_i}{\partial x_k} \frac{\partial x_k}{\partial p_j} + \sum_{k=1}^{m} \frac{\partial f_i}{\partial p_k} \frac{\partial p_k}{\partial p_j} \tag{9}$$

By assigning a gradient object (i.e. array) to each partial derivative, the sensitivity subsystem is compactly formulated with n_x equations as:

$$\frac{\partial \dot{x}_i}{\partial p}(x,p,t) = \sum_{k=1}^{n} \frac{\partial f_i}{\partial x_k} \frac{\partial x_k}{\partial p} + \frac{\partial f_i}{\partial p} \tag{10}$$

6.2 Compilation of Sensitivity Systems

When computing the sensitivity system of a Modelica model, an approach is to generate the sensitivity equation after optimizing the flat representation of the model. However the other way around might not be significantly less efficient. Naive generation of the sensitivity system of a flat equation system is also subject to optimization techniques of Modelica code. For instance, the differentiation of simple equations from connections leads to simple equations subject to removal at the optimized equation set. Another significant remark is demonstrated in Figure 3, the computational graph of an equation system describing the motion of a free pendulum in the Cartesian space (cf. (Fritzson, 2011)). The computational graph of the whole



1. $m\ddot{x} = -\frac{x}{L}F$
2. $m\ddot{y} = \frac{-y}{L}F - mg$
3. $x^2 + y^2 = L^2$
1
2
3
x
y
F
2
2
φ
F
L
y
x
−mg

Figure 3. Pendulum equation system

sensitivity system is shown in Figure 4.

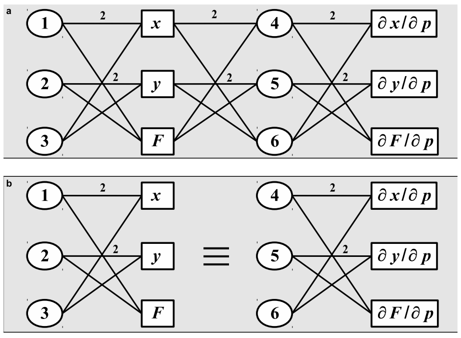


Figure 4. Computational graphs of pendulum equation sensitivity system

Obviously both computational graphs of the equation system and its sensitivity subsystems are isomorphic. This argument is generally valid as one of the mathematical foundlings proven in (Elsheikh and Wiechert, 2018). Consequently one can utilize already existing compiler capabilities (Mattsson, 1995; Maffezzoni et al., 1996), including index reduction (Pantelides, 1988; Mattsson and Söderlind, 1993), to transform the sensitivity system into a solvable optimized format. In many cases, there is no need to apply Modelica compiler techniques to the sensitivity subsystems. For example, the structural index (Leitold and Hangos, 2001) of both systems are identical. The equations within a sensitivity subsystem selected for differentiation towards index reductions are those corresponding to the equations within the original system selected for differentiation towards index reduction.

6.3 Simulation of Sensitivity Systems

Similarly, numerical integration of sensitivity equations with standard solvers can be slow for high-dimensional systems. By exploiting the structure of sensitivity systems, speed up can be achieved by exploiting the Modelica-friendly method described in (Dickinson and Gelinas, 1976). The original ODE / DAE system is numerically integrated together with a sensitivity subsystem corresponding to only one single active parameter. This computation is repeated for each active parameter. A generalization of this approach is to allow larger sensitivity subsystems corresponding to several active parameters. Hardware-oriented heuristics can be employed for selecting the optimal number of active parameters. Moreover, this approach is salable w.r.t. to parallelization (Elsheikh, 2015).

7 Summary and Outlook

This work summarizes some conducted efforts in the Modelica community to provide a functionality for computing DPS. In particular, equation-based AD allowing DPS to be present at the model level is highlighted. The study hints that these efforts may potentially converge to the integration of DPS facilities for Modelica. One possibility is to allow the operator *der* to have a second argument as a model parameter. From one side, this provides a uniform platform-independent solution for representing DPS. From the other side, many applications based on DPS are realizable. In order to promote the presence of DPS at Modelica level, a new Modelica library called PSTools is currently under development from which the example in Figure 1 is taken.

Representation of DPS at model level has been demonstrated in few application domains that are rather describing systems governed by continuous-time equation systems. While excessive triggering of events are not that common in the field of Systems Biology in comparison for instance with the field of Power Electronics, an absolute justification of *der(x,p)* demands a proper treatment of DPS in the presence of discontinuities and events. The mathematical foundation for DPS of hybrid systems have been extensively studied in literature which could be mapped to a proper Modelica-based treatment.

Acknowledgement

I would like to thank and acknowledge

- Jan Peter Axelsson (Vascaia AB, Stockholm, Sweden) for the model on which Figures 1 and 2 are based
- Hans Olsson (Dessault Systems, Lund, Sweden) for valuable feedback regarding *der(x,p)*

References

Åkesson, J., Braun, W., Lindholm, P., and Bachmann, B. (2012). Generation of sparse Jacobians for the function mock-up interface 2.0. In *Modelica'2012*, Munich, Germany.

Andersson, J., Houska, B., and Diehl, M. (2010). Towards a computer algebra system with automatic differentiation for use with object-oriented modelling languages. In *EOOLT'2010*, Oslo, Norway.

Barton, P. I. and Lee, C. K. (2002). Modeling, simulation, sensitivity analysis, and optimization of hybrid systems. *ACM Transactions on Modling and Computer Simulation (TOMACS)*, 12(4):256–289.

Braun, W., Kulshreshtha, K., Franke, R., Walther, A., and Bachmann, B. (2017). Towards adjoint and directional derivatives in FMI utilizing ADOL-C within OpenModelica. In *Modelica'2017*, Prague, Czech Republic.

Braun, W., Ochel, L., and Bachmann, B. (2011). Symbolically derived Jacobians using automatic differentiation - enhancement of the OpenModelica compiler. In *Modelica'2011*, Dresden, Germany.

Braun, W., Yances, S. G., Link, K., and Bachmann, B. (2012). Fast simulation of fluid models with colored jacobians. In *Modelica'2012*, Munich, Germany.

Cellier, F. E. (1991). *Continuous System Modeling*. Springer Verlag.

Dickinson, R. and Gelinas, R. (1976). Sensitivity analysis of ordinary differential equation systems - a direct method. *Journal of Computational Physics*, 21:123–143.

Elsheikh, A. (2012). ADGenKinetics: An algorithmically differentiated library for biochemical networks modeling via simplified kinetics formats. In *Modelica'2012*, Munich, Germany.

Elsheikh, A. (2014). Modeling parameter sensitivities using equation-based algorithmic differentiation techniques: The ADMSL.Electrical.Analog library. In *Modelica'2014*, Lund, Sweden.

Elsheikh, A. (2015). An equation-based algorithmic differentiation technique for differential algebraic equations. *Journal of Computational and applied Mathematics*, 281:135 – 151.

Elsheikh, A. and Kucherenko, S. (2019). Dynamic parameter sensitivities: Summary of applications – version 1.0. Technical report. to appear online.

Elsheikh, A., Noack, S., and Wiechert, W. (2008). Sensitivity analysis of Modelica applications via automatic differentiation. In *Modelica'2008*, Bielefeld, Germany.

Elsheikh, A. and Wiechert, W. (2012). Accuracy of parameter sensitivities of DAE systems using finite difference methods. In *MATHMOD 2012*, Vienna, Austria.

Elsheikh, A. and Wiechert, W. (2018). The structural index of sensitivity equation systems. *Mathematical and Computer Modelling of Dynamical Systems*, 24(6):553–572.

Feehery, W. F., Tolsma, J. E., and Barton, P. I. (1997). Efficient sensitivity analysis of large-scale differential-algebraic systems. *Applied Numerical Mathematics*, 25(1):41–54.

Fell, D. (1992). Metabolic control analysis: a survey of its theoretical and experimental development. *Biochemocal Journal*.

Fritzson, P. (2011). *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Wiley-IEEE Press, 1 edition.

Galán, S., Feehery, W. F., and Barton, P. I. (1999). Parametric sensitivity functions for hybrid discrete/continuous systems. *Applied Numerical Mathematics*, 31(1):17 – 47.

Hannemann-Tamas, R., Tillack, J., Schmitz, M., Förster, M., Wyes, J., Nöh, K., on Lieres, E., Naumann, U., Wiechert, W., and Marquardt, W. (2012). First and second order parameter sensitivities of a metabolically and isotopically non-stationary biochemical network model. In *Modelica'2012*, Munich, Germany.

Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., and Woodward, C. S. (2005). Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363–396.

Kucherenko, S. and Iooss, B. (2016). *Derivative-Based Global Sensitivity Measures*, pages 1–24. Handbook of Uncertainty Quantification, Springer International Publishing, Cham.

Leitold, A. and Hangos, K. M. (2001). Structural solvability analysis of dynamic process models. *Computers & Chemical Engineering*, 25:1633–1646.

Maffezzoni, C., Girelli, R., and Lluka, P. (1996). Generating efficient computational procedures from declarative models. *Simulation Practice and Theory*.

Maly, T. and Petzold, L. R. (1996). Numerical methods and software for sensitivity analysis of differential-algebraic systems. *Applied Numerical Mathematics: Transactions of IMACS*, 20(1–2):57–79.

Mattsson, S. E. (1995). Simulation of object-oriented continuous time models. *Mathematics and Computers in Simulation*, 39(5 – 6):513 – 518.

Mattsson, S. E. and Söderlind, G. (1993). Index reduction in differential-algebraic equations using dummy derivatives. *SIAM Journal on Scientific Computing*, 14(3):677 – 692.

Murota, K. (1987). *Systems Analysis by Graphs and Matroids*. Springer, Berlin.

Naumann, U. (2012). *The Art of Differentiating Computer Programs, an Introduction to Algorithmic Differentiation*. SIAM.

Olsson, H., Tummescheit, H., and Elmqvist, H. (2005). Using automatic differentiation for partial derivatives of functions in Modelica. In *Modelica'2005*, Hamburg, Germany.

Pantelides, C. C. (1988). The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231.

Petzold, L., Li, S. T., Cao, Y., and Serban, R. (2006). Sensitivity Analysis of Differential-Algebraic Equations and Partial Differential Equations. *Computers & Chemical Engineering*, 30:1553–1559.

Saccon, A., van de Wouw, N., and Nijmeijer, H. (2014). Sensitivity analysis of hybrid systems with state jumps with application to trajectory tracking. In *53rd IEEE Conference on Decision and Control*, pages 3065–3070.

Saltelli, A., Tarantola, S., Campolongo, F., and Ratto, M. (2004). *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. Halsted Press, New York, NY, USA.

Wiechert, W., Noack, S., and Elsheikh, A. (2010). Modeling languages for biochemical network simulation: Reaction vs equation based approaches. *Advances in Biochemical Engineering / Biotechnology*, 121:109 – 138.

Wolf, S., Haase, J., Clauß, C., Jöckel, M., and Lösch, J. (2008). Methods of sensitivity calculation applied to a multi-axial test rig for elastomer bushings. In *Modelica'2008*, Bielefeld, Germany.

Frequency Response Estimation Method for Modelica Model and Frequency Estimation Toolbox Implementation

Bingrui Bao[1] Junfeng Guo[1] Baokun Zhang[1] Fanli Zhou[1]

[1]Suzhou Tongyuan Software&Control Tech. Co., Ltd. China, {baobr,guojf,zhangbk,zhoufl}@tongyuan.cc

Abstract

The frequency-response method is widely used because of its convenience and applicability in the control system analysis and design, and the premise of the design based on this method is to obtain the frequency response of the system. Aiming at the problem that the strong nonlinearity is difficult to be linearized in practical engineering systems, this paper presents a method for estimating the frequency response based on the time domain simulation data of Modelica models. The spectrum of the appropriate IO data at the steady-state operating point of the system is analyzed by Fourier transform, and then converted into the frequency response of the system. The proposed method is applicable to the multi-domain Modelica model, FMU, and the black box model. A frequency response estimation toolbox is implemented and integrated into the MWorks platform based on this method, which provides important support for the control design for Modelica model. Simulation examples illustrated the validity of the proposed method and the toolbox.

Keywords: Strong nonlinear system, Frequency response estimation, Toolbox

1 Introduction

Compared with time domain analysis, the frequency domain analysis can provide more intrinsic properties of the system. The design of feedback control systems in industry is probably accomplished using frequency-response methods more often than any other (Dorf *et al*, 2016; Ogata, 2010). The frequency-response design is popular primarily because it provides good design ability in the case of uncertainty in the plant which is difficult to describe with accurate mathematical model. On the other hand, it is the easiest method to use for designing compensation (Franklin *et al*, 2009). The premise of feedback control design based on frequency response method is to obtain the frequency response of the system.

In Brief, the theoretical analysis and experimental estimation are two fundamentally methods for obtaining system frequency-response. As regards the theoretical analysis method, which performs Laplace transform on the mathematical model of the system to obtain its transfer function, and then replaces the Laplace operator $\boldsymbol{s}$ with $\boldsymbol{j\omega}$ to obtain the frequency response model. Speaking of the experimental method, it could be introduced as estimating system frequency-response by means of deeming system to be black box and then processing time-domain IO signal which is obtained from experiments.

The plant of industry systems are generally characterized by multi-domain, high stiffness and strong nonlinearity, thus Modelica has its natural advantages in describing systems with these characteristics (Fritzson, 2010). Meanwhile, plenty of remarkable studies on method of frequency-domain analysis of Modelica model have been conducted all this time. Martin Otter introduced the LinearSystems library in 2006, the library provides a definition of linear systems and also can be used to analyze it, including analysis of frequency response for linear systems (Otter, 2006). Andreas Abel et al. proposed a frequency domain analysis method for Modelica model based on the periodic steady-state simulation by using SimulationX, the transfer function and the frequency domain related properties of the system can be obtained by the models linearized in an operating point (Abel, 2008). Loig Allain et al. performed the linear analysis methods by the existing facilities in LMS Imagine.Lab, which is essentially a linearization method based on small perturbation theory (Allain *et al*, 2009). The LinearSystems2 library is an updated version of the LinearSystems library was completed in 2009 by Marcus Baur, Martin Otter, etc., which enhances and extends the functionality of the original library (Baur *et al*, 2009), for example, a linearize function for linearization of physical models is added. Tilman Bünte proposed a frequency response estimation method based on time-domain data (Bünte, 2011). However, for strongly nonlinear systems, time-domain data acquired from Chirp signals have poor performance in estimation. Based on the linearization method provided by Dymola, Garron Fish et al. analyzed the natural frequencies of powertrain systems, the linearization function of Dymola is also based on the small perturbation theory (Fish *et al*, 2012).

The research above of frequency-domain analysis for Modelica model can be divided into two categories: frequency-analysis based on linearization and frequency-response estimation based on IO data from model. The shortcomings of the current researches can be summarized as follows.

- For one, Linearization is generally based on the small perturbation theory, which have to meet two requirements. Firstly, there should be at least one steady-state operating point in system. Secondly, all-order derivatives of equations describing the system at the operating point require to be existed. It cannot be stressed enough that the second requirement could be hardly met.
- For another, neither user-friendly process of existing data-based frequency response estimation has been standardized nor has method of obtaining time-domain data with acceptable estimation results been put forward.
- In addition, user-friendly time-domain data based frequency response estimation toolbox has not been formed yet.

Given the difficulties of frequency analysis on Modelica model, the topics of study on obtaining the appropriate time-domain data for estimating system frequency response of a class of SIMO(single input multiple output) Modelica model are as follows.

- A general process and key algorithm of data based frequency response estimation for Modelica model.
- A time-domain data obtainment method suitable for estimating the frequency response of strongly nonlinear system.
- Frequency response estimation toolbox which is developed on the basis of above method.

2 Frequency Response Estimation Method for Modelica Model

Linear system response to sinusoidal inputs is named as the system frequency response. Frequency of the input signal is varied in a certain range for studying the resulting response in the method of frequency response estimation. The following state space models are used to represent the general continuous-time nonlinear Modelica model for the convenience of explaining the problem.

$$\begin{aligned}\dot{\boldsymbol{x}}(t) &= f[\boldsymbol{x}(t), \boldsymbol{u}(t), t]\\ \boldsymbol{y}(t) &= g[\boldsymbol{x}(t), \boldsymbol{u}(t), t]\end{aligned} \tag{1}$$

In models above, t is the time and $\boldsymbol{x}(t)$ is the state variables of the system. Besides, $\boldsymbol{u}(t)$ is the input of the system and $\boldsymbol{y}(t)$ is the output. The frequency response estimation for the system described in equation (1) is performed near a certain steady-state operating point of the system. It is assumed that the system has a steady-state operating point $\boldsymbol{x}(t_{op})$ under the input $\boldsymbol{u}(t)$. As a result, the steady-state output of the system at the operating point can be expressed as $\boldsymbol{y}(t_{op}) = g[\boldsymbol{x}(t_{op}), \boldsymbol{u}(t_{op}), t_{op}]$.

2.1 Principle of Frequency Response Estimation

The designed small disturbance input signal $\Delta(t)$ is added to the existing input signal $\boldsymbol{u}(t)$ at the linearization input point in the method of frequency response estimation. A new input signal excitation system model for obtaining the output at the linearization output point is described as follows. $\boldsymbol{u}_{test}(t) = \boldsymbol{u}(t) + \Delta(t)$

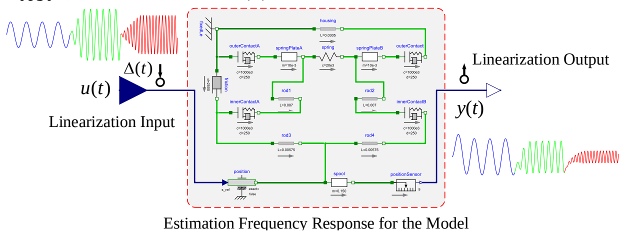


Figure 1. Acquisition of time domain data

There is a transient portion and steady state portion at each frequency of the simulated output $\boldsymbol{y}(t)$. The steady state portion is selected for sampling and filtering, etc. FFT transform is applied to the processed output signal to estimate the spectrum. Frequency response near the operating point can be easily calculated based on the spectrum of output signal.

2.2 Frequency Response Estimation Algorithm

- **Model encapsulation**

The model to be analyzed is encapsulated as a separate model. As is shown in Figure 1, linearized input point and linearized output point is defined. The following test signal will be added into the linearized input point. In consideration of the convenience of adding the test signal into input point, the type of the input interface is set as RealInput.

- **Create Estimation Input Signals**

Chirp signal and Sinestream signal are commonly used for frequency response estimation as disturbance signals.

$$\begin{aligned}&chirp(\omega, t) = A \cdot \sin(\omega t)\\ &t_s \le t \le t_e, \omega_s \le \omega \le \omega_e\end{aligned} \tag{2}$$

$$sinestream(t) = \begin{cases} A_0 \cdot \sin(\omega_0 t), t_s \le t < t_0\\ A_1 \cdot \sin(\omega_1 t), t_0 \le t < t_1\\ \vdots \\ A_n \cdot \sin(\omega_n t), t_{n-1} \le t < t_n\end{cases} \tag{3}$$

As is shown in equations (2) and (3), The Chirp signal and Sinestream signal are respectively expressed in equations (2) and (3), in which it can be seen that Chirp signal is a continuous signal while Sinestream signal is a segmentation signal. Chirp signal is suitable for system which is nearly linear in simulation range. However, if there existed strong nonlinearity in system and high accuracy of frequency response was required, Sinestream signal would be an excellent choice as disturbance signal.

Sinestream signal chosen as excitation signal is applied for time-domain data acquisition of system. To begin with, the range of frequency estimation along with number of the estimated points are defined appropriately. Secondly, steady-state operating point of the system needs to be set properly as well. Thirdly, amplitude, period and number of sampling points should be customized as attributes of each corresponding sinusoidal signal with specific frequency. The definitions of signal attributes are shown in Figure 2.

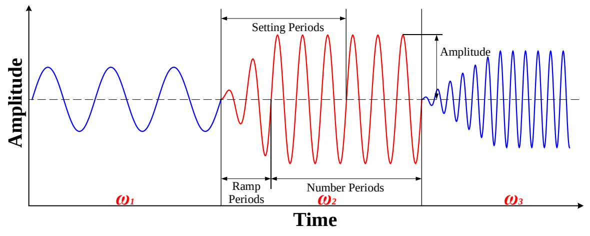


Figure 2. Definition of sine-stream signal attribute

According to the definition of the Sinestream signal above, model based on Modelica language can be easily implemented. Code part is omitted for lack of space.

- **Time Domain Simulation**

A Sinestream signal consists several adjacent sine waves with varying frequencies. Each frequency excites the system for a suitable time to ensure that the system could enter steady state. The output at the linearized output point is set as the original signal for the following estimation.

- **Signal Processing**

In order to ensure the accuracy of the estimation results, the output signal is divided into n segments according to the duration of different frequency components. n represents the number of different frequency points contained in the set frequency range. Figure 3 shows a schematic diagram of the system input signal and output signal at a specific frequency.

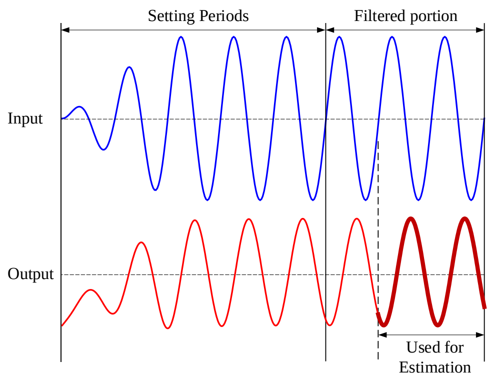


Figure 3. Signal selection

The simulated output at each frequency has a transient portion and steady state portion. Settling Periods corresponds to the transient components of the output and input signals. The periods following Settling Periods are considered to be at steady state. Discards the Settling Periods portion of the output (and the corresponding input) at each frequency. Sampling the signal used for estimation, assuming that F_s is the sampling frequency and N is the number of sampling points.Two points need to be noted are as follows.

i. According to the sampling theorem, it is necessary to ensure that F_s is two times larger than the signal frequency.
ii. In order to ensure the calculation speed of FFT transform, the number of sampling points N is usually an integer power of two.

- Frequency Response Estimation

The discrete Fourier transform (DFT) is performed on the processed output data. $\boldsymbol{x}[n]$ is assumed to be the processed output at a certain frequency of which length is n. The DFT is performed as follows.

$$\boldsymbol{Y}(k)=\sum_{j=1}^{n}\boldsymbol{x}(j)W_n^{(j-1)(k-1)}, W_n=e^{(-2\pi i)/n} \qquad (4)$$

$\boldsymbol{Y}(k)$ is the spectrum obtained by Fourier transform of the signal. It is said that the specific DFT algorithm described in detail in many literatures (Oppenhiem *et al*,1997). However, it is omitted in this paper. The amplitude and phase of each frequency could be calculated from the corresponding signal spectrum. Finally, the frequency response of the system could be estimated.

3 Implementation of Frequency Response Estimation Toolbox and Application Examples

3.1 Introduction for the toolbox

Currently the proposed-method based frequency response estimation toolbox has been developed and integrated into MWorks.Sysplorer platform.

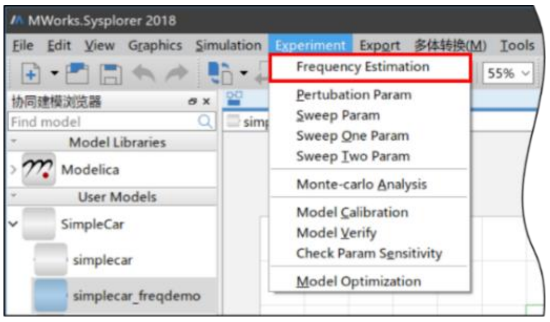
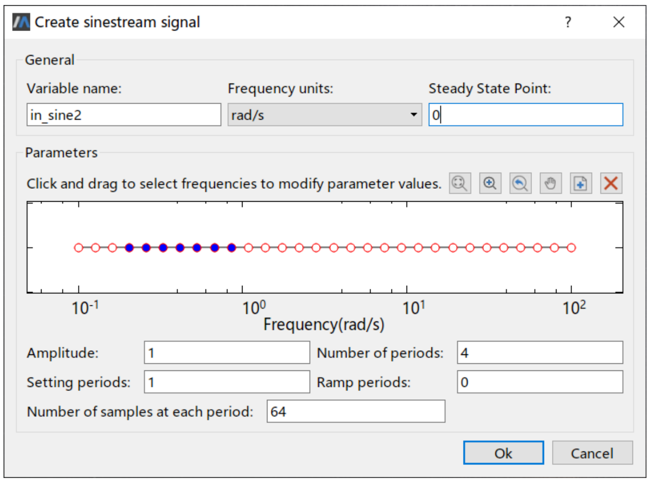


Figure 4 . Toolbox launch position

Main functions of the frequency response estimation toolbox are made up of creation of test signals, setting of linear input and output points along with graphical representation of the results of frequency response estimation.

The creation of test signals is a highlight of the toolbox that could create Sinestream signal flexibly via UI interface.



Create sinestream signal

General

Variable name: in_sine2

Frequency units: rad/s

Steady State Point: 0

Parameters

Click and drag to select frequencies to modify parameter values.

Amplitude: 1

Number of periods: 4

Setting periods: 1

Ramp periods: 0

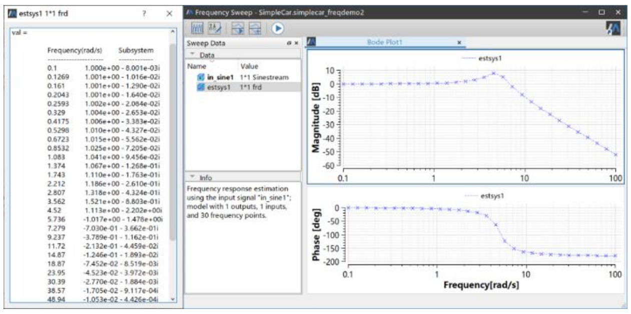
Number of samples at each period: 64

Figure 5. The create sine-stream input dialog box

As can be seen in Figure 5, Sinestream signal could be created visually by customizing system steady-state point, range of frequencies along with point numbers for estimation during it(logarithmically/linearly). In addition, customized parameters of signals for each frequency are as follows.

i. Amplitude;
ii. Number of periods;
iii. Settling periods;
iv. Ramp periods;
v. Number of samples at each period.

The estimation results of the system model can be presented in the toolbox in spectral data or Bode diagram.



estsys1 1*1 frd

val =

Frequency(rad/s)	Subsystem
0.1	1.000e+00 - 8.001e-03i
0.1269	1.001e+00 - 1.016e-02i
0.161	1.001e+00 - 1.290e-02i
0.2043	1.001e+00 - 1.640e-02i
0.2593	1.002e+00 - 2.084e-02i
0.329	1.004e+00 - 2.653e-02i
0.4175	1.006e+00 - 3.383e-02i
0.5298	1.010e+00 - 4.327e-02i
0.6723	1.015e+00 - 5.562e-02i
0.8532	1.025e+00 - 7.205e-02i
1.083	1.041e+00 - 9.456e-02i
1.374	1.067e+00 - 1.268e-01i
1.743	1.110e+00 - 1.763e-01i
2.212	1.186e+00 - 2.610e-01i
2.807	1.318e+00 - 4.324e-01i
3.562	1.521e+00 - 8.803e-01i
4.52	1.113e+00 - 2.202e+00i
5.736	-1.017e+00 - 1.478e+00i
7.279	-7.030e-01 - 3.662e-01i
9.237	-3.789e-01 - 1.162e-01i
11.72	-2.132e-01 - 4.459e-02i
14.87	-1.246e-01 - 1.893e-02i
18.87	-7.452e-02 - 8.519e-03i
23.95	-4.523e-02 - 3.972e-03i
30.39	-2.770e-02 - 1.884e-03i
38.57	-1.705e-02 - 9.117e-04i
48.94	-1.053e-02 - 4.426e-04i

Frequency Sweep - SimpleCar.simplecar_freqdemo2

Sweep Data

Name	Value
in_sine1	1*1 Sinestream
estsys1	1*1 frd

Info

Frequency response estimation using the input signal "in_sine1"; model with 1 outputs, 1 inputs, and 30 frequency points.

Figure 6. The results of estimation

Meanwhile, toolbox supports that obtaining the characteristics such as bandwidth of the system, resonance peak, amplitude margin and phase margin from estimation results.

3.2 Application of A Simple Example

Building model of a typical spring-damping-mass system of which parameters is shown as below in MWorks.Sysplorer. The model can be seen in Figure 7.

$M_1 = 6000kg$
$M_2 = 160kg$
$k_1 = 5000N/m$
$k_2 = 10000N/m$
$D = 1000N/(m/s)$

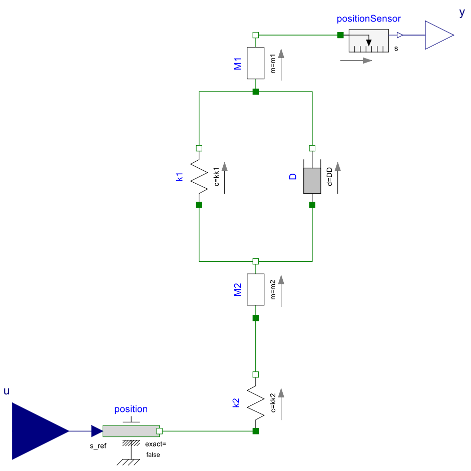
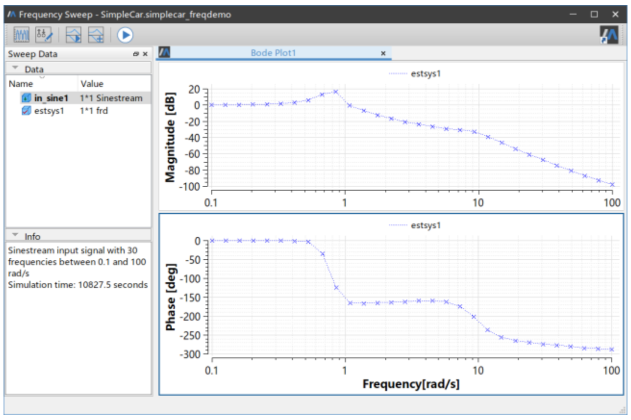


Figure 7. Spring-Damping-Mass system model

Estimating the frequency response of the system in the range of [0.1, 100] rad/s with frequency response estimation toolbox.



Frequency Sweep - SimpleCar.simplecar_freqdemo

Sweep Data

Name	Value
in_sine1	1*1 Sinestream
estsys1	1*1 frd

Info

Sinestream input signal with 30 frequencies between 0.1 and 100 rad/s
Simulation time: 10827.5 seconds

Figure 8. Estimation results of frequency response

The system of this example is actually a linear system. Therefore, the transfer function can be derived and then calculate the theoretical frequency response of the system to verify the accuracy of the above estimation results.

The transfer function of the system can be derived as follows:

$$G(s) = \frac{10s + 50}{0.96s^4 + 6.16s^3 + 90.8s^2 + 10s + 50} \tag{5}$$

The theoretical frequency response of the system is calculated by MATLAB and compared with the results of estimation.

As shown in Figure 9, the result estimated by the proposed method is highly matched with the theoretical analytical result, which verifies the effectiveness of the proposed method and toolbox.

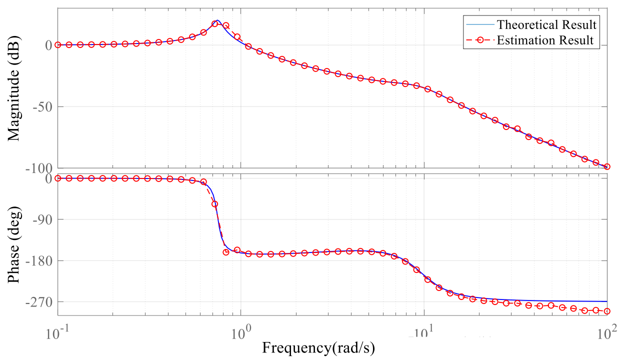


Figure 9. Verification of frequency response estimation

3.3 Frequency Response Analysis of Electro-Hydraulic Actuator

Taking the electro-hydraulic actuator of a certain type of aircraft flight control system as the research object, the actuator in the aircraft is usually called Power Control Unit (PCU). The design requirements of PCU includes the frequency response index. According to the principle of this type of PCU, its Modelica model is established as the following figure:

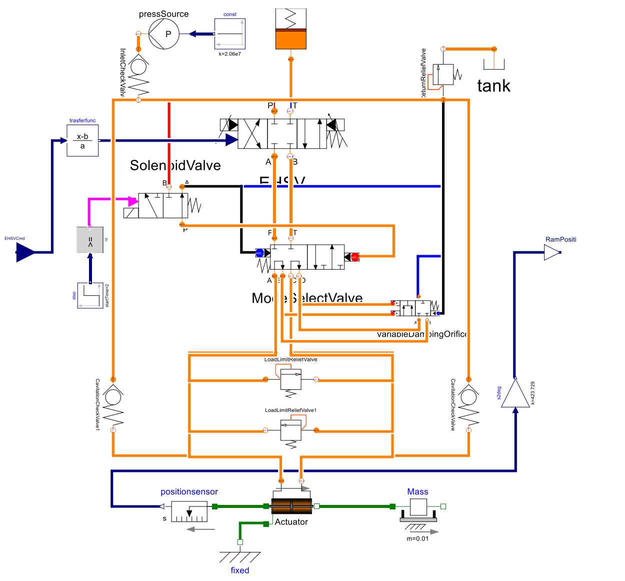
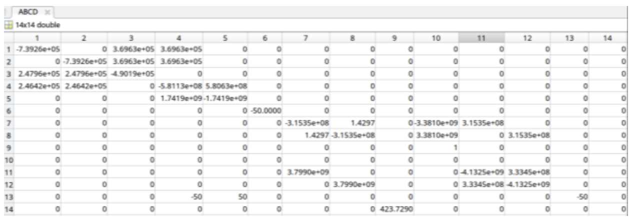


Figure 10. PCU model

Firstly, linearized function referred as `linearizeModel` on Dymola is applied to the PCU model which provide linearization process. The linearization result is shown as follows.



ABCD
14x14 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	-7.3926e+05	0	3.6963e+05	3.6963e+05	0	0	0	0	0	0	0	0	0	0
2	0	-7.3926e+05	3.6963e+05	3.6963e+05	0	0	0	0	0	0	0	0	0	0
3	2.4796e+05	2.4796e+05	-4.9019e+05	0	0	0	0	0	0	0	0	0	0	0
4	2.4642e+05	2.4642e+05	0	-5.8113e+08	5.8063e+08	0	0	0	0	0	0	0	0	0
5	0	0	0	1.7419e+09	-1.7419e+09	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	-50.0000	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	-3.1535e+08	1.4297	0	-3.3810e+09	3.1535e+08	0	0	0
8	0	0	0	0	0	0	1.4297	-3.1535e+08	0	3.3810e+09	0	3.1535e+08	0	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	3.7990e+09	0	0	0	-4.1325e+09	3.3345e+08	0	0
12	0	0	0	0	0	0	0	3.7990e+09	0	0	3.3345e+08	-4.1325e+09	0	0
13	0	0	0	-50	50	0	0	0	0	0	0	0	-50	0
14	0	0	0	0	0	0	0	0	423.7290	0	0	0	0	0

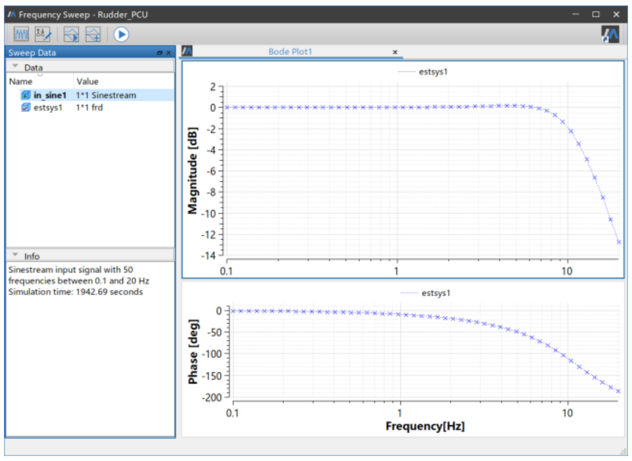
Figure 11. Linearization results of PCU model

As is shown in Figure 11, the thirteenth-order linear state-space model of the system could be obtained after linearization. Matrix ***B*** and ***D*** are as follows.

$$\boldsymbol{B} = \left[\overbrace{0 \quad 0 \quad \cdots \quad 0}^{13}\right]^T, \quad \boldsymbol{D} = 0 \tag{6}$$

It can be concluded that input of the system has no influence on state variables and system output. Therefore, it is incorrect to linearize the PCU model as many hydraulic components are contained of which strong nonlinearity could not meet the linearization conditions.

The frequency response is estimated in the frequency range of [0.1,20] Hz given the same conditions as those of frequency identification test of this type of PCU. The estimation result can be seen in the following figure.



Data

Name	Value
in_sine1	1*1 Sinestream
estsys1	1*1 frd

Info

Sinestream input signal with 50 frequencies between 0.1 and 20 Hz
Simulation time: 1942.69 seconds

Figure 12. Estimation Results Of The PCU Model

Comparison between result of the frequency response estimation and result of the actual system frequency identification test is shown in Figure 13.

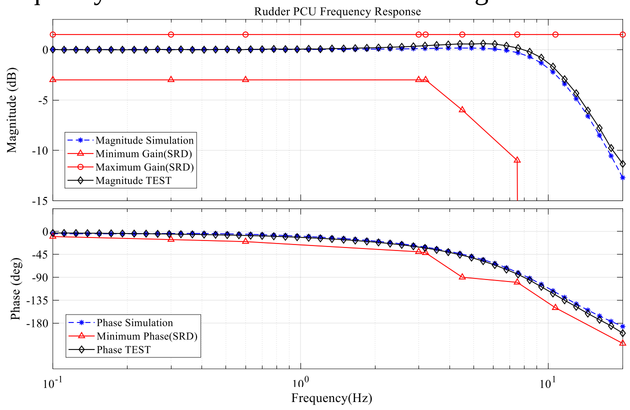


Figure 14. Verification of estimation results of PCU

As can be seen in Figure 14, red curve is represented as the frequency response range of the PCU defined in the Systems Requirements Document (SRD) while black curve is represented as the results of the frequency identification test. As for blue curve, it is defined as the results of estimation based on frequency response estimation toolbox. In summary, results of estimation confirms to those of identification test, thereby the validity and accuracy of the proposed method and corresponding toolbox.

4 Conclusion

Considering that the system has the characteristic of strong nonlinearity in the actual engineering, a method for estimating the frequency response of multi-domain Modelica model is proposed to copy with difficulty. Generating appropriate signal to excite system model and then estimating the system frequency response counting on output data from time-domain simulation to avoid stringent requirements for applying linearization method on system model. It is worth mentioning that frequency response estimation toolbox for Modelica model has been integrated into MWorks.Sysplorer platform to improve usability along with flexibility of the method. Last but not least, the viability of proposed method and toolbox could be verified by way of taking simulation above for example.

Currently the frequency response estimation introduced is applicable to single input multiple output (SIMO) Modelica model. There is high potential that this method could be widely used for multiple input multiple output (MIMO) system and finalizing promotion is established as target of next work. Besides, analysis on the method is mainly introduced, which is foundation for control system design subsequently. It will be important further work that guaranteeing availability of complete process of designing control system for Modelica model.

References

Andreas Abel, Tobias Nähring, Frequency-Domain Analysis Methods for Modelica Models, *Proceedings of the 6th International Modelica Conference*, 2008.

Alan V.Oppenhiem, Alan S. Willsky, With S. Hamid Nawab, Signals and Systems, Second Edition, Pearson, 1997.

Fritzson. Peter. Principles of object-oriented modeling and simulation with Modelica 2.1. John Wiley & Sons, 2010.

Gene F. Franklin, J. David Powell, Abbas Emami-Naeini, Feedback Control of Dynamic Systems, Fifth Edition. Pearson Education US. 2009.

Garron Fish, Mike Dempsey, Juan Gabriel Delgado, Neil Roberts, Natural frequency analysis of Modelica powertrain models, *Proceedings of the 9th International Modelica Conference*, 2012.

http://www.fftw.org/

Katsuhiko Ogata. Modern Control Engineering, Fifth Edition. Pearson. 2010.

Loig Allain, Stéphane Neyrat, Antoine Viel, Linear Analysis Approach for Modelica Models, *Proceedings of the 7th International Modelica Conference*, 2009.

Marcus Baur, Martin Otter, Bernhard Thiele, Modelica Libraries for Linear Control Systems, *Proceedings of the 7th International Modelica Conference*, 2009.

Martin Otter, The LinearSystems library for continuous and discrete control systems, *Proceedings of the 5th International Modelica Conference*, 2006.

Richard C.Dorf, Robert H.Bishop. Modern Control Systems, 12th Edition. Pearson. 2016.

Tilman Bünte, Recording of Model Frequency Responses and Describing Functions in Modelica, *Proceedings of the 8th International Modelica Conference*, 2011.

The MathWorks Inc. https://ww2.mathworks.cn/help/slcontrol/ug/frest.chirp.html

Modelica Models for the Control Evaluations of Chilled Water System with Waterside Economizer

Yangyang Fu[1] Xing Lu[1] Wangda Zuo[1]

[1]Department of Civil, Architectural and Environmental Engineering, University of Colorado at Boulder, USA,
{yangyang.fu,xing.lu-1,wangda.zuo}@colorado.edu

Abstract

Chilled water system with waterside economizer is a common cooling system used for large commercial buildings and data centers. To evaluate the design and control of the cooling system, modeling and simulation techniques are essential. This paper presents an equation-based modeling package for chilled water cooling system and a library of system- and equipment-level control. Then a case study is conducted to evaluate performance of the system-level control under different climate zones. Simulation results show that both temperature and humidity of the climate zone have influences on the economizing hours of the system, which thus influences the energy consumption.

Keywords: Chilled Water System, Waterside Economizer, Control Evaluation

1 Introduction

Commercial buildings have large cooling loads that are removed by Heating Ventilation and Air Conditioning system. American Society of Heating Refrigeration, and Air-Conditioning Engineers (ASHRAE) standards 90.1 states the requirements of the cooling system with waterside economizer for different climate zones.

Modeling and simulation is a cost-effective way to evaluate of the design and operation of the cooling systems. Modeling refers to the process that the real physical system is represented as mathematical models. Different physical systems (thermal, electrical, and electromagnetic, etc.) with different time-scaled dynamics are involved. This usually leads to high-indexed differential algebraic equations. Simulation is then conducted to numerically solve the mathematical equations in order to calculate the unknowns, which involves computer representation of models, different numerical solvers, solution procedures etc.

Many tools have been developed in academia and industry to perform computer modeling and simulation of the cooling systems in buildings. For example, eQuest (Lee and Chen, 2013), EnergyPlus (Pan et al., 2008) (Ham and Jeong, 2016), TRNSYS (Agrawal et al., 2016), and some customized simulation tools such as Energy Modeling Protocol (Shehabi et al., 2008) have been used to study the cooling systems with waterside economizers (WSEs) and airside economizers (ASEs) in data centers. Most of these traditional tools are based on imperative programming languages such as FORTRAN, C/C++ etc.

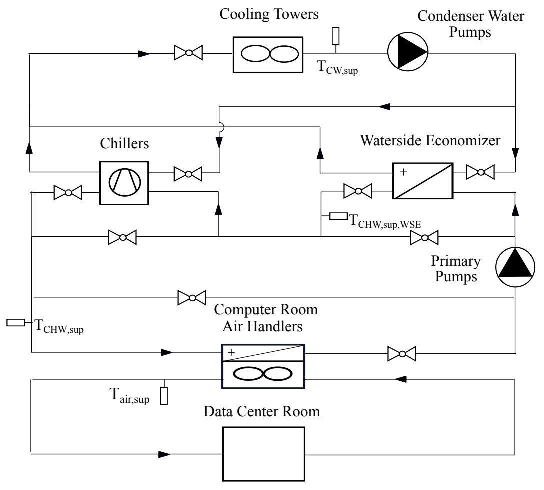
These imperative programming language based tools have exposed several challenges in the context of modeling, simulation and optimization. They have limited capacity when it comes to control designs and evaluations. For instance, EnergyPlus adopts idealized controls to reduce computation time(Fu et al., 2018). Although TRNSYS has dynamic control models, its constant time step poses numerical challenges(Kim et al., 2013). Further, conventional tools often intertwine model equations and numerical solvers in their source codes; this makes it difficult to extend these programs to support control-oriented cases.

Equation-based language such as Modelica (Elmqvist et al., 1998) can provide solutions to the above-mentioned issues. Modelica separates physical equations and numerical solvers wherever possible. The separation can mitigate the risks of intertwinement, and can fully take advantages of different expertise from different domains(Wetter et al., 2016; Fu et al.). For example, model developers can concentrate on how to develop efficient high-fidelity physical models, while computer engineers can focus on the development of robust numerical solvers. Also, the State Graph package in Modelica can be used to perform discrete control which contains dead band or delay time. The rich library of numerical solvers in Modelica can be chosen for different systems and different use cases.

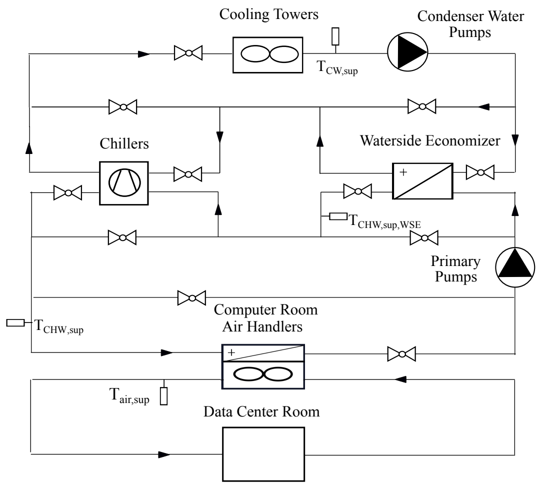
In this paper, we present a Modelica-based package for the chilled water system with waterside economizers. The cooling component models are built on Modelica Buildings library(Wetter et al., 2014), and the control logic of the cooling system are adopted from engineers' experience. We first introduce the chilled water system with waterside economizers, and then discuss the Modelica models for the above-mentioned cooling and control system. In Section 4, we perform an comprehensive evaluation of the mentioned cooling mode control for different climate zones. The conclusions are discussed in Section 5.

2 Chilled Water System with Waterside Economizer

Chilled water system is usually used for commercial buildings. A typical chilled water system includes chillers,



(a) Parallel



(b) Series

Figure 1. Primary-only chilled water system with integrated WSE

air handling units (AHUs), pumps, and cooling towers (Huang et al., 2016). The heat generated in the building is first transferred to the chilled water through AHUs, and the chillers then transfer the heat in the chilled water loop into the condenser water loop through a refrigeration system. The cooling towers at last reject the the heat in the condenser water loop to the outdoor environment.

Waterside economizers (WSE) can also be installed to provide auxiliary cooling when outdoor condition allows. It can be configured with chillers in different ways. For example, the WSE can be integrated with chillers, meaning that the economizer can meet all or some of the load while the chiller meets the rest of the load, or non-integrated, meaning the economizer can only operate when it can meet the entire load.

Two common configurations of the chiller plant with integrated WSEs are shown in Figure 1. The WSE is located on the load side of the common leg rather than plant side in the chiled water loop, which can guarantee that the WSE can meet the warmest return chilled water and maximize the economizing hours. On the condenser water side, the WSE can be installed in parallel (Figure 1(a)) or in series with chillers (Figure 1(b)). The chiller plant with integrated WSEs can operate in three modes: Free Cooling (FC) mode when only WSEs are enabled for cooling, Partial Mechanical Cooling (PMC) mode when the chillers and WSEs are both triggered, and Fully Mechanical Cooling (FMC) mode when only the chillers are activated. The cooling mode of the cooling system is determined by a cooling mode controller, and achieved by manipulating the associated isolation valves and the bypass valves of the chillers and the WSE.

3 Modelica Models

In the following section, Modelica models for the cooling system and the controls are elaborated.

3.1 Cooling System

In this section, the implementation of equipment models such as the chiller, WSE, and the cooling tower is first presented. Then, the subsystem models for two system configurations are illustrated.

3.1.1 Chiller

We developed a chiller model that supported the head pressure control and could model numbers of identical electric chillers. A group of identical chillers are modeled by vectorising existing chiller models in the Moelica Buildings library. The vectorized equipment model is assigned with the same design parameters but different performance curves if needed. A partial class of the chiller model is instantiated through vectorization with a number *num* by specifying the length of the array chiller, which can be redeclared with a detailed chiller model later. Medium used in the chillers, design parameters such as the design capacity, and performance curves of each chiller are specified. Note that although the chillers have identical nominal conditions, they can have different performance curves specified in performance data *per*. In addition, we add isolation valves in the vectorized models to avoid circulating flow among components.
The head pressure control in this model is achieved by modulating the isolation valve on the condenser water side to maintain a minimum temperature difference between evaporator and condenser, such as 11°C.

3.1.2 WSE

The waterside economizer model consists of a heat exchanger with outlet temperature control and two isolation valves on both medium sides. This waterside economizer model can be used in two different control scenarios: (1) the outlet temperature at chilled water side is controlled by a built-in PID controller and a three-way valve by setting the parameter *use_controller* as true. (2) the outlet temperature at chilled water side is NOT controlled by a built-in controller by setting the parameter *use_controller* as false.

Hence, an outside controller can be used to control the temperature. For example, in the free-cooling mode, the speed of variable-speed cooling tower fans can be adjusted to maintain the supply chilled water temperature around the set point.
The user can select different heat exchanger models according to the data availability and modeling requirement. In this model, the heat exchanger model uses the nominal approach temperature and heat capacity as its critical design parameters. The heat exchanger model has various efficiency, which changes based on the UA value under off design conditions. The part-load UA value in this model is based on the ratio of the fluid mass flowrate to its nominal flowrate. The principles of the heat exchanger model is shown as the following equations,

$$PLR_1 = \dot{m}_1/\dot{m}_{0,1}, \tag{1}$$

$$PLR_2 = \dot{m}_2/\dot{m}_{0,2}, \tag{2}$$

$$UA = UA_0 * PLR_1^a * PLR_2^b, \tag{3}$$

$$C_{min} = min(\dot{m}_1 * Cp_1, \dot{m}_2 * Cp_2), \tag{4}$$

$$C_{max} = max(\dot{m}_1 * Cp_1, \dot{m}_2 * Cp_2), \tag{5}$$

$$NTU = UA/C_{min}, \tag{6}$$

$$C_r = \frac{C_{min}}{C_{max}}, \tag{7}$$

$$Q_{max} = C_{min} * (T_{i1} - T_{i2}), \tag{8}$$

$$eff = f(NTU, C_r), \tag{9}$$

$$Q = eff * Q_{max}, \tag{10}$$

where PLR is the part load ratio, expressed as normalized flow rate, $\dot{m}$ is the mass flow rate. The subscripts 0,1 and 2 represent nominal condition, fluid 1 and fluid 2 respectively. C_{min}, C_{max} and C_r are the minimum, maximum and ratio of the flow thermal capacity. Cp is the specific thermal capacity. NTU and eff denote the number of heat transfer units and effectiveness. The function f expresses the relationship between NTU,C_r and eff for different flow configurations. Q is the heat transfer rate and Q_{max} is the maximum possible heat transfer rate. The function f is a formula related to the flow configuration of the heat exchanger, which can be referred in (DoE, 2010). T_i is the inlet temperature. The superscripts a and b are the curve coefficients.

3.1.3 Cooling Tower

The model is based on the York regression model in the Modelica Buildings library and it is integrated with an electric heater to provide auxiliary heating for the cooling tower when the weather is very cold. To compute the thermal performance, this model takes as parameters the approach temperature, the range temperature and the inlet air wet bulb temperature at the design condition. Along with the design mass flow rate (of the chiller condenser loop) as parameter, these parameters define the rejected heat. For off-design conditions, the model uses the actual range temperature and a polynomial to compute the

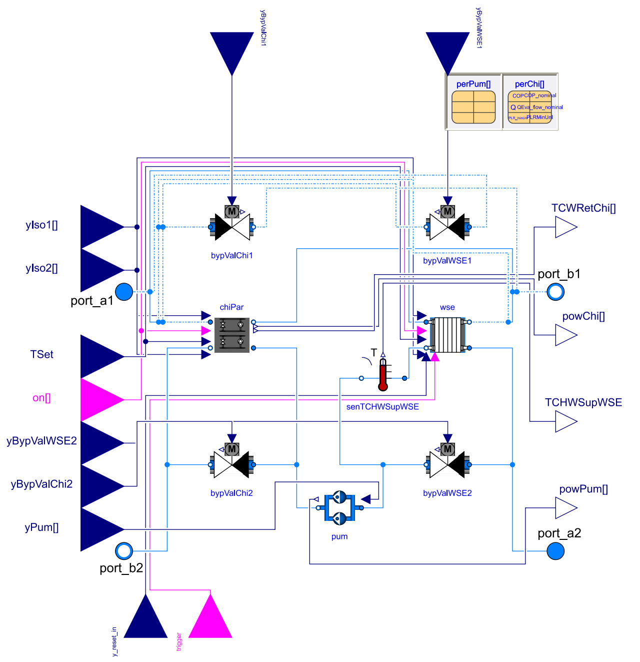


Figure 2. Implementation of the two piping configurations on the condenser water side

approach temperature for free convection and for forced convection, i.e., with the fan operating.
The heater is controlled by an on/off controller. The controller is activated when the outlet temperature of cooling tower is lower than the anti-freezing temperature setpoint $antFreTem$. Otherwise, the heater is deactivated. For the details of cooling tower and electric heater model, please see the model in the Modelica Buildings library.

3.1.4 Subsystem

In the following section, we implemented two configurations of integrated WSE on the load side of the primary-only chilled water system, with both series and parallel piping on condenser waterside and series piping on chilled water side. A partial model is built which could link the piping connection based on the condition whether it is series or parallel configuration on the condenser side. That said, the selection of the piping on the condenser waterside is achieved by setting the Boolean variable $parallelCondenserWater$ by extending this partial model. The implementation of this model is depicted in Figure 2. The outputs of this model can be the temperature of leaving condenser water in chillers, electric power consumed by chiller compressor, chilled water supply temperature in the waterside economizer and the electrical power consumed by the pumps. Users can model multiple chillers with only one integrated WSE.

Integrated in Series on Condenser Water Side The series piping on condenser waterside is achieved by extending the partial model discussed in Section 3.1.4 and setting $parallelCondenserWater$ to false.

Integrated in Parallel on Condenser Water Side The parallel piping on condenser waterside is achieved by extending the partial model discussed in Section 3.1.4 and setting *parallelCondenserWater* to true.

3.2 Control System

In this section, the implementation of cooling model controls and equipment-level local controls is described respectively.

3.2.1 Cooling Mode Control

Based on the operational status of chillers and WSEs, the chiller plant with WSEs can operate in three cooling modes, including Free Cooling (FC) mode when only WSEs are enabled for cooling, Partial Mechanical Cooling (PMC) mode when both chillers and WSEs are triggered, and Full Mechanical Cooling (FMC) mode when only chillers are activated. To consider the condition when the system is off, we introduce an unoccupied mode. Different sets of implementable control sequences for transitioning among different cooling modes are available in literature and industry applications for the chiller plant with WSEs. In this paper, we use one control logic from Meakins and Griffin from their research to showcase the implementation of the cooling mode controller using state-graph package in Modelica. Figure 3 depicts the schematics of the control sequence in the form of a state graph as well as the implementation of this cooling mode controller. It is noted that this general implementation approach could be extended and applied to other sets of the cooling mode controller.
As shown in Figure 3(a), the initial state of the cooling system is in unoccupied mode, where the whole system is off. When the system is operated, it will transition to the FC mode. The chiller is switched on if

$$T_{sup,CW,WSE} > T_{sup,CHW,set} \, and \, \Delta t_{chiller,off} > \Delta t_{thr}, \quad (11)$$

and switched off if

$$T_{sup,CW,WSE} < T_{sup,CHW,set} \, and \, \Delta t_{chiller,on} > \Delta t_{thr}, \quad (12)$$

where $T_{sup,CW,WSE}$ is the supply condenser water temperature, $T_{sup,CHW,set}$ is the supply chilled water temperature setpoint, $\Delta t_{chiller,on/off}$ is the elapsed time since the chillers were on/off, and Δt_{thr} is a waiting time threshold. The WSE is enabled when

$$T_{sup,CW,WSE} < T_{ret,CHW,WSE} - 3^oF \, and \, \Delta t_{WSE,off} > \Delta t_{thr}, \quad (13)$$

and switched off if

$$T_{sup,CW,WSE} > T_{ret,CHW,WSE} + 2.5^oF \, and \, \Delta t_{WSE,on} > \Delta t_{thr}, \quad (14)$$

where $T_{ret,CHW,WSE}$ is the chilled water return temperature upstream of WSE, $\Delta t_{WSE,on/off}$ is the elapsed time since the WSE was on/off, and Δt_{thr} is a waiting time threshold. 2.5^oF or -3^oF is an adjustable offset temperature difference.
Figure 3(b)shows the Modelica implementation of the above control sequence. On the left are the connectors for the control input signals expressed as real and boolean number, such as *TSupCWWSE* (Supply condenser water temperature at downstream of WSE), *TRetCHWWSE* (Return chilled water temperature at downstream of WSE), *TSupCHWSet* (Suppy chilled water temperature setpoint) and occupancy mode *uOcc* (True if the room is occupied). The modules *timGre* and *timLes* count and output the time when the signal u is over/below the offset temperature difference as shown in Eq.(11) - Eq.(14).In the middle is the state graph implemented in Modelica State graph 2. There are four states in this controller, indicated by the squared block icons. The states are FC mode, PMC mode, FMC mode, and the unoccupied mode. The initial state is set to the unoccupied mode when simulation starts. The transitions between the states are represented by the horizontal black bars, and each transition has exactly one preceding state and one succeeding state, and is set accordingly based on the transition conditions. On the right are the *mulSwi* module which converts the stage number to the output *y*.

3.2.2 Equipment-level Control

Chiller plants with WSEs require different equipment-level local controls of components compared with conventional chiller plants without WSEs. The selection of the local controls for different components is critical to the performance of the whole system. The local controls we selected are based on the inputs from the dicussion with the experienced industry engineers in this field. In this section, the equipment-level control sequences are described.

Chiller Control The chiller control involves a load-based stage control. Stage control of chiller is usually based on the supply chilled water temperature, chiller loads or partial load ratio (PLR), and chilled water pump speed. In this logic, chillers are

- staged up if current stage has been active for at least 30 minutes and the PLR for any active chiller is greater than 80 % for 10 minutes
- staged down if current stage has been activated for at least 30 minutes and the PLR for any active chiller is less than 25 % for 15 minutes.

It is noted that the threshold and delay time are adjustable.

Chilled Water Pump Controls The chilled water pump controls involve a stage control and a speed control.
Chilled water pumps stage up or down based on measured flowrate. The thresholds and delay time are also adjustable as shown in the following logic. Pumps are

- staged up if current stage has been active for at least 15 minutes and the measured flowrate is larger than

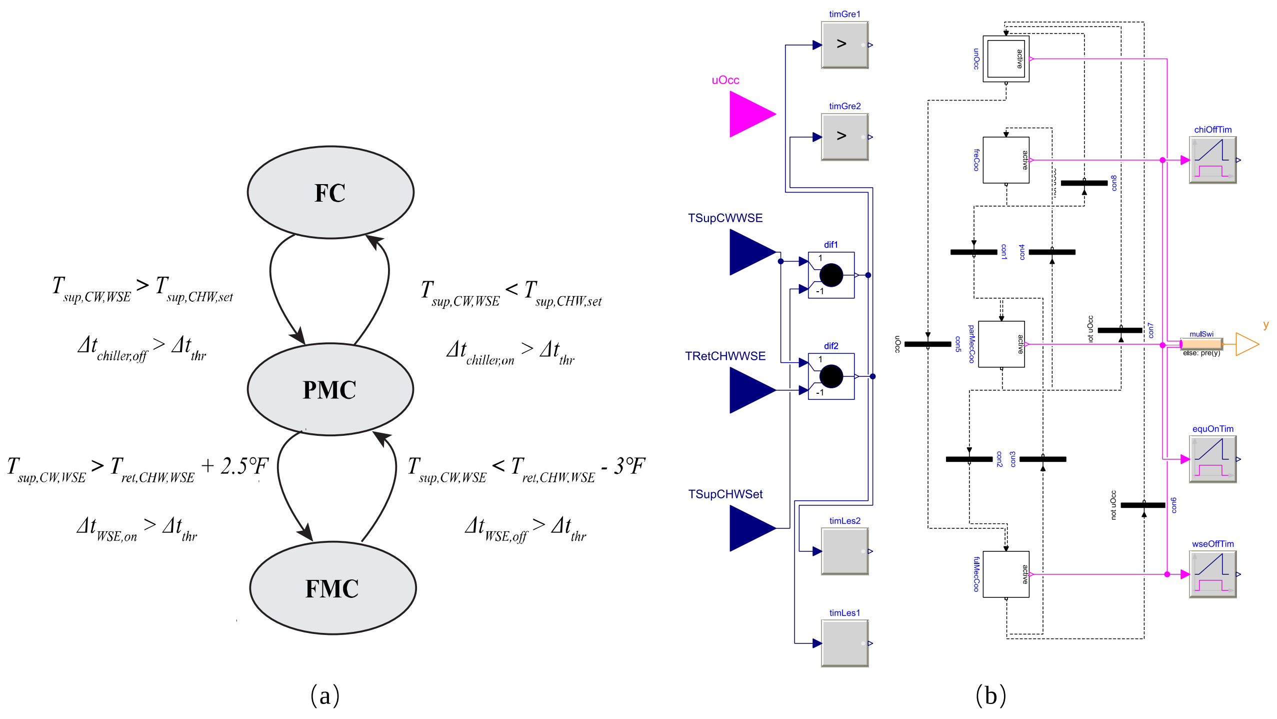

FC
PMC
FMC
$T_{sup,CW,WSE} > T_{sup,CHW,set}$
$\Delta t_{chiller,off} > \Delta t_{thr}$
$T_{sup,CW,WSE} < T_{sup,CHW,set}$
$\Delta t_{chiller,on} > \Delta t_{thr}$
$T_{sup,CW,WSE} > T_{ret,CHW,WSE} + 2.5°F$
$\Delta t_{WSE,on} > \Delta t_{thr}$
$T_{sup,CW,WSE} < T_{ret,CHW,WSE} - 3°F$
$\Delta t_{WSE,off} > \Delta t_{thr}$
(a)
uOcc
TSupCWWSE
TRetCHWWSE
TSupCHWSet
timGre1
timGre2
dif1
dif2
timLes2
timLes1
chiOffTim
mulSwi
equOnTim
wseOffTim
y
(b)

Figure 3. Implementation of the cooling mode control (a) Schematics (b) State-graph in Modelica

85% of the total nominal flowrate of the active pumps for 2 minutes.

- staged down if current stage has been active for at least 15 minutes and the measured flowrate is less than 45% of the total nominal flowrate of the active pumps for 15 minutes.

The chilled water pump speed is adjusted by a PID loop to maintain the differential pressure (DP) signal at a setpoint. All pumps receive the same speed signal. Minimum speed setpoint is prescribed based on the system configuration.

Temperature Reset Control In this control logic, both the chilled water supply temperature setpoint and the chilled water loop DP setpoint are reset based on the difference between the actual and set temperature of the supply air in the part load condition. A single reset control point should be used to control both setpoints. When the plant includes a WSE, the reset should lead with pressure to keep the water temperature as high as possible to maximize WSE operation (see Figure 4). The setpoint reset strategy is to first increase the different pressure DP of the chilled water loop to increase the mass flow rate. If DP reaches the maximum value and further cooling is still needed, the chiller temperature setpoint, $T_{sup,CHW,set}$ is reduced. If there is too much cooling, $T_{sup,CHW,set}$ and DP will be changed in the reverse direction.

Cooling Tower Controls The local controls related to the cooling tower are the cell stage control and fan speed control.

In the cell stage control, the tower cells are staged based on measured condenser water flowrate.

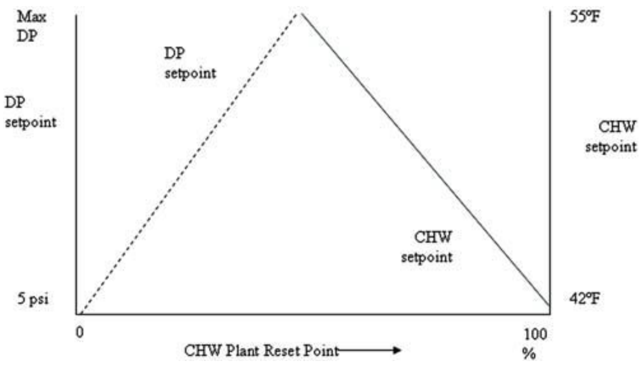


Figure 4. Principle of temperature reset control

- One additional cell stages on if average flowrate through active cells is greater than a stage-up threshold for 15 minutes
- One additional cell stages off if average flowrate through active cells is lower than a stage-down threshold for 5 minutes.

A minimum tower cell number reset control based on the different cooling modes is commonly used to support the stage control.

- In unoccupied mode, the minimum number is 0.
- In FC mode, the minimum number of active cooling towers should be equal to the number of active condenser water pumps.
- In PMC mode, the minimum number of active cooling towers should be equal to the total number of

cooling towers, which means all the cooling towers should be commanded on.

- In FMC mode, the minimum number of active cooling towers should be equal to the number of active condenser water pumps.

The cooling tower fan speed is also controlled to satisfy the requirement of temperature setpoint, and fan speed shall not exceed the maximum speed.

- In unoccupied operation mode, the fan is turned off.
- In FC mode, the fan speed is controlled to maintain a predefined chilled water supply temperature at the downstream of the economizer, and not exceed the predefined maximum fan speed.
- In PMC and FMC modes, the fan speed is controlled to maintain the supply condenser water at its setpoint.

The maximum fan speed is reset based on cooling modes and operation status.

- In unoccupied mode, the maximum speed is not reset.
- In FC mode, if all condenser pumps are enabled, the maximum fan speed is reset to full speed 100%; otherwise the maximum fan speed is reset to a lower speed, e.g. 90%.
- In PMC mode, the maximum fan speed is set to a high speed, e.g. 100%.
- In FMC mode, if all the condenser water pumps are active, the maximum fan speed is reset to full speed 100%; otherwise it is reset to a lower speed, e.g. 90%.

Condenser Water Pump Controls The condenser water pump controls include a pump stage control and a speed control.
The condenser water pump stage control have different logics according to operational mode. Here assume the number of condenser pumps and that of the chillers are identical.

- In unoccupied mode, the condenser pump should be turned off.
- In FC mode, the number of operating condenser water pumps is staged based on the cooling tower fan speed signal.When the fan speed signal exceeds 80% for 5 minutes, then stage up one condenser pump. When the fan speed signal is below 45% for 10 minutes, then stage down one condenser pump.
- In PMC mode, if not all chillers are active, the number of active condenser water pumps should be one less than the total condenser water pumps, else all the condenser pumps are commanded to run.
- In FMC mode, the number of active condenser water pumps should be equal to the number of active chillers.

The condenser water pump speed control sequence is shown as follows.

- In unoccupied mode, the pump is turned off.
- In FC mode, the condenser water pump speed is equal to a high signal select of a PID loop output and a minimum speed (e.g. 40%). The PID loop outputs the cooling tower fan speed signal to maintain chilled water supply temperature at its setpoint.
- In PMC and FMC modes, the condenser water pump speed is equal to a high signal select of the following three: (1) a minimum speed (e.g. 40%); (2) highest chiller percentage load; (3) CW system differential pressure PID output signal.

Valve Controls Four local controls related to the valves in the cooling system are considered. They are controls for the modulating isolation valve, the two position valves, the chiller bypass valve, and the WSE bypass valve.
Modulating Isolation Valve Control. The modulating isolation valve is usually installed on the condenser water side of a chiller to perform head pressure control during cold climate. The modulating isolation valve is modulated to control variables such as chiller temperature/pressure lift between evaporator and condenser. The logics for different operational modes are as follows.

- When chiller is not enabled, the isolation valves are closed.
- When chiller is enabled, the isolation valves on the condenser side are fully opened at the beginning. Then after a stable time delay, the valve is modulated to control variables such as temperature lift or pressure lift in a chiller.

Two Position Valve Control. The two position valve controls the on/off of the equipment.
Chiller Chilled Water Bypass Control. Bypass valves are used to switch the cooling system among different operation modes, and maintain desired differential pressure through the equipment such as evaporators or condensers. The control sequence of chiller chilled water bypass is as follows.

- In unoccupied operation mode, the bypass for chillers and economizers are closed.
- In FC mode, the bypass valve on chiller side is fully opened.
- In PMC mode, the chiller bypass valve is modulated to maintain the differential pressure through the active evaporators at its setpoint such as design differential pressure.

- In FMC mode, the chiller bypass is modulated to maintain the differential pressure through the active evaporators at its setpoint such as design differential pressure.

WSE Chilled Water Bypass Control. Bypass valves for economizers are used to switch the cooling system among different operation modes, and maintain desired differential pressure through economizers. This model can be used for both chilled and condenser water side. The control sequence of WSE chilled water bypass is as follows.

- In unoccupied operation mode, the bypass valves for economizers are closed.
- In FC mode, the bypass valve on economizer side is modulated to maintain differential pressure across their respective economizers at a setpoint, such as design differential pressure.
- In PMC mode, the bypass valve on economizer side is modulated to maintain the differential pressure through the economizer at a setpoint such as design differential pressure.
- In FMC mode, the bypass valve on economizer side is fully opened.

4 Case Study

4.1 Case Description

We perform a case study utilizing Modelica models to comprehensively evaluate performance of the cooling mode control logic mentioned in Section 3.2.1 in all ASHRAE climate zones. There are in total 19 thermal climate zones in ASHRAE Stanadard 169-2013, which categorizes different areas based on their historical weather data such as heating degree days, average temperatures, and precipitation. The climate zones and their representative cities are shown in Table 1. The index 0 means extremely hot area, while index 8 means arctic area. The index A, B and C mean humid, dry and marine area.

Table 1. ASHRAE climate zones

Index	*City*	*Index*	*City*
0A	Singapore	4B	Albuquerque
0B	Riyadh	4C	Salem-McNary
1A	Miami	5A	Chicago
1B	Kuwait	5B	Boise
2A	Houston	5C	Bremerton
2B	Phoenix	6A	Burlington
3A	Memphis	6B	Helena
3B	EI Paso	7	Duluth
3C	San Francisco	8	Fairbanks
4A	Batimore		

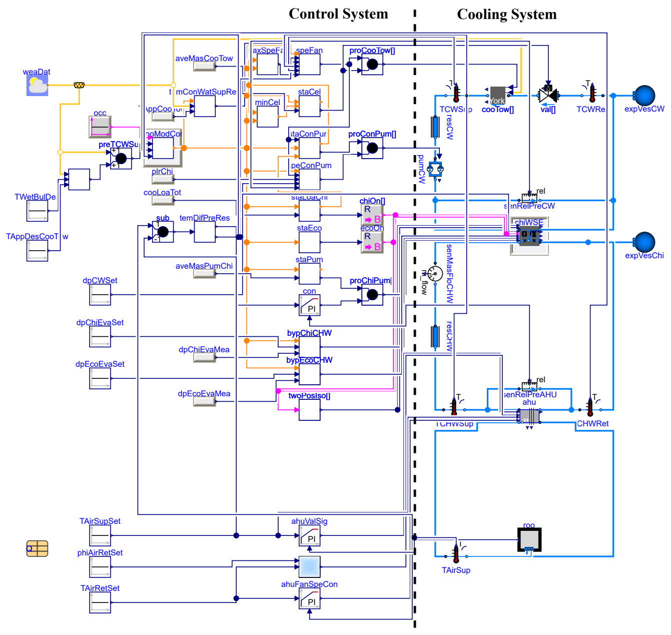


Figure 5. Modelica implementation of a chilled water system with waterside economizer

The Modelica model is built as shown in Figure 5. The chilled water system with waterside economizer is designed for a virtual data center. The data center room has a nominal cooling load of 2000kW but operates at a part load ratio 0.65. Two identical chillers with a cooling capacity of 1000kW for each, and one WSE that can provide 2000kW under the design condition are provided as cooling sources. The waterside economizer is integrated with chillers on the chilled water side, and in parallel with chillers on the condenser water side. The pumps, cooling towers and air handling units are sized accordingly based on the design cooling load. The cooling mode and equipment-level control are described in Section 3.2.

4.2 Simulation Results

An annual simulation is performed for each climate zone. The energy consumption of the cooling system are shown in Figure 6. The economizing hours in different climate zones when the WSE is activated to provide cooling are shown in Figure 7.

Both the dry bulb temperature and humidity can influence the energy consumption of the chilled water system with WSEs. In the humid area (A), as the dry bulb temperature decreases from 0A to 6A, the total energy decreases from 151 MWh to 139 MWh. This means with, similar humidity, the higher the dry bulb temperature is, the more energy is consumed by the cooling system. The reason is with a higher outdoor dry bulb temperature, the cooling system has less economizing hours during annual operation, as shown in Figure 7. In hot climate zone 1, as the humidity reduce, the cooling system operates under more economizing hours. The reason is that given similar dry bulb temperature, dryer air means lower wet bulb temperature. As a consequence, the cooling towers can

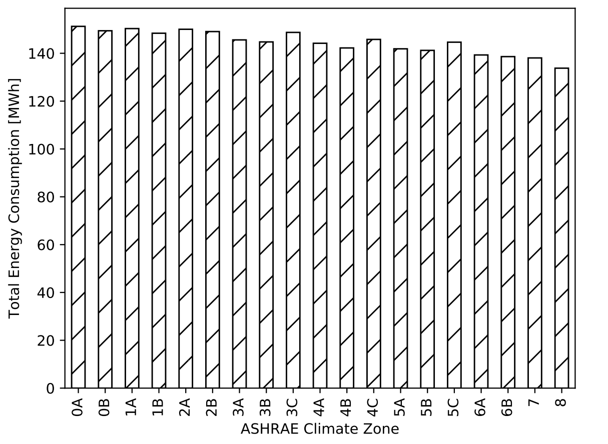


Figure 6. Total energy consumption for the cooling system in all climate zones

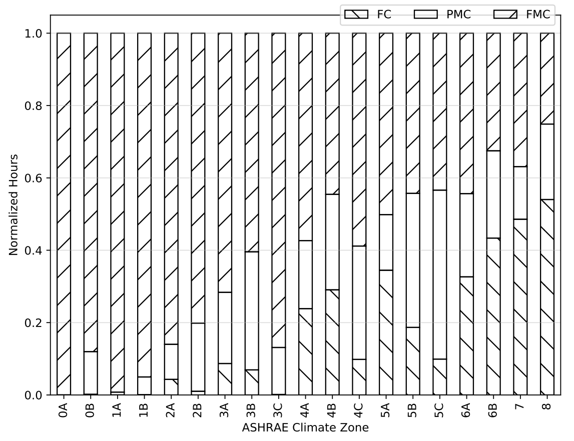


Figure 7. Normalized economizing hours in all climate zones

produce cold condenser water temperature for a longer period, which then can help extend the economizing hours.

5 Conclusions

We presented a Modelica-based tool for the modeling and simulation of the chilled water system with waterside economizers. The cooling component models and control models are also discussed in details. These Modelica models are then used to perform a case study to evaluate the energy and control performances of the cooling mode control in different climate zones. Simulation results show that the cooling mode control has different performances in different climate zones. Both the temperature and the humidity can influence the system performance.

6 Acknowledgement

This research was supported by the American Society of Heating, Refrigeration, and Air-conditioning Engineers under the research project TRP-1661, and the National Science Foundation under Award No. IIS- 1633338.

References

Aayush Agrawal, Mayank Khichar, and Sanjeev Jain. Transient simulation of wet cooling strategies for a data center in worldwide climate zones. *Energy and Buildings*, 127:352–359, 2016.

US DoE. Energyplus engineering reference. *The reference to energyplus calculations*, 2010.

Hilding Elmqvist, Sven Erik Mattsson, and Martin Otter. Modelica: The new object-oriented modeling language. In *12th European Simulation Multiconference, Manchester, UK*, 1998.

Yangyang Fu, Wangda Zuo, Michael Wetter, James VanGilder, Xu Han, and David Plamondon. Equation-based object-oriented modeling and simulation for data center cooling: A case study. *accepted by Energy and Buildings*.

Yangyang Fu, Michael Wetter, and Wangda Zuo. Modelica models for data center cooling systems. In *2018 Building Performance Analysis Conference and SimBuild, Chicago, Illinois, United States of America*, 2018.

Sang-Woo Ham and Jae-Weon Jeong. Impact of aisle containment on energy performance of a data center when using an integrated water-side economizer. *Applied Thermal Engineering*, 105:372 – 384, 2016.

Sen Huang, Wangda Zuo, and Michael D Sohn. Amelioration of the cooling load based chiller sequencing control. *Applied Energy*, 168:204–215, 2016.

Donghun Kim, Wangda Zuo, James E Braun, and Michael Wetter. Comparisons of building system modeling approaches for control system design. In *Proceedings of the 13th International Conference of the International Building Performance Simulation Association (Building Simulation 2013)*, pages 3267–3274. Citeseer, 2013.

Kuei-Peng Lee and Hsiang-Lun Chen. Analysis of energy saving potential of air-side free cooling for data centers in worldwide climate zones. *Energy and Buildings*, 64:103–112, 2013.

Yiqun Pan, Rongxin Yin, and Zhizhong Huang. Energy modeling of two office buildings with data center for green building design. *Energy and Buildings*, 40(7):1145–1152, 2008.

Arman Shehabi, Srirupa Ganguly, Kim Traber, Hillary Price, Arpad Horvath, William W Nazaroff, and Ashok J Gadgil. Energy implications of economizer use in california data centers. Technical report, Lawrence Berkeley National Laboratory, 2008.

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

Michael Wetter, Marco Bonvini, and Thierry S. Nouidui. Equation-based languages – a new paradigm for building energy modeling, simulation and optimization. *Energy and Buildings*, 117:290 – 300, 2016.

Predicting the Vehicle Performance at an Early Stage of Development Process via Suspension Bushing Design Tool

Sooncheol Park[1] Yonggwon Jeon[1] Dae-oh Kang[2] Min-su Hyun[3] Seung-jin Heo[3]

[1]Hyundai Motor Company, Republic of Korea, {testdrv,yonggwon.jeon}@hyundai.com
[2]Institute of Vehicle Engineering, Republic of Korea, bigfive@ivh.com
[3]School of Automotive Engineering, Kookmin University, Republic of Korea, {slay,sjheo}@kookmin.ac.kr

Abstract

This paper describes a method for verifying vehicle performance when applying a new suspension bushing at the concept phase of vehicle development. At the concept phase, it is difficult to obtain the nonlinear characteristics of the bushing, which plays an important role in the performance of the vehicle. Thus, a tool to design bushing has been developed.

A method that designers can intuitively and easily design a bush is proposed. It combines the results of the developed bushing design tool with a Modelica system model to evaluate NVH, R&H, and durability performance. Designers can create a new bushing using the bushing design tool and check the vehicle performance at an early stage of development process. The developed bushing design tool allows designers to evaluate vehicle performance by reflecting bushing characteristics without actual products.

Keywords: suspension bushing, bushing design, vehicle dynamics, parameter identification.

1 Introduction

The vehicle development cycle in the automotive industry is getting shorter and shorter. To cope with shorter development cycle, it is important to achieve vehicle performance such as NVH (Noise, Vibration and Harshness), R&H (Ride and Handling) and durability performance in the concept phase as shown in Fig. 1. Each performance must be designed in harmony with each other. To be able to verify this performance, it must be possible to carry out test without actual vehicles and parts, because tests using actual vehicles and parts are time-consuming, costly and effort-intensive. In other words, a simulation model that can reflect actual phenomena must be constructed in the concept phase.

For the vehicle model, a variety of force elements and material properties are needed. For example, vehicle performance is affected by static stiffness, dynamic stiffness and loss angle of suspension bushing and engine mount, shock absorber properties, boll-join properties and tire properties. Thus, the reliability of the results is determined by the consistency of the force components that make up the model. These components have nonlinear properties and usually can be obtained through component tests. However, it is difficult to obtain physical properties at the concept stage. Also, since there is no actual component, it is difficult to apply it to various characteristics.

In this paper, new characteristics of suspension bushing are determined using a bushing design tool written in-house code. To verify vehicle performance, the results of the bushing design tool and a Modelica system model are used.

2 Bushing design tool

Suspension bushing is described briefly. It is explained the bushing design tool for designers. It is included the bushing model, parameter identification, and bushing design method. Based on relevant technology, bushing design tool has been made with in-house code.

2.1 Suspension bushing

The suspension components are connected to each other, to the subframe, and to the body structure via rubber bushings. They are a key element in designing desired static and dynamic behavior of suspension system. The dynamic characteristics of a rubber bushing are often very complex and nonlinear because the response is dependent on several variables, such as frequency, amplitude, preload, and temperature.

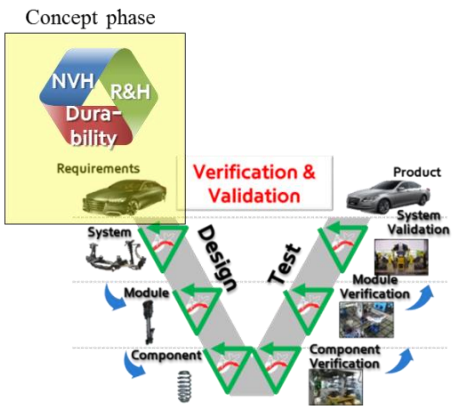


Figure 1. Vehicle development cycle and concept phase.

2.2 Bushing model & Parameter identification

A generalized linear elastic viscoelastic elastoplastic model (generalized Maxwell model) is used as a bushing model. It contains several Maxwell elements and frictional elements as shown in Fig. 2. Each element is called as a cell. This component model is able to model both frequency and amplitude dependent rubber behavior such as dynamic stiffness and phase angle.

The bushing model has several unknown parameters. In order to obtain the unknown parameters, optimization algorithm is applied to the bushing model. Optimization is searching process to find a minimum value for a certain function which expresses the sum of the relative error compared to the target value.

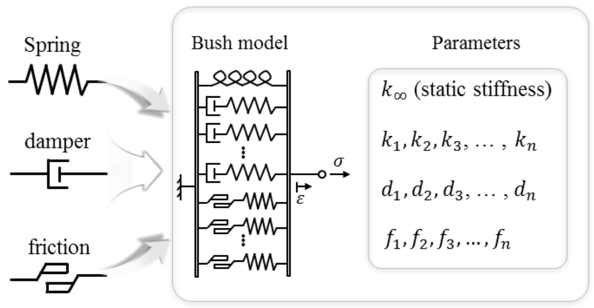

Spring
damper
friction
Bush model
Parameters
k_∞ (static stiffness)
$k_1, k_2, k_3, \dots, k_n$
$d_1, d_2, d_3, \dots, d_n$
$f_1, f_2, f_3, \dots, f_n$
σ
ε

Figure 2. Generalized linear elastic viscoelastic elastoplastic model with n-cells and its parameters.

2.3 Bushing design method

The interface of the developed bushing design tool is shown in Fig. 3. It consists of target setup, model setup, optimization, and model parameters as an output.

At the target setup, designers use test data of original bushing as an input of bushing design tool. Next, designers determine the number of cells for the bushing model at the model setup. Then, the bushing design tool proceeds with the parameter identification through an optimization process based on the input test data and the number of cells of the bushing model. It is observed that the optimized results are within 10% error.

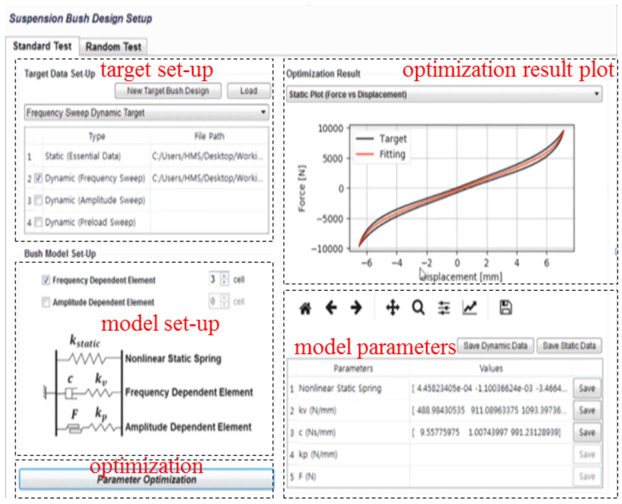
Designers then determine the new bush characteristics which have the desired value based on original bushing model. They modify static and dynamic stiffness curve as shown in Fig. 4 and Fig 5. The bushing design tool calculates the parameters of the model according to the new bushing characteristics as shown in Fig. 6. When verifying vehicle performance, it is used the bushing model parameters from bushing design tool.

3 Modelica based model

It is explained how to apply the new bushing characteristics calculated by the bushing design tool to a Modelica system model. A bushing model which has six degrees of freedom is constructed. Vehicle performance is verified by applying the bushing model to the suspension model. Also additional test rigs were constructed.

3.1 Suspension bushing model

The unidirectional bushing model was explained in section 2. However, a bushing has six degrees of freedom actually. Therefore, the model for three translational and three rotational directions is constructed as shown in Fig. 7. Each directional model is composed of generalized Maxwell model.



	Type	File Path
1	Static (Essential Data)	C:/Users/HMS/Desktop/Worki...
2	☑ Dynamic (Frequency Sweep)	C:/Users/HMS/Desktop/Worki...
3	☐ Dynamic (Amplitude Sweep)	
4	☐ Dynamic (Preload Sweep)	

	Parameters	Values	
1	Nonlinear Static Spring	[4.45823405e-04 -1.10036624e-03 -3.4664...	Save
2	kv (N/mm)	[488.98430535 911.08963375 1093.39736...	Save
3	c (Ns/mm)	[9.55775975 1.00743997 991.23128939]	Save
4	kp (N/mm)		Save
5	F (N)		Save

Figure 3. Interface of suspension bushing design tool.

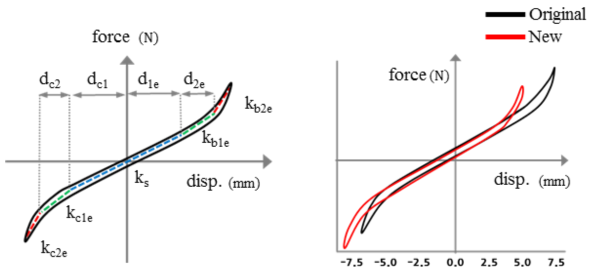


Figure 4. Determination of new static characteristics.

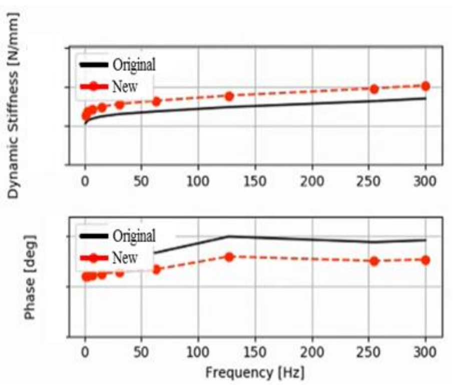


Figure 5. Determination of new dynamic characteristics.

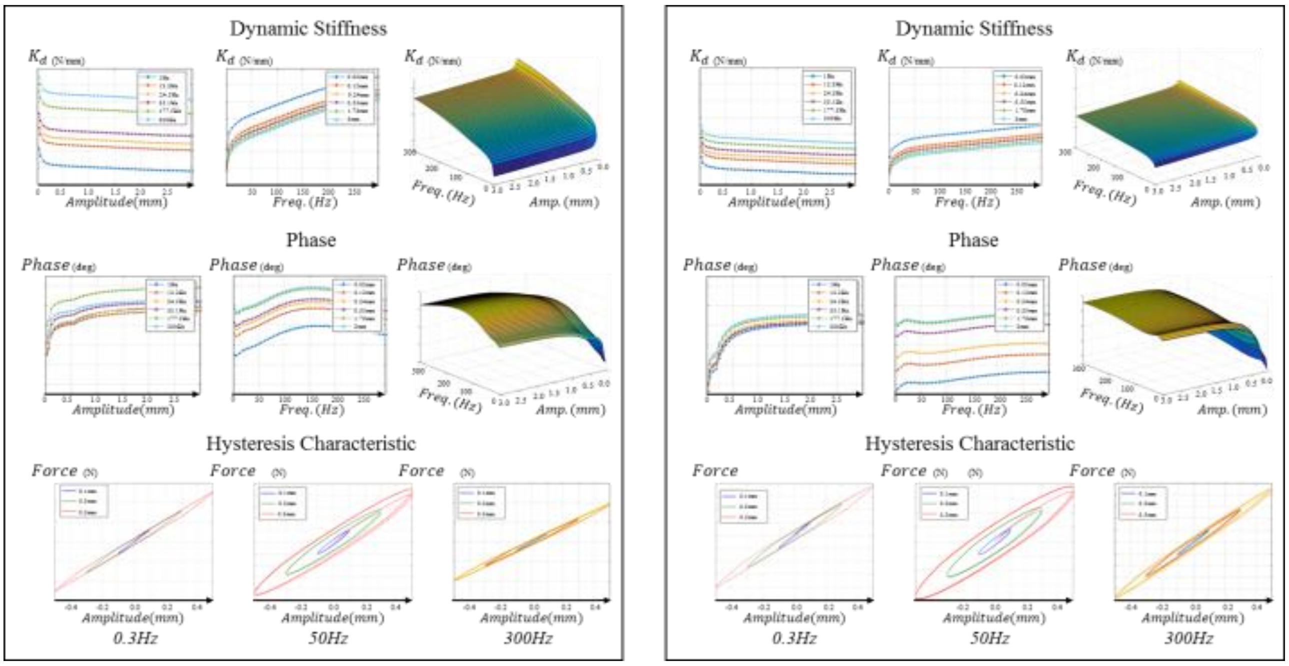


Figure 6. Characteristics of original bushing model (left side) and new bushing model (right side).

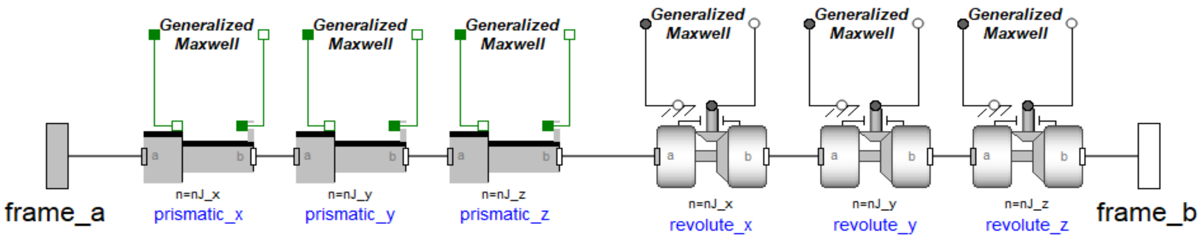


Figure 7. Six degrees of freedom suspension bushing model connected to each link.

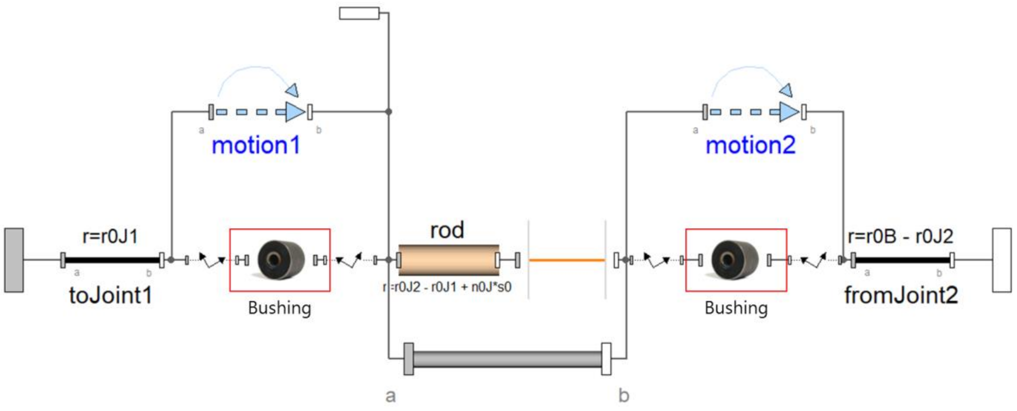


Figure 8. Upper arm and bushing model.

3.2 Suspension system model

The Vehicle Dynamics Library (VDL) of Modelon is applied as a suspension system model. The bushing models that make up the suspension are replaced with the generalized Maxwell model from the Kelvin-Voight model that is provided as standard. As shown in Fig. 8, the bushing is connected to the arm. Bushings and arms combine with other components to form a linkage model as shown in Fig. 9. The configured linkage model is coupled to the suspension model as shown in Fig. 10.

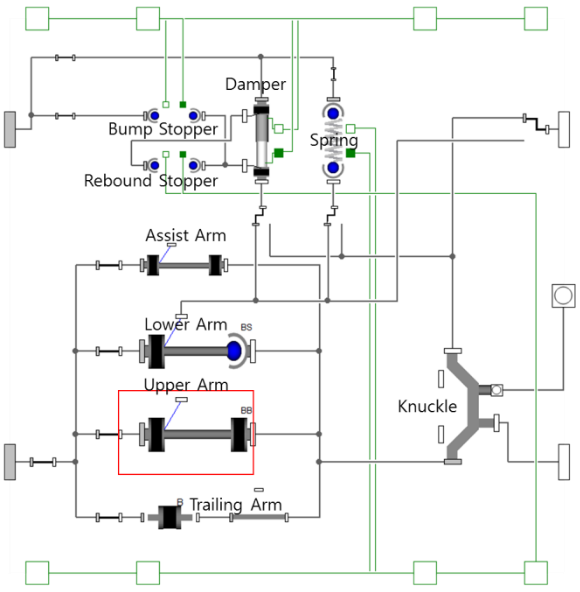


Figure 9. Trailing arm type multilink suspension linkage.

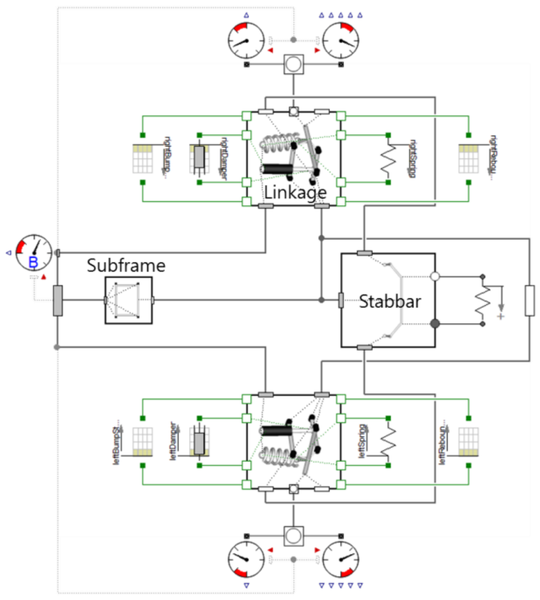


Figure 10. Trailing arm type multilink suspension model.

4 Simulation

Using the results of bushing design tool in section 2 and the models mentioned in section 3, the vehicle performance is evaluated in the view of durability, NVH and R&H.

4.1 Model for Durability and NVH

In order to verify vehicle performance in the view of durability and NVH, a model is constructed as shown in Fig. 11. The wheel load measured on the Belgian road is used as the input drive signal for the durability test. To verify NVH performance, white pink noise with a border frequency of 250 Hz and a cutoff frequency of 300 Hz is given as vertical displacement.

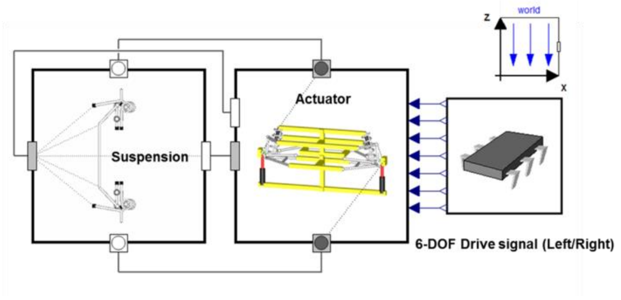


Figure 11. Trailing arm type multilink suspension linkage.

4.2 Model for R&H

For the step steer test, the model is constructed as shown in Fig. 12. Test conditions are applied based on ISO-7401 standard. A constant velocity value and the ramp type steering value with duration time of 0.15 sec are used as inputs.

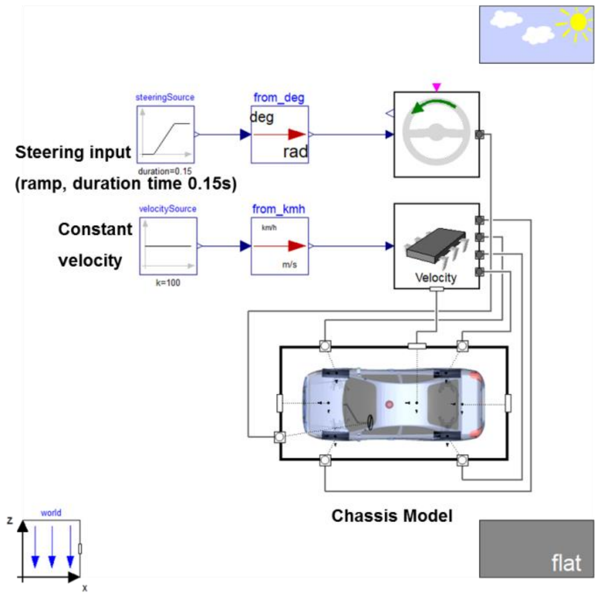


Figure 12. Step steer test model.

For the constant radius cornering test, the model is constructed as shown in Fig. 13. Test conditions are applied based on ISO-4138 standard. Closed loop steering input and velocity profile with duration time 600 sec are used as input values to make a turn radius of 50m.

For the four-post test, the model is constructed as shown in Fig. 14. ISO C-class road surface profile is used as drive signal.

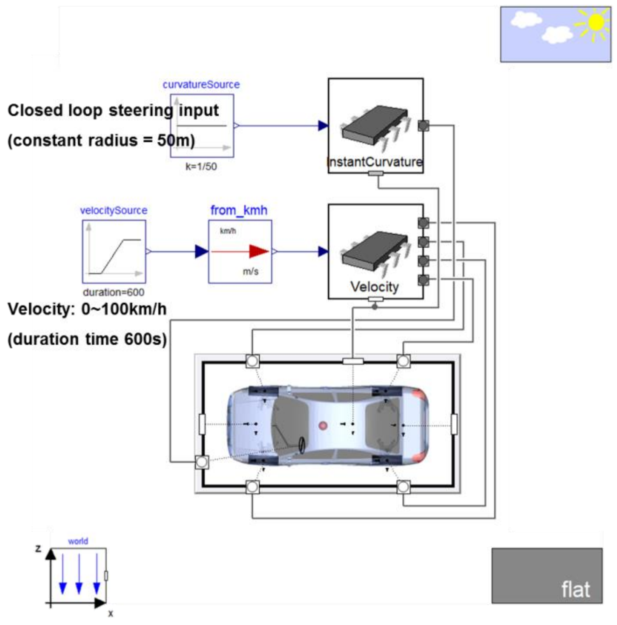


Figure 13. Constant radius cornering test model.

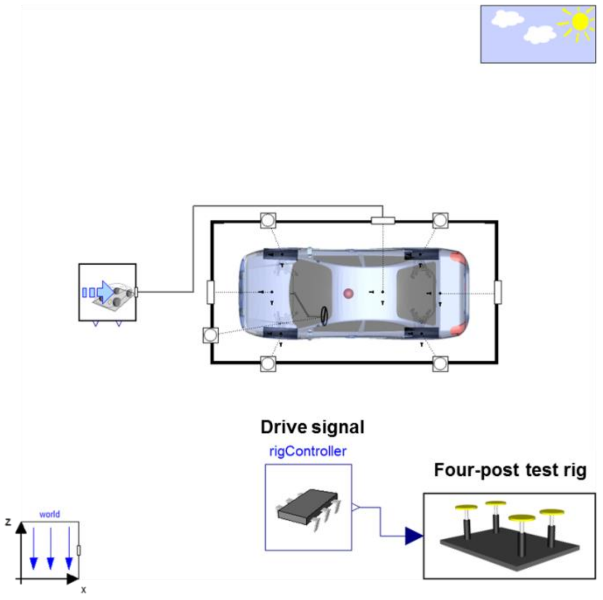


Figure 14. Four-post test model.

4.3 Simulation results

These are the simulation results while changing the characteristics of the suspension bushings such as lower arm bushing, upper arm bushing, assist arm bushing and trailing arm bushing.

For each result, the blue line is the result of the original bushing. The red line indicates the larger dynamic stiffness value than the original one. The green line indicates the smaller dynamic stiffness value than the original one.

4.3.1Durability and NVH

In the view of durability performance, the change in load applied to each bushing can be confirmed from the values of the time domain and the power spectral density of the frequency domain as shown in Fig. 15.

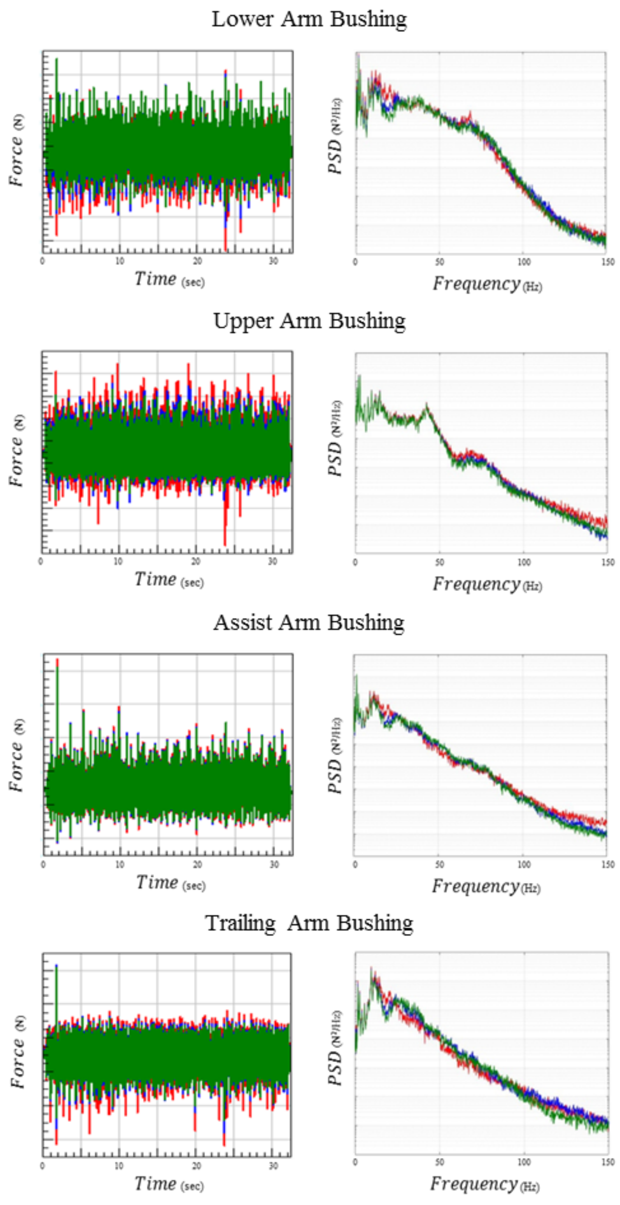


Figure 15. Durability performance

Similarly, the change of the load of each bushing and the acceleration of the vehicle body can be checked in the view of NVH as shown in Fig. 16.

4.3.2R&H

Fig. 17 depicts constant cornering test and step steer test results. The performance of handling can be confirmed by the changed values of the understeer gradient, response time and overshoot value as the new bushing is applied. From Fig. 18, ride performance can be checked through vehicle body acceleration analysis.

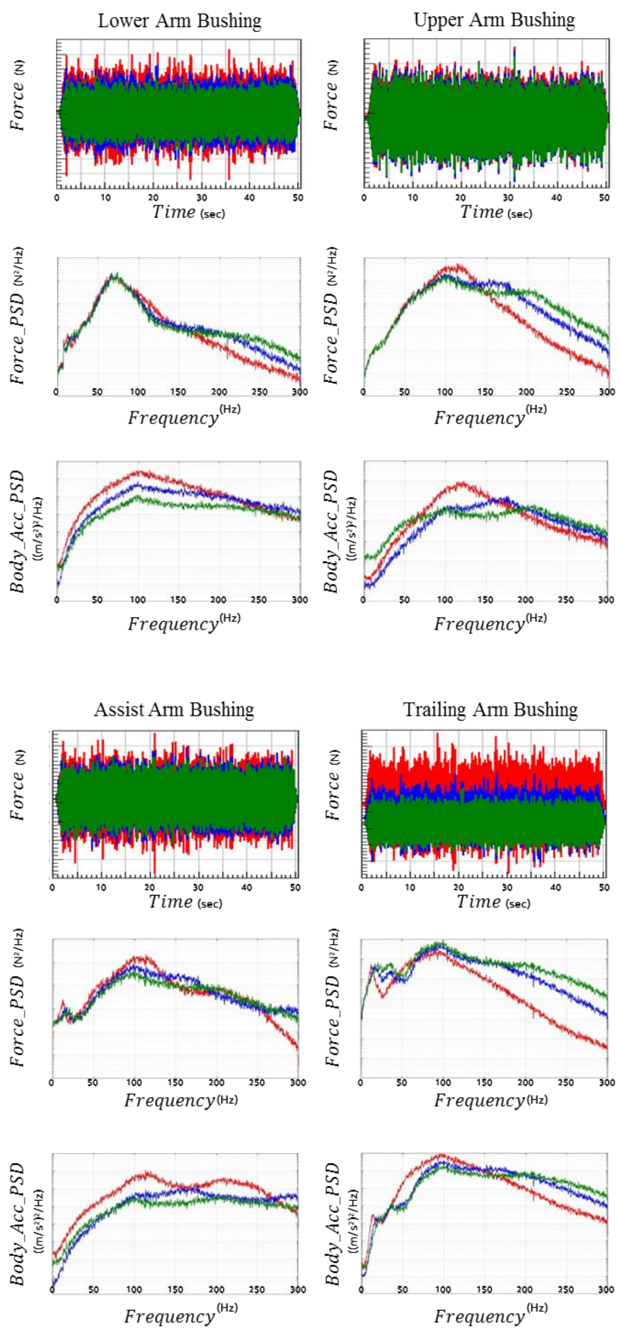
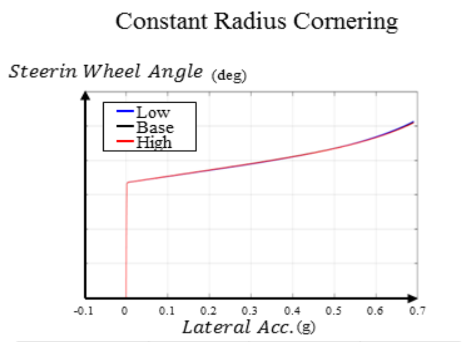
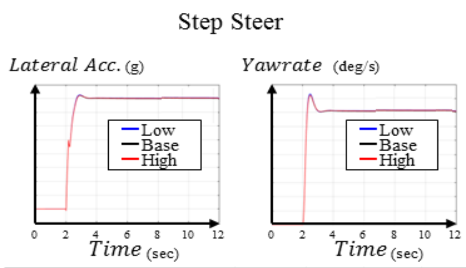


Figure 16. NVH performance



	Low	Base	High
Understeer gradient	+3.01%	-	+1.84%



Parameter		Unit	Low	Base	High
Steady-state Yaw rate Response Gain		1/s	+0.31%	-	+0.26%
Peak Response Time	Lateral Acc.	s	-2.06%	-	+0.97%
	Yaw Rate	s	-0.48%	-	+1.43%
90% Response Time	Lateral Acc.	s	-1.09%	-	+1.09%
	Yaw Rate	s	-0.51%	-	+1.03%
Overshoot Value	Lateral Acc.	%	+5.54%	-	-4.72%
	Yaw Rate	%	+4.63	-	-2.26%

Figure 17. Handling performance.

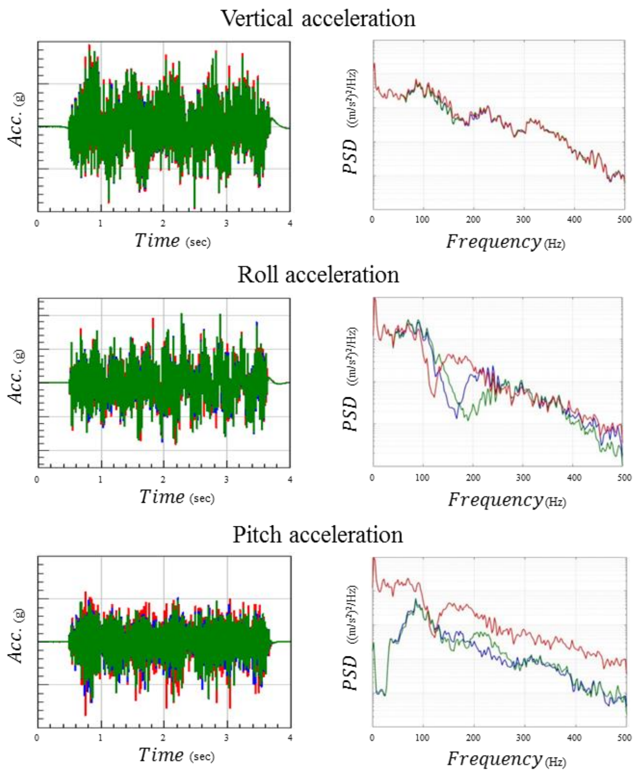


Figure 18. Ride performance.

5 Conclusion

In this paper, vehicle performance is verified at the concept stage by using the results of the developed bushing design tool and a Modelica system model. Designers can make various bushing characteristics in an intuitive and easy way using the bushing design tool. First, designers use original model’s test data as an input of bushing design tool and determine the number of cells for the bushing model. The bushing design tool proceeds with the parameter identification through an optimization process based on the input test data and the number of cells of the model. The designers then determine the new bushing characteristics to get the desired static and dynamic characteristics in the original characteristics. The bushing design tool calculates the parameters of the bushing model according to the bushing characteristics changed to the result value. From simulation results, the vehicle performance is changed according to the characteristics of the suspension bushing. The designers can confirm the vehicle performance at the concept stage if the bushing, determined by the designers, is applied.

The optimum bushing specification will be determined by optimizing the vehicle performance with the bushing characteristics in the future.

References

Fredrik Karlsson and Anders Persson. Modelling non-linear dynamics of rubber bushings –Parameter identification and validation . *Master’s thesis*, 2003.

Jun Nakahara, Koji Yamzaki, Yusuke Otaki. Rubber bushing model for vehicle dynamics performance development that considers amplitude and frequency dependency. *SAE Int. J. Commer:Veh*, 8(1):117–125, 2015. doi: 10.4271/2015-01-1579.

Kai Sedlaczek , Sven Dronka and Jochen Rauh. Advanced modular modelling of rubber bushings for vehicle simulations. *Vehicle system dynamics* 49(5):741–759, 2011. doi: 10.1080/00423111003739806.

Michael Fleps-Dezasse Jakub Tobolá˘r Johannes Pitzer. Modelling and parameter identification of a semi-active vehicle damper. *Proceedings of the 10th International ModelicaConference*, 2014. doi: 10.3384/ECP14096283.

Modelica-Based Modeling and Application Framework on Hybrid Electric Vehicles

Yuhui Liu [1] Liping Chen [1] Yan Zhao [2] Shanshan Liu [3] Fanli Zhou[2] Duansen Shangguan[1]

[1]School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan, China, {yuhuiliu, chenlp,Shanggds}@hust.edu.cn

[2]Suzhou Tongyuan Software & Control Technology Co., Ltd. Suzhou, China, {zhaoy,zhoufl}@tongyuan.cc

[3]Jianghuai Automobile Co., Ltd., Hefei, China, xjs-js@jac.com.cn

Abstract

In order to meet low-emissions criteria outlined in Chinese regulations, which require a progressively increasing percentage of automobiles to be ultralow or zero emissions, in this paper a sort of light hybrid electric drivetrain is studied and modeled in detail using Modelica. An application framework is also designed to improve the usability and the efficiency of the models. Performance of the whole vehicle and some key components are analyzed.. Comparison between simulation results and experiment results is performed, which validates the effectiveness of the models. Based on the comparison, we conclude that, the methods presented in this paper can support a rapid design of hybrid electric vehicles and further optimization.

Keywords: hybrid electric vehicles, Modelica, MWorks, application framework, signal bus

1 Introduction

Hybrid electric vehicles (HEVs) have attracted a great deal of attentions due to its high efficiency and less fuel consumption, which combines the advantages of both conventional internal combustion engine (ICE) and pure electric vehicles (Liang et al, 2016; Jionas et al, 2002). HEVs can be classified into Micro hybrid system, light hybrid system, medium hybrid system and complete hybrid system depending on the degree of colloquial in the hybrid system (Liu, 2014). Light hybrid system has been widely used due to its relatively simple structure and braking energy recovery (Zhang et al, 2014).

Basically, modeling and simulation methods of HEVs can be classified into the following three types. The first type is the traditional calculation method using excel where static results are obtained and characteristics of key components are rendered. The second type is that simulation models have already been realized based on commercial software such as advisor, AVL Cruise or Matlab libraries Toolbox (Karen et al, 1999; Meradji et al, 2016), where these models are focused on general and complex predefined. The third type is Modelica, which is suitable for dynamic analysis and developing components or function modules because of its support for multi-domain unified modelling, object-oriented physical modelling, non-causal modelling, and continuous discrete mixed modeling (Xiong et al, 2016).

In this paper, the structure of light hybrid drivetrain architecture of HEVs, is firstly studied and modeled in detail using Modelica, and an application framework is designed. Subsequently, the simulation results of the performance parameters of the whole vehicle and the key parts are obtained. Finally, the results are verified by experiment results.

2 System Descriptions

Before modeling the HEVs drive train represented in Fig.1 some characteristics of each element composing the system need to be mentioned:

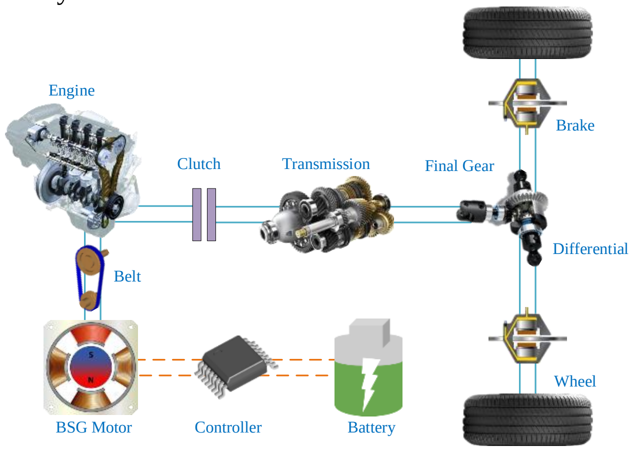


Figure 1. HEVs drive train structure.

Internal combustion engine characterized by torque-speed curve with optimal torque $T_{ICE, opt}$=252 N m, The main indicator of the internal combustion engine is power output.

BSG(Belt Driven Starter Generator) motor is a belt drive integrated power generation integrated machine. Its function is to achieve fast start engine, electronic idle speed and boost when outputting positive torque, and to obtain high efficiency intelligent power generation and braking energy recovery when negative torque.

Battery of Nickel-Metal-Hydride (NiMH) having a discharge power $P_{bat,dis}$=33700 W and a charge power $P_{bat,dis}$=26040 W, is responsible for outputting or storing electrical energy, connected to BSG and DC/DC converter. The main indicators of the battery module include SOC, the state of battery charge and discharge, battery output power, and the temperature impact on the battery module.

Transmission having five forward gears are used to transmit torque and speed, which should be gear ratio and efficiency, represented in Table 1.

Table 1. Speed ratio and efficiency of transmission.

Gears	1 st	2 st	3 st	4 st	5 st
Gear Ratios	3.769	1.862	1.298	1	0.765
Efficiency	0.87	0.89	0.91	0.93	0.95

Differential for distribute torque, is a mechanical transmission device capable of associating three shafts with the following rotation speed and torque relations:

$$\varphi_{in} = \frac{\varphi_{out,r}}{1+\lambda} + \frac{\lambda}{1+\lambda} \cdot \varphi_{out,l} \tag{1}$$

$$T_{out,l} = \frac{T_{in}}{1+\lambda} \tag{2}$$

$$T_{out,r} = \lambda \cdot T_{out,l} \tag{3}$$

With λ representing torque split factor, usually its value is 1.

Clutch used to transmit and cut off engine power, have a maximum transferable torque of 350 N ·m.

3 Models

In this section, model interfaces are presentation firstly, and then introduce in detail the components and system model of HEVs developed by MWorks/Modelica.

3.1 Interfaces

The development is focused on standardizing the assemblies interface definitions without enforcing a standard vehicle model architecture, so that the same assembly models can be reused in different model architectures (C. Schweiger, 2005). The PowerTrain Library based on Dymola software uses expandable bus interface to facilitate the transfer of variable informations between different components, but this also leads to excessive connections and unclear connections. On the basis of PowerTrain, this paper introduces the inner/outer mechanism, which can make the variable information of each component pass through the inner component without an additional connection bus. The principle is each component contains a component signal bus pointing to the outside. The signal bus contains an expandable connector called as bus connector that sends or receives data to the bus connector within each component of the system. All components point to the same external component, allowing implicit connection of system components. The schematic diagram is shown in Fig.2.

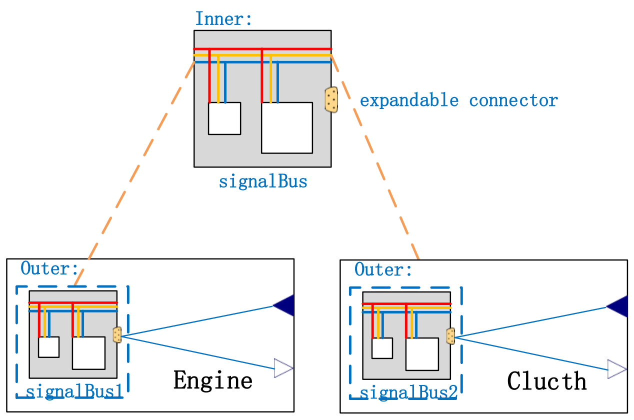


Figure 2. **Bus connector design diagram.**

3.2 Component Models

The BSG motor model consists of an internal heating model, inertia and BSG Core including both generator and motor modes. The internal heating model is used to calculate the BSG Motor temperature based on the law of conservation of energy. The BSG Core calculates the output torque and operating current of the motor, and the inertia model is used to calculate angular velocity and angular acceleration, represented in Fig.3.

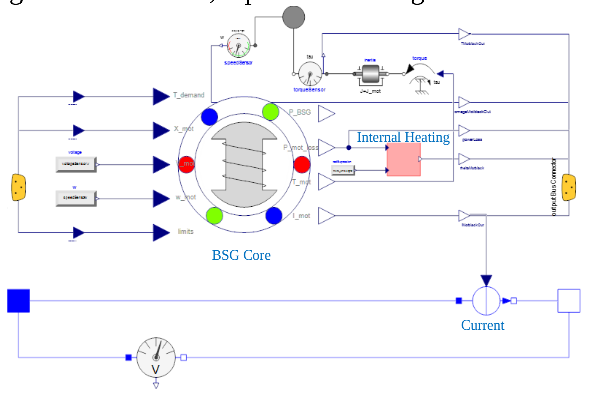


Figure 3. BSG Motor model.

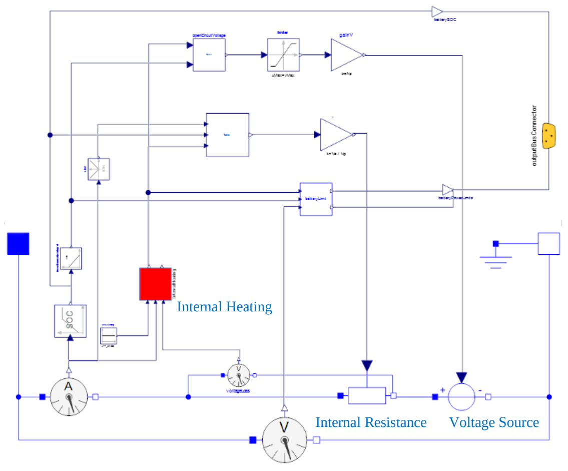


Figure 4. Battery model.

The battery model consists of a voltage source, battery internal resistance, SOC, and an internal heat module. The battery voltage is obtained by interpolation of SOC and temperature while the interpolation data obtained by experiments. The SOC characterizes the remaining capacity of the battery and is obtained by integration of current. The internal resistance of the battery is obtained by interpolation of temperature, current and SOC. The internal heating module is used to calculate battery temperature. (Fig. 4)

The hyboost controller model includes several function modules such as electronic idle, the brake energy recovery, power train torque demand management and the intelligent charging and discharging module (Crolla et al, 2012; Zou et al, 2015; Sanz et al, 2012). The electronic idle determines whether the engine should be dragged by the BSG motor to maintain the idle speed. The brake energy recovery is used to recover the energy by BSG power generation during braking. Power train torque demand management is used to calculate the demand torque of the driving vehicle. The function of the intelligent charging and discharging module is whether the BSG should be charged or discharged to make the engine work in an efficient area as much as possible according to factors such as the optimal operating point of the engine and the current battery capacity. The results of each hybrid control module are combined to calculate the torque and finally provide for the engine and the BSG, respectively.

The engine model provides a power output based on driving needs. Engine torque is interpolated from engine speed and throttle opening, taking into account the effects of dragging. The fuel consumption of the engine is obtained by interpolating the engine speed and the engine output torque, considering the effects of idle fuel consumption and engine temperature on fuel consumption. In this model, the engine external characteristic curve and the universal characteristic curve should be provided, represented in Fig.5.

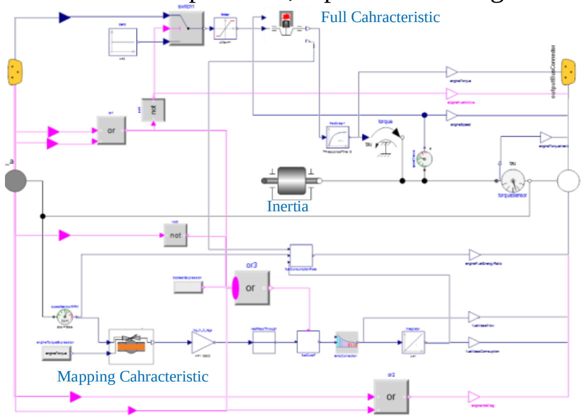


Figure 5. Engine model.

Driver model represented in Fig. 6, will determine the reference position of accelerator pedal, brake pedal and clutch pedal. Desired gear requested by comparing the vehicle speed to the desired one.

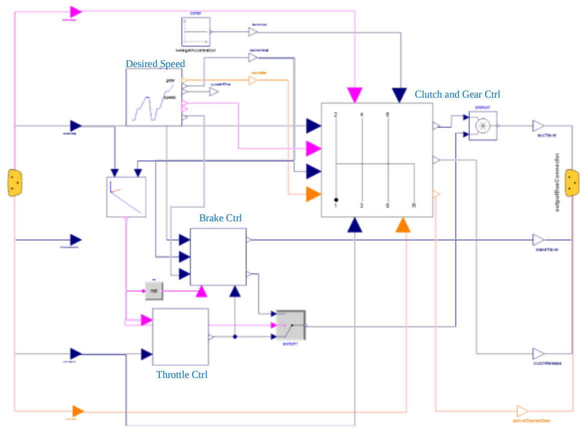


Figure 6. Driver model.

The vehicle body model is used to determine the resistant forces applied to the transmission. This is realized through a modular approach to determine different kinds of resistant forces applied to a vehicle, physical and aerodynamic drag force which provides four calculation methods. The force on the vehicle body is transmitted to the tires through the inner/outer mechanism of Modelica, represented in Fig.7.

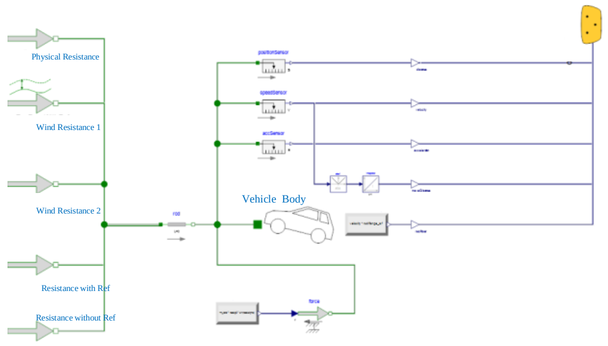


Figure 7. Vehicle body model.

The clutch, brake, final gear, transmission, and belt drive models are based on principles of the second chapter and derived from the Modelica Standard Library (version 3.2.1).

3.3 System Models

According to the model established in subsection 3.2 the system model of a light HEV constructed by building blocks and connecting lines, represented in Fig.8.

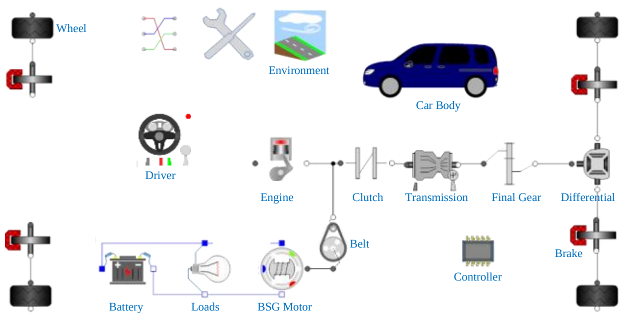


Figure 8. The light hybrid electric vehicles system model.

4 Framework Design

In order to improve the usability of the software, this paper designs a general software framework for vehicle dynamics and economic simulation based on MWorks software. The workflow is shown in Fig.9, The framework can also be easily used in other domain model libraries. The software's workflow is divided into 6 steps:

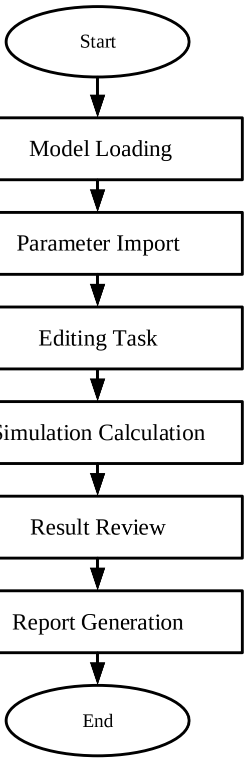
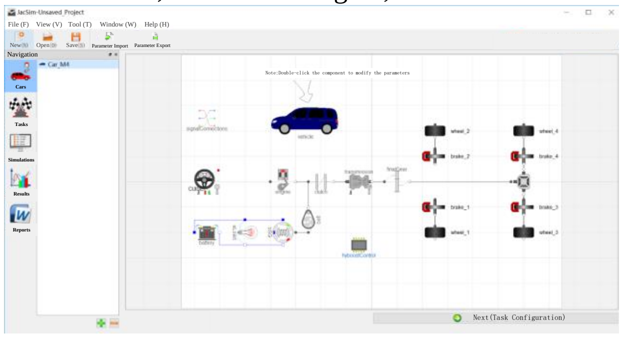


Figure 9. Flow chart of framework.

Step 1: Model loading. Select a model from the vehicle model library, for example, a manual transmission vehicle model or a Hyboost-equipped vehicle model, as shown in Fig.10;

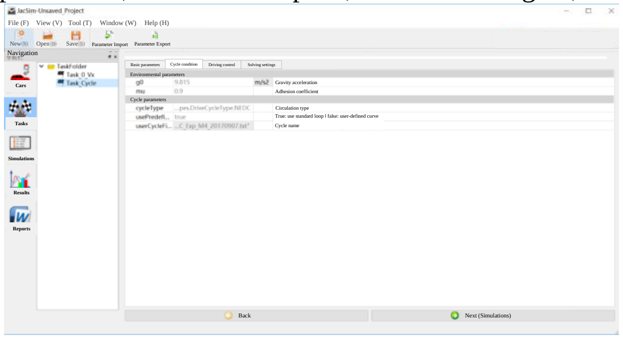


Note:Double-click the component to modify the parameters

Figure 10. Model Loading.

Step 2: Parameter import. It can be set by using the parameter dialog box or importing the whole vehicle or component parameters from Excel;

Step 3: Editing task. Select tasks that need to be calculated, such as 0-100 km/h acceleration experiment, or NEDC experiment, and then set experimental parameters, such as shift speed, as shown in Fig.11;



Environmental parameters			
g0	9.815	m/s2	Gravity acceleration
mu	0.9		Adhesion coefficient
Cycle parameters			
cycleType	...pes.DriveCycleType.NEDC		Circulation type
usePredefi...	true		True: use standard loop I false: user-defined curve
userCycleFi...	...C_Exp_M4_20170907.txt"		Cycle name

Figure 11. Editing Task.

Step 4: Simulation calculation. Three simulation methods are provided with single simulation, matrix simulation and composite simulation. Single simulation refers to the simulation of all active experimental tasks defined in the above steps. Matrix simulation means that some adjustment parameters in the model can be selected to be added to the matrix list, and then all parameters and their optional values are arranged, and then simulation is performed to all tasks for simulation. Composite simulation refers to adding components in the model to the composite simulation list, setting different batches of simulation parameters for the component, and then arranging and combining for simulation.

Step 5: Result review. The simulation results are displayed through the general and special curve windows, and the general curves are the MWorks simulation result curves, and the special curves are pre-configured curves using the xml method, such as the engine universal characteristic map.

Step 6: Report generation. The result report is automatically generated according to the configured Word template.

5 Results and Discussion

In this part, Modeling parameters is firstly explained based on the light hybrid vehicle (M4) of JAC, and then analyze the main performance indicators of the whole vehicle and parts on the acceleration condition from 0 to 100 km/h and the NEDC condition. Finally, the simulation results are compared with the experiment results. Modeling parameters are shown in Table 2.

Table 2. Parameter of HEVs system model.

Engine Specifications		***Final Specifications***	
Idle speed	700 rpm	Speed ratio	3.273
Maximum speed	6000 rpm	***Wheel Specifications***	
Idle fuel consumption	1.2 L/h	Rolling radius	0.333 m
Fuel density	0.75g/m^3	Moment of inertia	1.5 kg.m^2
Response	2 s	Adhesion	0.95

time		coefficient	
Heating Value	44000000 J/kg	*Cluth Specifications*	
Vehicle Specifications		Maximum Torque	350 N.m
Curb Weight	2082 kg	Input Inertia	0.0315 kg.m^2
Wheelbase	2.7 m	Output Inertia	0.0047 kg.m^2
Battery Specifications		*BSG Secifications*	
Rated capacity	8 Ah	Motor Inertia	0.005 kg.m^2
number of cells	15	Maximum Speed	6000 rpm

5.1 The acceleration condition from 0 to 100 km/h

The velocities of the simulation and the experiment are shown in Fig.12. It can be seen that simulation velocity tracks that of the experimental result well. The acceleration time from 0 to 100 km/h is 16.2 s.

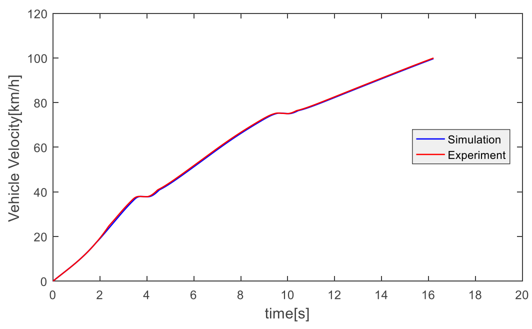


Figure 12. Results for velocities of the simulation and the experiment under condition of acceleration from 0-100 km/h.

Fig.13 and Fig.14 show variations of engine speed and engine torque respectively where simulation results are basically consistent with the experiment results. In Fig. 10, it is difficult to keep speed of model consistent with the actual engine speed in the starting phase because the throttle and the clutch need to be cooperated by controller , and in fact it is difficult to predict the driver's operation for the driver model.

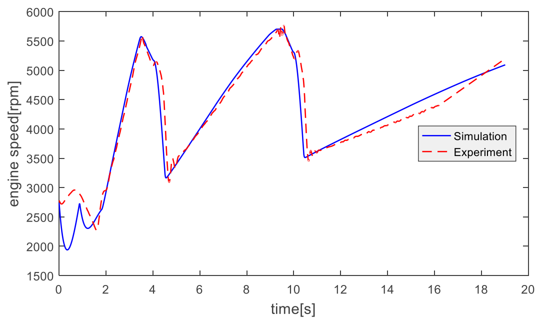


Figure 13. Results for engine speed of simulation and Experiment under condition of accelerated from 0-100 km/h.

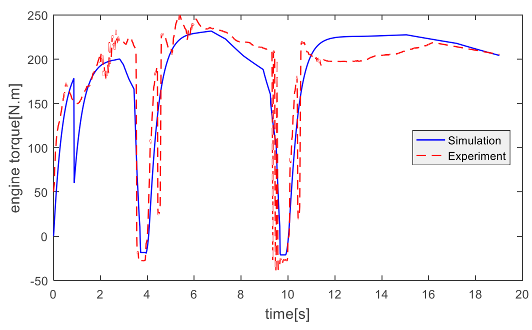


Figure 14. Results for engine torque of simulation and experiment under condition of accelerated from 0-100 km/h.

5.2 The NEDC condition

The vehicle velocities of the simulation and the experiment are shown in Fig.15. It can be seen that simulation velocities tracks that of the experiment well.

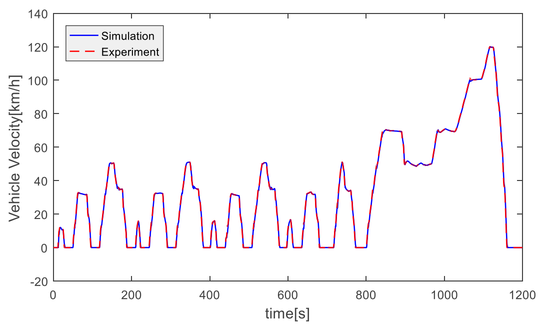


Figure 15. Results for vehicle velocity of simulation and Experiment under NEDC condition.

Fig.16 and Fig.17 show variations of engine speed and engine torque respectively where simulation results are basically consistent with the experiment results. The reason for having different results during the parking stages is that the simulation models consider the control strategy when the vehicle is idling, the fuel supply is cut off. For real vehicle experiment, while the engine is still running leading that energy is consumed due to friction during the experiment.

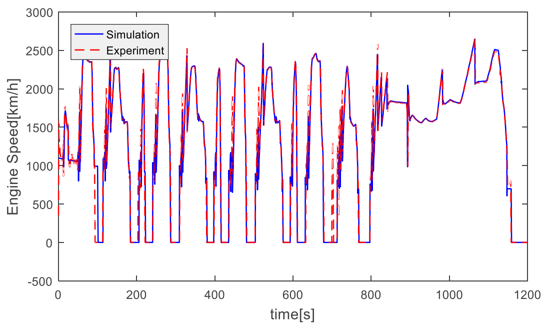


Figure 16. Results for engine speed of simulation and Experiment under NEDC condition.

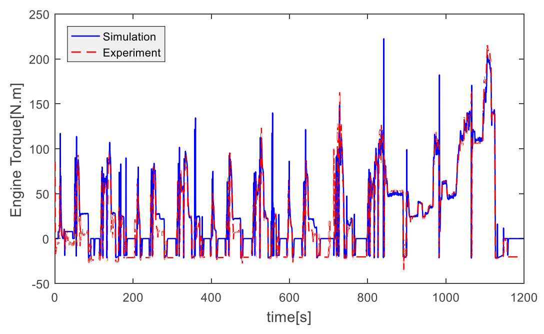


Figure 17. Results for engine torque of simulation and Experiment under NEDC condition.

Finally, Fig.18 and Fig.19 show variations of SOC and fuel consumption over the defined time period. The SOC is set as 56% initially, and the value becomes 56% at last. During the whole process of SOC is in a range from 45% to 75%, which is beneficial to battery health. The fuel consumption is 0.95 gasoline. Since the last battery's charge is equal to the initial charge, the fuel consumption per hundred kilometers is approximate 9.64 *L/ (100 km)*.

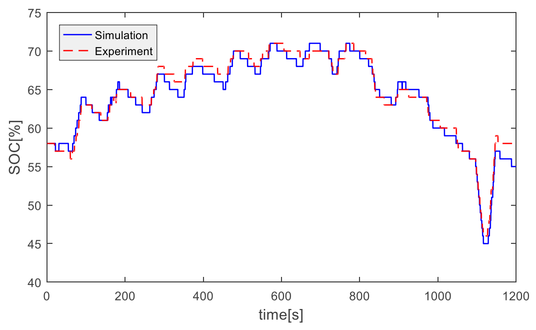


Figure 18. Results for battery SOC of simulation and Experiment under NEDC condition.

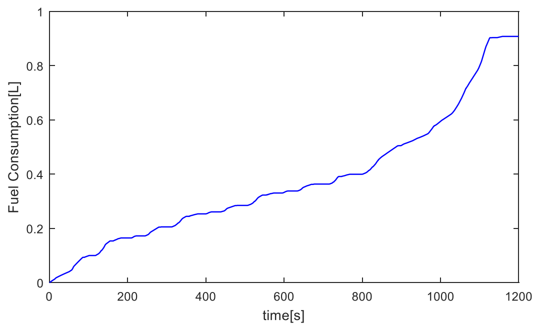


Figure 19. Results for fuel consumption of simulation and Experiment under NEDC condition.

6 Conclusions

In this paper a system model of a kind of light HEVs is developed, and an application framework is designed to simplify the simulation process. By using this application framework, the performance of the whole vehicle and components has been analyzed, and finally compared with the experimental results. The conclusions are as follows:

1) The application framework proposed can improve the usability of the models and the simulation efficiency.

2) The simulation results are basically consistent with the experimental results, which shows the effectiveness of the Modelica based multiphysical modeling method.

3) Through the analysis of the performance of the whole vehicle and key components such as BSG, battery, engine and vehicle body, it helps the whole vehicle matching optimization and the rapid design and verification of the system.

References

C. Schweiger, M. Dempsey, And M. Otter, The PowerTrain Library: New Concepts and New Fields of Application, in Proceedings of the 4th International Modelica Conference, Hamburg–Harburg, March 2005, The Modelica Association and Hamburg University of Technology, pp. 457–466.

D. Crolla, D. Cao. The impact of hybrid and electric powertrains on vehicle dynamics, control systems and energy regeneration. Veh. Syst. Dyn.20:95-109, 2012.

Jionas H. Modeling of hybrid electric vehicles in Modelica for virtual prototyping. 2nd International Modelica Conference, Proceedings, 2002, pp.247-256.

Karen L, Mehrdad Ehsani, Preyas Kamath. A Matlab-Based Modeling and Simulation Package for Electric and Hybrid Electric Vehicle Design. IEEE Transations on Vehicular Technology, 48 (6):1770-1778, 1999.

Liang L, Xiangyu W, Jian S. Fuel Consumption Optimization for Smart Hybrid Electric Vehicle during a Car-following Process. Mechanical Systems and Signal Processing, 87:17-29, 2017. doi.org/10.1016/j.ymssp.2016.03.002.

Liu W. Introduction to hybrid vehicle system modeling and control. China Machine Press, China, 2014.pp.25-28.

M Meradji, C Cecati, G Wang, D Xu. Dynamic modeling and optimal control for hybrid electric vehicle drivetrain. IEEE International Conference on Industrial Technology, 1424-1429, 2016.doi: 10.1109/ICIT.2016.7474967.

Sanz V, Urquia A, Cellier, Francois E. Modeling of hybrid control systems using the DEVSLib Modelica library. Control Engineering Practice, (1):24-34, 2012.

Xiong H Y. Zhan S, Yu L M, Zhou Y S. Modelica-based Modeling and Simulation of Electric Vehicle Brake System and Parameter Optimization. Basic Research, 2:33-37, 2016.

Zhang S J, Wu Y. Hu J N, Huang R K. Zhou Y, Bao X F. Can Euro V heavy-duty diesel engines, diesel hybrid and alternative fuel technologies mitigate NOX emissions? New evidence from on-road tests of buses in China. Applied Energy, 132:118-126, 2014. doi.org/10.1016/j.apenergy.2014.07.008.

Zou. Y, X. Hu. M S Li. Combined state of health estimation over lithium-ion battery cell cycle lifespan for electric vehicles. Power Source, 273:793-803, 2015.

Implementation of a Non-Discretized Multiphysics PEM Electrolyzer Model in Modelica®

John Webster   Carsten Bode

Institute of Engineering Thermodynamics, Hamburg University of Technology, Germany,
jcwebster@edu.uwaterloo.ca, c.bode@tuhh.de

Abstract

In this paper a multi-physics model of a proton exchange membrane electrolyzer with selectable physics submodels is developed in Modelica®. It will be included in the open-source TransiEnt Library for future studies on the efficiency of energy storage for intermittent renewable sources and the coupling of power, gas, and heat grids. The model is derived almost explicitly from a previous research paper by (Espinosa-López et al., 2018) but uses different models for cooling system power and anode/-cathode gas pressures. The model is then demonstrated in an application with wind speed records and corresponding power generation over the course of one year at a wind farm in northern Germany. It produces results similar to experimental results in other papers for use in general applications of further study.

Keywords: renewable energy, PEM electrolyzer, Power-to-Gas, TransiEnt Library, energy storage, efficiency

1 Introduction

As society becomes more and more reliant on renewable energies in efforts to reduce carbon emissions in accordance with the Paris Agreement (United Nations, 2015), the optimization of sustainable energy systems becomes increasingly important. One method of increasing energy efficiency is by coupling the energy and gas grids through Power-to-Gas (PtG) systems, which typically involve electrolyzers. To study the efficiency of using electrolyzers to harvest energy from intermittent power sources and store it in the form of hydrogen gas, a software model is developed to complement an existing proton exchange membrane (PEM) electrolyzer model. This new model increases the scope of applications of the TransiEnt Library created at the Hamburg University of Technology (Hamburg University of Technology, 2018b; Andresen et al., 2015). The existing model uses an efficiency curve to simulate the transformation of energy from power to gas. The new model is to be used in future studies on the efficiency of this technology for long term energy storage for the energy produced by renewable sources, as well as for studying performance behavior under different operating conditions, such as overload operation, or waste heat capture and reuse.

1.1 Software Used

Modelica is a declarative programming language used for mathematical modeling across many physical realms, allowing for pressure, temperature, electrochemical effects, and any other numerical relationships to be easily coupled. It is able to solve acausal systems of equations, thus one can use a graphical editor to model systems as they appear in real life, allowing for simpler development in many applications. There are many open-source libraries available for Modelica, such as TransiEnt Library and ClaRa. TransiEnt allows for the modeling of coupled energy networks with high shares of renewable energies (Hamburg University of Technology, 2018a). ClaRa is a library of power plant components which can be used to simulate transient behavior of power generating machines and technology (Hamburg University of Technology et al., 2018; Brunnemann et al., 2012). A LimPID block has been used from ClaRa to control the cooling heat flow rate. Dymola is a common tool that allows for the modeling and simulation of complex systems in Modelica, and is the primary tool used to develop the electrolyzer model (Dassault Systèmes, 2018).

Modelica variables can be declared using the keywords `inner` and `outer`, which are modifiers that allow variables to be communicated between classes. `inner` and `outer` variables are used in the new Electrolyzer Model (EM) for variables shared between physics submodels, which means that they can be interchanged and defined differently in different models while maintaining the same overall model structure.

All variables and parameters are defined in SI units unless stated otherwise.

1.2 Literature Review

PEM electrolyzers have been studied and modeled by several authors to date, using similar physical relationships each time with slight modifications (Abdin et al., 2015; Agbli et al., 2011; Awasthi et al., 2011; Espinosa-López et al., 2018; García-Valverde et al., 2012; Han et al., 2015; Lee et al., 2013; Martinson et al., 2014; Rozain and Millet, 2012; Shen et al., 2011; Zhang et al., 2012). (Olivier et al., 2017) performed a thorough study of all published research to date and compared physical expressions used in various papers. It was from this review that the electrolyzer was decidedly modeled using ODEs (ordinary differential equations) for efficiency characterization. (Olivier et al., 2017) note that ODEs are most commonly used for practical applications of electrolyzers, such as in an industrial environment, as opposed to some papers using PDEs (partial differential equations), which are more commonly used for characterization of mass transport behavior and discretized thermodynamics within each cell. PEM technology is chosen over alkaline for its faster response to load changes (Letcher, 2016).

The TransiEnt Library would benefit most greatly from having an accurate efficiency characterization model of an industrial electrolyzer, which authors have modelled using PEM physics in (Abdin et al., 2015; Agbli et al., 2011; Awasthi et al., 2011; Espinosa-López et al., 2018; García-Valverde et al., 2012; Han et al., 2015; Lee et al., 2013; Martinson et al., 2014; Rozain and Millet, 2012; Shen et al., 2011; Zhang et al., 2012). (Espinosa-López et al., 2018) include the most recent thorough review of popular papers in the characterization of electrolyzer physics and develops a detailed model of an industrial electrolyzer taking pressure, temperature, and current effects into consideration. This paper is chosen as the basis for development of the new EM since the appropriate ranges for most parameters in PEM electrolyzer modeling have been defined and used in the process of validation, explained in Section 2.2. The heat exchange model of the cooling system is omitted from their paper, and is derived in Section 2.5, for which DLR's Optimization Library (DLR, 2018) has been used to assist in the fitting of the model to correlate the new EM with the model in (Espinosa-López et al., 2018).

In the validation process, curves are digitized from (Espinosa-López et al., 2018) and used in CombiTimeTable source blocks as inputs. The EM behavior is validated in a temperature range of 20-60 °C and anode/-cathode pressures of 15-35 bar in (Espinosa-López et al., 2018).

2 PEM Electrolyzer Modeling

2.1 New Model Structure

The new electrolyzer model consists of components shown in Figure 1.

The physics are separated into submodels of replaceable voltage, temperature, pressure, and mass flow models, and shared variables are declared in the root model using the keyword `inner`. Each submodel contains `outer` variables that are shared with one or more other submodels. This forms a kind of tree structure of the electrolyzer with the main class as the root and physics models as branches/leaves (Figure 2). The main class is named `PEMElectrolyzer_L2` where L2 represents the level of detail in accordance with TransiEnt Library conventions (Brunnemann et al., 2012). The variables that each submodel must define in order for the electrolyzer model to work are noted in each submodel's base class.

In addition to the physics submodels, there is also a replaceable `Specification` record containing a set of five parameters (described in Section 2.2) experimentally determined for any electrolyzer system following the procedure outlined in (Espinosa-López et al., 2018), as well as other system specific parameters, like number of PEM cells per stack, membrane thickness, and membrane area. The replaceable submodels allow for other characteristic systems to be developed and swapped with ease. Cost and power consumption tracking models as well as fluid properties are imported from TransiEnt and TIL Media (Hamburg University of Technology, 2018b; Institut fur Thermodynamik, Technische Universitat Braunschweig and TLK-Thermo GmbH, 2018). The new EM has inputs of desired operating current, current density, power supply, or hydrogen mass output profiles that can be created by the user, with options for the user to define a temperature profile as well. Pressure can be defined through a `gasPortOut` interface component (Hamburg University of Technology, 2018b). The unknown variables of voltage, temperature, power consumption, or hydrogen output are calculated by the model according to the input profile and selected physics.

2.2 System Configuration

The electrolyzer system uses the same parameters as the 46 kW_{el} PEM Electrolyzer studied by (Espinosa-López et al., 2018) by default. This system consists of a Giner Inc. electrolyzer with 60 PEM cells in series, each with an active cell area of 290 cm^2. Areva Energy Storage assembled the electrolyzer with all of its auxiliary components, including an AC/DC converter, a water vessel, a water pump, a heat exchanger, gas separators, a gas purifying system, and multiple sensors and actuators for control, supervision and data measurement (Espinosa-López et al., 2018). Different current profiles were used to validate the operation of the EM, although the nominal operating current is 400 A across the stack electrodes (which equates to a current density of close to 1.4 A/cm^2). The characterizations of the default submodels is detailed in the following sections. Five parameters have been determined ex-

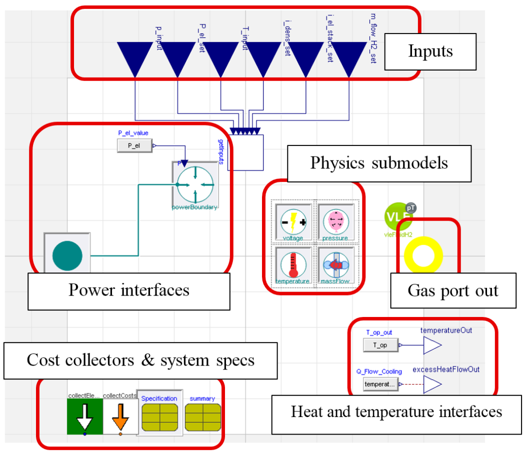


Figure 1. Graphical model of the Electrolyzer Model.

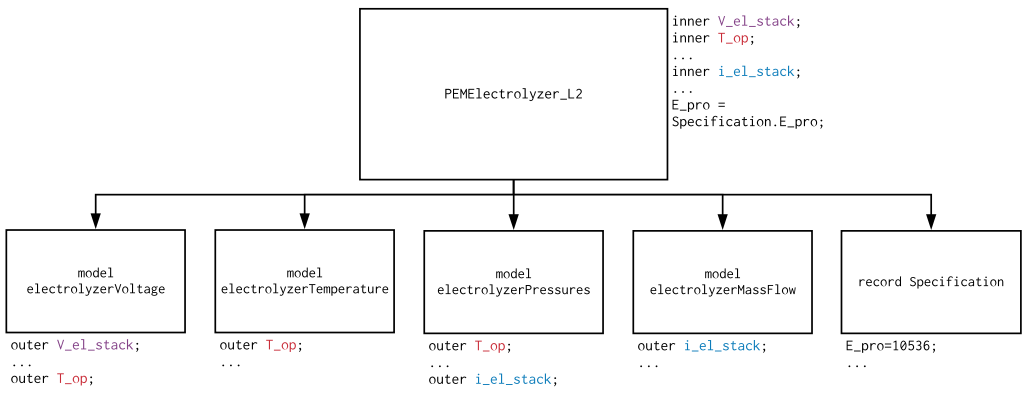


Figure 2. Modelica electrolyzer model structure showing inner/outer relationships between selected variables. Variables in the same color of text refer to the same object.

perimentally by (Espinosa-López et al., 2018): the charge transfer coefficient of the anode (α_{an}), the exchange current density on the anode at standard temperature ($i_{0,an,std}$), the activation energy required for electron transport in the anode (E_{exc}), the membrane conductivity at standard temperature ($\sigma_{mem,std}$), and the activation energy required for the proton transport in the membrane (E_{pro}).

2.3 Neglected Model Components

Some components modeled in other papers such as concentration overvoltage have been omitted from the current EM. This overvoltage, known also as diffusion overvoltage, is often neglected in industrial applications because of its minimal effects at the operating current densities. It has a much more significant effect at high pressure and higher current densities than the typical 1.4 A/cm^2 maximum in industrial applications. Mass flow across the membrane due to diffusion, which contributes to the concentration overpotential, has also not been accounted for in the current model. In industrial studies and applications, these physical effects are not predicted to have any significant impact on the resultant hydrogen produced.

2.4 Voltage Submodel

The operating PEM cell voltage (V_{cell}) can be expressed as the sum of multiple overpotentials (overvoltages) due to different material inefficiencies and natural physical effects, and multiplied by the number of cells to get the overall stack voltage. Equation (1) shows the total cell voltage as the sum of the open-circuit voltage (V_{ocv}), activation overvoltage (V_{act}), and ohmic overvoltage (V_{ohm}).

$$V_{cell} = V_{ocv} + V_{act} + V_{ohm} \tag{1}$$

The open-circuit voltage is calculated by using the reversible cell voltage (V_{rev}) in the Nernst equation. V_{ocv} is the voltage required to initiate the water electrolysis reaction under ideal conditions. V_{rev} is often expressed as Equation (2) (Espinosa-López et al., 2018) or (3) (García-Valverde et al., 2012) with T_{op} expressed in K. T_{op} represents the temperature of the water in contact with the cell membrane, and is assumed to be uniform throughout the stack for simplicity. For the EM, Equation (2) is used by default. V_{std} is 1.23 V for water electrolysis, and standard temperature T_{std} is defined as 298.15 K.

$$V_{rev}(T_{op}) = V_{std} - 0.0009(T_{op} - T_{std}) \tag{2}$$

$$\begin{aligned} V_{rev}(T_{op}) = & 1.5184 - 1.5421 \cdot 10^{-3} \cdot T_{op} \\ & + 9.523 \cdot 10^{-5} \cdot T_{op} \cdot \ln T_{op} \\ & + 9.84 \cdot 10^{-8} \cdot T_{op}^2 \end{aligned} \tag{3}$$

The open-circuit voltage is then calculated using the Nernst equation (Equation (4)), where variables pp_{H_2}, pp_{O_2}, and pp_{H_2O} refer to the partial pressures of hydrogen at cathode, oxygen at anode, and water vapor, respectively.

$$V_{ocv} = V_{rev} + \frac{R \cdot T_{op}}{2 \cdot F} \cdot \ln\left(\frac{pp_{H_2} \cdot pp_{O_2}^{0.5}}{pp_{H_2O}}\right) \tag{4}$$

F and R represent Faraday's and gas constants, respectively. The partial pressures must be converted to atm units for use in Equation (4). The partial pressures of gases are described in Section 2.5.

The activation overvoltage comes from the energy required to start the electrochemical reaction through the electrodes, but has been reduced to only consider a contribution from the anode, as in (Espinosa-López et al., 2018). α_{an} is determined experimentally to be 0.7353 (Espinosa-López et al., 2018).

$$V_{act} = \frac{R \cdot T_{op}}{2 \cdot \alpha_{an} \cdot F} \cdot \operatorname{asinh}\left(\frac{i_{dens}}{2 \cdot i_{0,an}}\right) \tag{5}$$

i_{dens} represents the current density on the PEM stack electrodes in A/m^2. The exchange current density $i_{0,an}$ is modeled from the expression used by many authors, shown in Equation (6).

$$i_{0,an} = i_{0,an,std} \cdot \exp\left(-\frac{E_{exc}}{R} \cdot \left(\frac{1}{T_{op}} - \frac{1}{T_{std}}\right)\right) \tag{6}$$

$i_{0,\text{an}}$ varies with temperature and by reference exchange current density ($i_{0,\text{an,std}} = 1.08 \cdot 10^{-4}\,\text{A/m}^2$), which has been measured on different orders of magnitude by several authors. For the sake of consistency, values from (Espinosa-López et al., 2018) have been used ($E_\text{exc} = 52994\,\text{J}$).

The ohmic overpotential is due to resistance of ion flow in the cell components. It can be expressed as simply as Ohm's law, using the inverse of membrane conductivity to determine the resistance of the cell, as in Equation (7).

$$V_\text{ohm} = R_\text{mem} \cdot i_\text{dens} \tag{7}$$

The membrane resistance R_mem ($\Omega\,\text{m}^2$) is calculated from the membrane conductivity σ_mem (S/m) and membrane thickness δ_mem (m) using Equation (8), and in conjunction with the current density on the electrodes, the voltage can be calculated.

$$R_\text{mem} = \frac{1}{\sigma_\text{mem}} \delta_\text{mem} \tag{8}$$

The default membrane has specifications of a Nafion 117 membrane, with a thickness (δ_mem) of $178 \cdot 10^{-6}\,\text{m}$. The membrane conductivity expression is selectable in the EM as Equation (9) (Espinosa-López et al., 2018), Equation (10) (Biaku et al., 2008), or Equation (11) (Awasthi et al., 2011; Han et al., 2015; Zhang et al., 2012) and is always expressed in S/m.

$$\sigma_\text{mem} = \sigma_\text{mem,std} \cdot \exp\left(\frac{E_\text{pro}}{R} \cdot \left(\frac{1}{T_\text{op}} - \frac{1}{T_\text{std}}\right)\right) \tag{9}$$

$$\begin{aligned}\sigma_\text{mem} =& 4.8 \cdot 10^{-4} + 8.15 \cdot 10^{-6} \cdot T_\text{op} \\ &+ 5.12 \cdot 10^{-9} \cdot T_\text{op}^2\end{aligned} \tag{10}$$

$$\begin{aligned}\sigma_\text{mem} =& (0.005114 \cdot \lambda_\text{mem} - 0.00326) \\ &\cdot \exp\left(1268 \cdot \left(\frac{1}{303} - \frac{1}{T_\text{op}}\right)\right)\end{aligned} \tag{11}$$

λ_mem is the degree of humidity of the membrane, which is equal to 14 by default. The default model is Equation (8), with $E_\text{pro} = 10536\,\text{J/mol}$ and $\sigma_\text{mem,std} = 10.31\,\text{S/m}$.

2.5 Temperature Submodel

Similar to many papers, (Espinosa-López et al., 2018) implement a lumped thermal capacitance model, so that the temperature of the entire electrolyzer system can be simplified in one equation, as shown in Equation (12).

$$\begin{aligned}C_\text{th} \frac{\mathrm{d}T_\text{op}}{\mathrm{d}t} =& \dot{Q}_\text{electrolysis,heat} + \dot{W}_\text{pump,loss} - \dot{Q}_\text{cooling} \\ &- \dot{Q}_\text{loss} - \sum_j \dot{n}_j \cdot \Delta h_j\end{aligned} \tag{12}$$

The thermal capacity of the electrolyzer stack, C_th, is determined experimentally by (Espinosa-López et al., 2018) to be $162116\,\text{J/K}$. The other terms in the equation (all positive, expressed in W) are $\dot{Q}_\text{electrolysis,heat}$ for the heat generated by the electrolysis reaction, $\dot{W}_\text{pump,loss}$ for the work contributed by the pump in the water supply network, $\dot{Q}_\text{cooling}$ for the heat removed by a cooling system in a separate pipe network, $\dot{Q}_\text{loss}$ for the heat lost to ambient environment by convection, and finally, a term for enthalpy lost with H_2 and O_2 products leaving the system.

$\dot{Q}_\text{electrolysis,heat}$ is generated when operating the electrolyzer at voltages above the thermoneutral voltage. Thus, it can be expressed as Equation (13),

$$\dot{Q}_\text{electrolysis,heat} = (V_\text{cell} - V_\text{tn}) \cdot I \cdot n_\text{cells} \tag{13}$$

where $V_\text{tn} = 1.48\,\text{V}$ is the thermoneutral voltage of water electrolysis used in (Espinosa-López et al., 2018). I is the current across the stack electrodes, and n_cells is the number of cells in the electrolyzer stack.

The work that the pump contributes to the system is considered to be proportional to the electrical energy consumed, which is simplified from the implementation in (Espinosa-López et al., 2018), using the rated electric power consumption of the pump, $\dot{W}_\text{pump,elec} = 1100\,\text{W}$, and the pump efficiency, $\eta_\text{motor,elec} = 0.75$ (Equation (14)).

$$\dot{W}_\text{pump,loss} = \dot{W}_\text{pump,elec} \cdot \eta_\text{motor,elec} \tag{14}$$

Since $\dot{Q}_\text{cooling}$ is not published in their paper, a new expression is derived for it by using a similar procedure to that which (Espinosa-López et al., 2018) describes. A simulation is run with the input current set to 400 A and T_op is allowed to rise until it reaches a nominal operating temperature of 328.95 K, at which point the derivative of operating temperature is set to 0, such that $\dot{Q}_\text{cooling}$ attains the value of the excess heat at that operating current. The value of $\dot{Q}_\text{cooling}$ is seen to be 6911 W at 30 bar (or 6039 W at 5.86 bar; the cooling power required varies with pressure), which is considered to be the maximum cooling power of the heat exchanger in order to match the model in (Espinosa-López et al., 2018). A LimPID block (Hamburg University of Technology et al., 2018) is implemented within the `Temperature1` thermal model, acting as a PI controller, activated only when T_op surpasses $T_\text{set} = 318.95\,\text{K}$, and only cools (does not heat the system). `use_activateInput` becomes true when $T_\text{op} > T_\text{set}$. To choose appropriate tuner values for the LimPID k_p and τ_i, DLR's Optimization Library (DLR, 2018) is used to compare the temperature output of the EM with constant 400 A current driven temperature output in (Espinosa-López et al., 2018), and calculate k_p and τ_i to minimize the deviation between the models. The optimized values are $\tau_\text{i} = 7.741 \cdot 10^{-4}$ and $k_\text{p} = 500$. If the electrolyzer operates at currents or pressures greater than 400 A or 30 bar, respectively, the operating temperature of the system will increase past 60 °C because of the cooling power limit.

$\dot{Q}_\text{loss}$ is calculated using the convective cooling relationship in Equation (15), and the thermal resistivity R_th is taken from experimental results in (Espinosa-López et al., 2018) as 0.0668 K/W. The ambient temperature T_amb is set to 296 K by default to match the starting temperature in experiments from (Espinosa-López et al., 2018).

$$\dot{Q}_{\text{loss}} = \frac{1}{R_{\text{th}}} \cdot \left(T_{\text{op}} - T_{\text{amb}}\right) \tag{15}$$

The final component of the energy balance equation comes from enthalpy lost with the products leaving the system, as calculated using two empirical equations for molar heat capacities from (Cengel and Boles, 2008). The expressions use coefficients of molar specific heat (J/(mol K)) for H_2 and O_2 (c_{p,m,H_2} and c_{p,m,O_2}, from Equations (16) and (17), respectively) as a function of T_{op}, and are summed as Equation (18) (Espinosa-López et al., 2018). The moles generated/consumed of each fluid ($\dot{n}_{H_2O}, \dot{n}_{H_2}, \dot{n}_{O_2}$) are explained in the mass flow submodel.

$$\begin{aligned} c_{p,\text{m},H_2} =&(29.11 - 1.92 \cdot 10^{-3} \cdot T_{\text{op}} \\ &+ 4.0 \cdot 10^{-6} \cdot T_{\text{op}}^2 - 8.7 \cdot 10^{-10} \cdot T_{\text{op}}^3) \quad (16)\\ c_{p,\text{m},O_2} =&(25.48 + 1.52 \cdot 10^{-2} \cdot T_{\text{op}} \\ &- 7.16 \cdot 10^{-6} \cdot T_{\text{op}}^2 + 1.31 \cdot 10^{-9} \cdot T_{\text{op}}^3) \quad (17)\\ \sum_j \dot{n}_j \cdot \Delta h_j =&\dot{n}_{H_2} \cdot c_{p,\text{m},H_2} \cdot \left(T_{\text{op}} - T_{\text{amb}}\right) \\ &+ \dot{n}_{O_2} \cdot c_{p,\text{m},O_2} \cdot \left(T_{\text{op}} - T_{\text{amb}}\right) \quad (18) \end{aligned}$$

2.6 Pressure Submodel

The pressure exerted by water vapour (pp_{H_2O}) in the cell is calculated in atm using Equation (19), from (Espinosa-López et al., 2018). The partial pressures of H_2 and O_2 gases are calculated from Dalton's law of partial pressures which assumes ideal gas behavior in Equations (20) and (21), after pp_{H_2O}, p_{cat} and p_{an} are converted to Pa.

$$\begin{aligned} pp_{H_2O} =&6.1078 \cdot 10^{-3} \\ &\cdot \exp\left(17.2694 \cdot \frac{T_{\text{op}} - 273.15}{T_{\text{op}} - 34.85}\right) \quad (19)\\ pp_{H_2} =&p_{\text{cat}} - pp_{H_2O} \quad (20)\\ pp_{O_2} =&p_{\text{an}} - pp_{H_2O} \quad (21) \end{aligned}$$

p_{cat} and p_{an} are the pressures of the hydrogen and oxygen storage tanks, respectively. A 1 bar negative pressure gradient from cathode to anode side is used, which (Espinosa-López et al., 2018) explain is to reduce the mechanical stress on the membrane.

2.7 Mass Flow Submodel

The molar production rates of hydrogen and oxygen (mol/s) can be defined using Faraday's Law, as in Equations (22) and (23). Water molar flow is calculated as well (Equation (24)).

$$\dot{n}_{H_2} = \frac{n_{\text{cells}} \cdot I}{2 \cdot F} \cdot \eta_{\text{f}} \tag{22}$$

$$\dot{n}_{O_2} = \frac{n_{\text{cells}} \cdot I}{4 \cdot F} \cdot \eta_{\text{f}} \tag{23}$$

$$\dot{n}_{H_2O} = \frac{n_{\text{cells}} \cdot I}{2 \cdot F} \cdot \eta_{\text{f}} \tag{24}$$

η_{f} is the Faraday efficiency of reaction, which is equal to 1 as in (Espinosa-López et al., 2018).

3 Validation

To validate the electrolyzer model, figures are taken from (Espinosa-López et al., 2018), digitized and plotted alongside the EM output plots. The voltage and temperature curves are compared first with a constant current input of 400 A in Figure 3. The EM curves match closely to the experimental results in (Espinosa-López et al., 2018).

Two more validation models are created, showing dynamic current profiles from a solar photovoltaic (PV) array starting up (Figure 4) and from 7AM to 9PM on a cloudy day (Figure 5). The current profiles, voltage and temperature curves from experimental results in (Espinosa-López et al., 2018) are shown alongside the output of the EM.

In Figures 3 and 4, it is observed that the temperature of the EM rises slightly more quickly than in (Espinosa-López et al., 2018), but that the voltage and temperature do not deviate by much when the current fluctuates below 400 A. For the cloudy day PV current profile, the normalized integrated squared deviation is calculated between the resultant voltage and temperature curves and with those from (Espinosa-López et al., 2018). In the central region of operation, after start up and before shutoff, the resultant temperature and voltage curves have deviations of 2.66 °C and 0.853 V, respectively. This deviation rises outside a simulation time of 1750 s to 28 550 s because of differently implemented voltage values when the electrolyzer is powered off. In the EM, the default voltage when current is 0 A is 0 V. The deviations between the EM and (Espinosa-López et al., 2018) are due to a few factors. One source of error is due to the digitization of the original curves, where the accuracy relies on the user selecting data points manually. A second source of error is due to the interpolation of the Modelica CombiTimeTable blocks used to generate the input and reference output curves. It is also

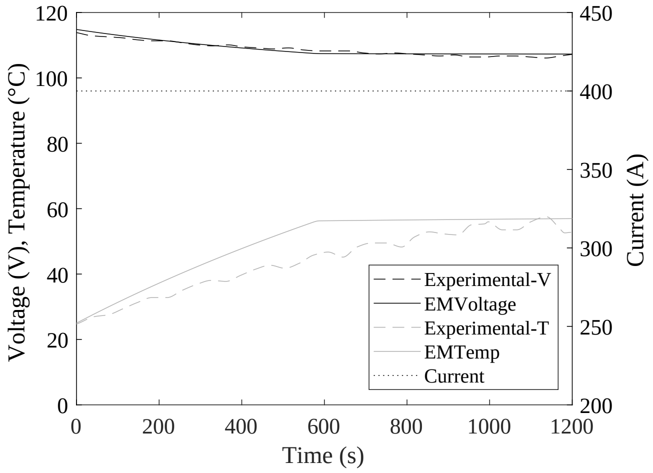


Figure 3. Temperature and voltage models compared for 400 A constant input current.

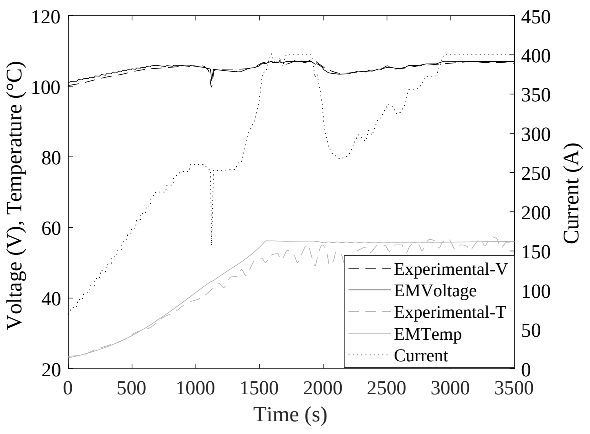


Figure 4. Temperature and voltage models compared for PV startup current profile.

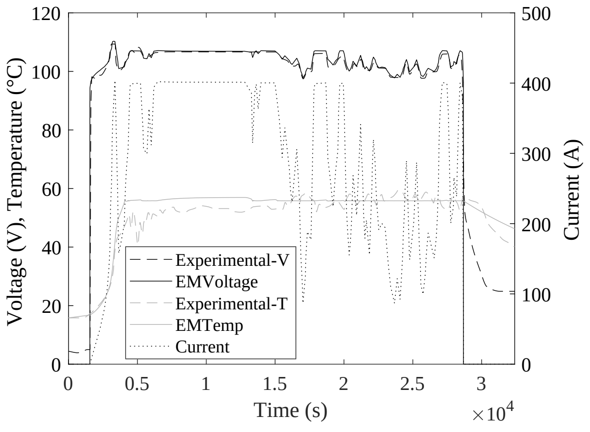


Figure 5. Temperature and voltage models compared for cloudy day PV current profile, starting at 7AM.

observed that the temperature models behave differently around the equilibrium temperature, which is due to the different implementations of the PID cooling system and tank storage pressure controls for the electrolyzer. Despite this, the deviation becomes visibly smaller as the operating current fluctuates below the nominal 400 A. It is observed that the operating pressure has a significant effect on the stack voltage, and the deviations between voltages can be greatly reduced if operating cathode pressure is reduced from 30 bar, although the default pressure is kept at 30 bar to stay consistent with (Espinosa-López et al., 2018). Espinosa-López has explained that the pressure model would differ for each electrolyzer system, and that they have implemented a pressure model that changes pressure relative to the amount of hydrogen produced (Espinosa-López, 2018), whereas in the EM the anode and cathode side pressures are static.

4 Applications

An experiment is conducted to inspire further research coupling electrolyzers with intermittent renewable power sources. A current profile is generated from a Vestas112-3.0MW wind turbine power curve with wind speed records from Wrohm-Osterrade Wind Farm in the north of Germany from 2015 (Deutscher Wetterdienst, 2018). In this application, all of the wind power is used to produce hydrogen instead of being channelled into the grid. Given the wind speeds in m/s at various heights, the "driving wind speed," defined as the average wind speed across the diameter of a rotor in (Brown, 2012), is calculated first. A linear interpolation is then used to map the driving wind speed to a power output of a single turbine of the seven at Wrohm-Osterrade using the power curve given by (Kopp, 2018), which is then used as the power input for the one year operation of 66 electrolyzers in the Areva configuration. The wind speeds, resulting power output (generated by one turbine) and hydrogen output from 66 theoretical electrolyzers connected to the turbine are shown in Figure 6. The average efficiency calculated using the net calorific value of the hydrogen produced is 75.3% over the course of the year, while the efficiency using the gross calorific value is 89.0%. A single electrolyzer produces approximately 4125 kg in a single year.

A total of 1.906 t of hydrogen could be theoretically produced if 66 electrolyzers were installed for each of the seven wind turbines at Wrohm-Osterrade, which is enough hydrogen to fill over 400 000 tanks of the Toyota Mirai sedan (Toyota, 2018).

5 Conclusions and Outlook

A new electrolyzer model has been developed with detailed physics which can be developed and substituted with ease, accounting for physical effects of temperature, pressure, operating current and electrochemistry. In addition, multiple inputs have been created so that the user is able to control new parameters of the electrolyzer operation, including operating temperature and current. The model is in good agreement with (Espinosa-López et al., 2018) experimental and simulated data and is thus suitable for practical use. The model has been used with data from the Wrohm-Osterrade Wind Farm to calculate a theoretical quantity of hydrogen produceable by a wind farm over the course of one year.

The future should focus on obtaining parameters specific to more electrolyzer systems in use today. Using the procedures outlined in (Espinosa-López et al., 2018), any electrolyzer system can be characterized in a Modelica record and imported as a `Specification` in the EM for an accurate simulation of the system's behavior at different temperatures, pressures, and powers. Further studies can use the EM to model overload behavior of electrolyzers for use during peak demand for nominal electric power, and simultaneously examine the excess heat flows generated in the system to increase the overall efficiency and profitability of the system.

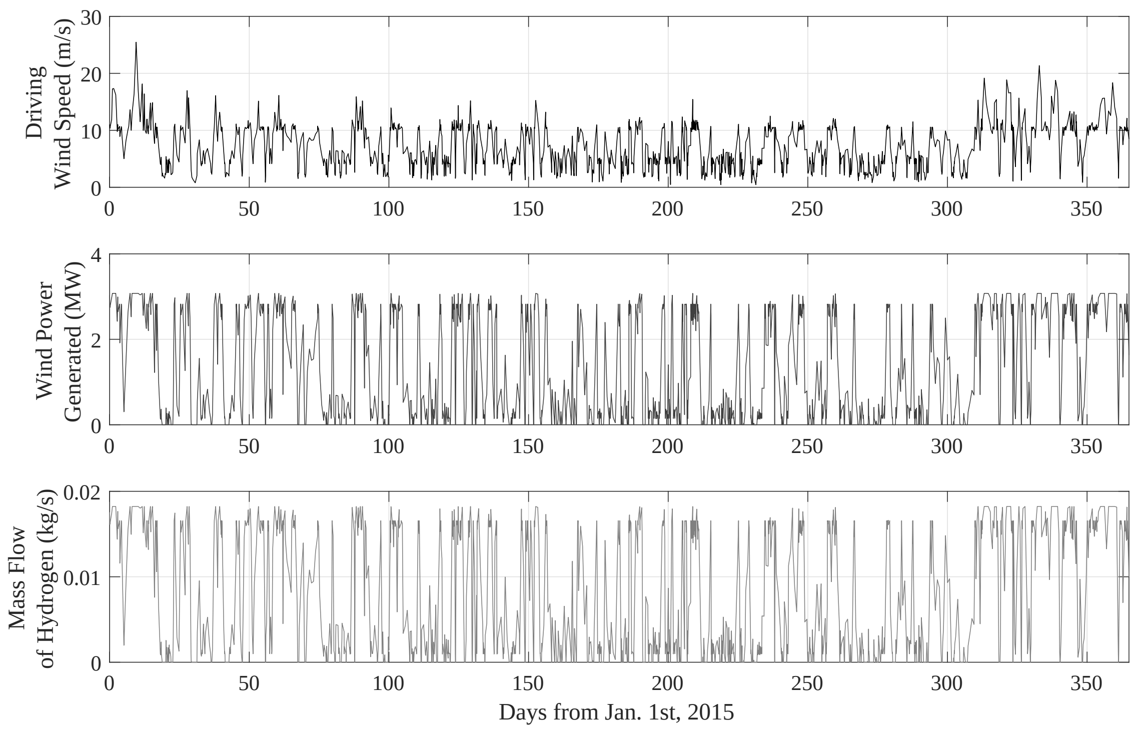


Figure 6. Driving wind speed and corresponding power and hydrogen mass flow output over the course of one year, from one Vestas112-3.0MW wind turbine powering 66 electrolyzers.

Acknowledgements

The authors would like to acknowledge the German Federal Ministry for Economic Affairs and Energy for its funding for the project "ResilientEE - Resilience of integrated energy networks with a high share of renewable energies" (03ET4048).

References

Z. Abdin, E. MacA. Gray, and C.J. Webb. Modelling and simulation of a proton exchange membrane (PEM) electrolyzer cell. *International Journal of Hydrogen Energy*, 40(39):13243–13257, 2015. doi:10.1016/j.ijhydene.2015.07.129.

K. S. Agbli, I. Doumbia, D. Hissel, M. C. Péra, O. Rallières, and C. Turpin. Multiphysics simulation of a PEM electrolyser: Energetic macroscopic representation approach. *International Journal of Hydrogen Energy*, 36(2):1382–1398, 2011. doi:10.1016/j.ijhydene.2010.10.069.

L. Andresen, P. Dubucq, R. Peniche, G. Ackermann, A. Kather, and G. Schmitz. Status of the transient library: Transient simulation of coupled energy networks with high share of renewable energy. *Proceedings of the 11th International Modelica Conference*, pages 695–705, 2015. doi:10.3384/ecp15118695.

A. Awasthi, S. Basu, and Keith Scott. Dynamic modeling and simulation of a proton exchange membrane electrolyzer for hydrogen production. *International Journal of Hydrogen Energy*, 36(22):14779–14786, 2011. doi:10.1016/j.ijhydene.2011.03.045.

C. Y. Biaku, N. V. Dale, M. D. Mann, H. Salehfar, A. J. Peters, and T. Han. A semiempirical study of the temperature dependence of the anode charge transfer coefficient of a 6 kw PEM electrolyzer. *International Journal of Hydrogen Energy*, 33: 4247–4254, 2008. doi:10.1016/j.ijhydene.2008.06.006.

Cameron Brown. Fast verification of wind turbine power curves: Summary of project results. Technical report, Technical University of Denmark, Kongens Lyngby, 2012.

Johannes Brunnemann, Friedrich Gottelt, Kai Wellner, Ala Renz, André Thuering, Volker Roeder, Christoff Hasenbein, Christian Schulze, Gerhard Schmitz, and Joerg Eiden. Status of ClaRaCCS: Modelling and simulation of coal-fired power plants with CO2 capture. *Proceedings of the 9th International Modelica Conference*, pages 609–618, 2012. doi:10.3384/ecp12076609.

Y. A. Cengel and M. A. Boles. *Thermodynamics: an engineering approach, sixth ed.* Sea, 2008. ISBN 9789814595292.

Dassault Systèmes. Dymola – Dassault Systèmes, 2018. URL `https://www.3ds.com/products-services/catia/products/dymola/`.

Deutscher Wetterdienst. Pamore – Abruf archivierter Daten der Vorhersagemodelle, 2018. URL `https://www.dwd.de/DE/leistungen/pamore/pamore.html`.

DLR. Commercial Modelica Libraries developed by DLR-SR, 2018. URL `https://www.dlr.de/rm/en/desktopdefault.aspx/tabid-9281/`.

Manuel Espinosa-López. Personal communication. November 2018.

Manuel Espinosa-López, Philippe Baucour, Serge Besse, Christophe Darras, Raynal Glises, André Rakotondrainibe Philippe Poggi, and Pierre Serre-Combe. Modelling and experimental validation of a 46 kw PEM high pressure water electrolyser. *Renewable Energy*, 119:160–173, 2018. doi:10.1016/J.RENENE.2017.11.081.

R. García-Valverde, N. Espinosa, and A. Urbina. Simple PEM water electrolyzer model and experimental validation. *International Journal of Hydrogen Energy*, 37(2):1927–1938, 2012. doi:10.1016/j.ijhydene.2011.09.027.

Hamburg University of Technology. ResiliEntEE – Welcome, 2018a. URL `https://www.tuhh.de/transient-ee/en/index.html`.

Hamburg University of Technology. TransiEnt Library, 2018b. URL `https://www.tuhh.de/transient-ee/en/`.

Hamburg University of Technology, TLK-Thermo GmbH, and XRG-Simulation GmbH. ClaRa 1.3.0, 2018. URL `https://www.claralib.com/`.

Bo Han, Jinkge Mo, Stuart M. Steen III, and Feng-Yuan Zhang. Electrochemical performance modeling of a proton exchange membrane electrolyzer cell for hydrogen energy. *International Journal of Hydrogen Energy*, 40(22):7006–7016, 2015. doi:10.1016/j.ijhydene.2015.03.164.

Institut fur Thermodynamik, Technische Universitat Braunschweig and TLK-Thermo GmbH. TILMedia 1.3.0 ClaRa, 2018. URL `https://www.tlk-thermo.com/index.php/en/software-products/tilmedia-suite`.

Stefan Kopp. Leistungskurven von modernen Binnenland-Windenergieanlagen, 2018. URL `http://www.windenergie-im-binnenland.de/powercurve.php`.

Bonghwan Lee, Hyung-Man Kim, and Kiwon Park. Dynamic simulation of PEM water electrolysis and comparison with experiments. *International Journal of Electrochemical Science*, 8:235–248, 2013.

Trevor Letcher. *Storing Energy, 1st Edition, with Special Reference to Renewable Energy Sources*. Elsevier, 2016. ISBN 9780128034491.

C. A. Martinson, D. Bessarabov, G. van Schoor, and K. R. Uren. Characterisation of a PEM electrolyzer using the current interrupt method. *International Journal of Hydrogen Energy*, 39(36):20865–20878, 2014. doi:10.1016/j.ijhydene.2014.09.153.

Pierre Olivier, Pr. Belkacem Bouamama, and Cyril Bourasseau. Low-temperature electrolysis system modelling: A review. *Renewable and Sustainable Energy Reviews*, 78:280–300, 2017. doi:10.1016/j.rser.2017.03.099.

C. Rozain and P. Millet. Electrochemical characterization of polymer electrolyte membrane water electrolysis cells. *Electrochimica Acta*, 131:160–167, 2012. doi:10.1016/j.electacta.2014.01.099.

Muzhong Shen, Nick Bennett, Yulong Ding, and Keith Scott. A concise model for evaluating water electrolysis. *International Journal of Hydrogen Energy*, 36(22):14335–14341, 2011. doi:10.1016/j.ijhydene.2010.12.029.

Toyota. 2017 Mirai Product Information, 2018. URL `https://ssl.toyota.com/mirai/assets/core/Docs/Mirai%20Specs.pdf`.

United Nations. Paris Agreement, 2015.

Houcheng Zhang, Jincan Chen, Guoxing Lin, and Shanhe Su. Efficiency calculation and configuration design of a PEM electrolyzer system for hydrogen production. *International Journal of Electrochemical Science*, 7:4143–4157, 2012.

Translating Simulink Models to Modelica using the Nsp Platform

Jean-Philippe Chancelier[1] Sébastien Furic[2] Pierre Weis[2]

[1]Université Paris-Est, CERMICS (ENPC), 77455 Marne-la-Vallée 2, France,
jean-philippe.chancelier@enpc.fr
[2] Inria Paris, 2 rue Simone Iff, 75589 Paris, France & Université Paris-Est, CERMICS (ENPC), 77455 Marne-la-Vallée 2, France {sebastien.furic,pierre.weis}@inria.fr

Abstract

We present a new Simulink (Simulink) to Modelica (Modelica) translation chain embedded into Nsp. Translated models can be edited (original Simulink diagrams are preserved through translation) and simulated. This translation chain makes use of the Simport tool, originally designed to translate Simulink models to Scicos models, and also relies on Modelicac, i.e. Scicos' Modelica companion compiler.

Using some examples, we demonstrate the effectiveness of the translation process and detail some technical aspects of it. This new Nsp feature extends Nsp's simulation capabilities and makes it a reference platform for users looking for means to simulate Simulink models within a Modelica framework. Resulting Modelica code can even be exported to other Modelica compatible tools.
Keywords: Nsp; Simulink; Modelica

1 Introduction

Nsp is a Matlab-like numerical environment which can run the Scicos modeling environment, a Simulink-like block diagram editor and simulator.

From 2003 to 2008, in the course of funded projects SimPA and SimPA2, a Modelica compiler named Modelicac has been developed allowing Scicos to handle genuine Modelica models. This integration of Modelica within Scicos has been the subject of several papers published at the Modelica conference (Nikoukhah and Najafi, 2008; Nikoukhah and Furic, 2009). The purpose of using Modelica is to serve as a high-level description language to extend Scicos expressiveness: Modelica allows users to compose "acausal" models where the original environment forced users to describe their models as block diagrams.

In this paper we focus on a new application of Modelica within Scicos under Nsp, that is as a target language for Simulink model translation.

Several Simulink to Modelica translation tools have already been proposed in the past, we mention in particular Mike Dempsey's Simelica and AdvancedBlocks library (Dempsey, 2003) and Dirk Reusch's Coselica initiative (Reusch). AdvancedBlocks was a fairly complete library of Modelica blocks which allowed users to use Modelica blocks as a one-to-one replacement for Simulink blocks. Up to our knowledge this work remains the most advanced effort in that direction. It is however no longer maintained. In the Scicos software environment, the Coselica library offers signal models to allow users to better exploit Modelica from within Scicos by proposing a large set of Modelica submodels in the same spirit as the standard Modelica library (MSL). Many Simulink-like blocks are also available in Coselica.

The approach presented here differs from Mike Dempsey's and Dirk Reusch's approaches in that translation of original blocks is not attempted on a one-to-one basis. Instead, we use a two steps translation process: the first step translates the Simulink model into an equivalent Scicos native model; the second step translates the Scicos native model to Modelica code.

To handle the initial Simulink to Scicos translation, we use an external tool named Simport: it translates a Simulink model by translating each block in the original Simulink model with one or several blocks of the Scicos native block library, so that original semantics is preserved with a high degree of confidence.

For the second step, we developed a set of Nsp special purpose compilation routines to translate Scicos blocks to genuine Modelica blocks. When a Scicos block has a direct Coselica equivalent, the compilation routine simply emits thre corresponding Coselica block; when there is no Coselica equivalent, the compilation routine generates an entirely new block containing ad-hoc Modelica code to handle the Scicos block behavior. Using this compile-time Modelica code generation and the two steps translation process was the key ideas to fill the huge semantic gap between Simulink and Modelica.

We give in this paper a detailed description of this new translation chain hosted by the Nsp environment.

2 Involved Tools

As mentioned above, the translation chain relies on a combination of several tools. We give hereafter a short description of each of them.

2.1 Nsp, a Programming Environment for Numerical Applications

Nsp (Nsp) is a mature Matlab-like Scientific Software Package developed under the GPL license. Nsp features a high-level, safe imperative programming language with automatic memory management. This language can be used interactively, giving users an easy access to efficient numerical routines; It can also be used as a more conventional programming language to extend Nsp's capabilities.

Nsp contains internally a class system with simple inheritance and interface implementation. When used as an interactive computing environment, it comes with online help facilities and an easy access to GUI facilities and graphics.

A large set of libraries are available and it is moreover easy to implement new functionalities. External libraries can also be used: this requires writing some wrapper code (also called *interface*) to live in harmony with Nsp's internal state. The interface mechanism can be either static or dynamic. By using dynamic functionalities one is able to build toolboxes.

Nsp shares many traits with other Matlab-like Scientific Softwares such as Matlab, Octave, ScilabGtk (ScilabGtk; Campbell et al., 2006), and also with scripting languages such as Python.

The main toolbox used in this work is Scicos that we describe now.

2.2 Scicos, a Block Diagram Modeler and Simulator

Scicos (Scicos) is a graphical dynamical system modeler and simulator originally developed in the Metalau project at INRIA, Paris-Rocquencourt center. With Scicos, users can create block diagrams to model and simulate the dynamics of hybrid dynamical systems and compile models into executable code. Scicos is used for signal processing, systems control, queuing systems, and to study physical and biological systems. Extensions allow generation of component-based modeling of electrical and hydraulic circuits using the Modelica language.

We consider in this paper the Scicos/Nsp version of Scicos maintained and developed at ENPC. Scicos/Nsp is a Nsp toolbox and runs in the Nsp environment. Having access to Nsp functions when designing simulation models is of great importance.

Scicos users often needs to use Nsp functions such as those dedicated to filter design for signal processing or controller design in the construction of simulation models. Nsp's programming language can be used for batch processing of multiple simulation tasks, and more generally, models designed by Scicos can be used as functions in Nsp. Nsp's graphical facilities can be used for post processing simulation results. But the integration of Scicos and Nsp goes beyond that. The Scicos editor is entirely written in Nsp's language. This provides many advantages and was in particular of tremendous importance in the current work, indeed: Scicos model data structure is a Nsp structure and thus Scicos models can be programmatically manipulated and build using Nsp scripts. We use this facility in two ways. First to obtain Scicos models from Simulink models, using the fact that the Simport converter produces a Nsp script whose execution in Nsp produces a Scicos model data structure. Second, using Nsp scripts we are able to convert, in a Scicos model data structure, some Scicos blocks to Modelica blocks.

In the conversion process from Simulink to Modelica, the scicos compiler/scheduler also plays a key role. It infers dimensions and types used in the Modelica blocs. This is quite an exciting feature since it gives the possibility to have Modelica blocks for which the associated Modelica model is not a fixed Modelica class but a specific one adapted to specific dimensions and types generated on the fly.

2.3 Modelicac, a Simple Yet Useful Modelica Compiler

Development of Modelicac started in 2003 as a joint work between Inria and TNI-Valiosys (now Dassault Systèmes) in the course of the SimPA (SIMulation pour le Procédé et l'Automatique) french funded project. The goal was to make Scicos compatible with a significant subset of the Modelica language in order for users to be able to describe complex hybrid models without having to resort to low-level block diagram descriptions. Indeed, building a block diagram from a physical model requires 1)performing a *complete* analysis of physical phenomena into play (to determine which elementary blocks to use in the diagram), and 2)determining how data flows between blocks (to connect elementary blocks together). On the other hand, Modelica tools considerably ease physical model construction by automatically analysing the overall structure of physical models described in a much more user-friendly way: familiar physical components (e.g. springs, transistors, hydraulic pumps, etc.) can be used to build models. Translation from this high-level description to low-level data flow is performed automatically in a quite satisfactory way, which frees users from a painful work. Moreover, even slight modifications of physical models may require considerable changes in corresponding block-diagram descriptions; this is not the case with a high-level description.

In its initial version, Modelicac essentially focussed on the "continuous part" of hybrid models. This mainly comprises differential equations and event-trigerring mechanisms (e.g. , "when equations"). Difference equations were however also be described, although with many restrictions, because the idea was to discourage users from writing discrete equations in Modelica. Indeed, Scicos is primarily a hybrid modelling environment and, in particular, it handles discrete, event-trigered changes, much more robustly than Modelica because of its synchronous roots.

In the course of the SimPA project, the Scicos editor has been extended to enable graphical handling of Modelica, native Scicos, as well as hybrid Modelica-Scicos blocks in the same design (see Figure 1).

This combination of synchronous and Modelica-based features offered enough modelling expressiveness to enable useful libraries to be developed. Coselica is one of these libraries, and is one of the ingredients of our

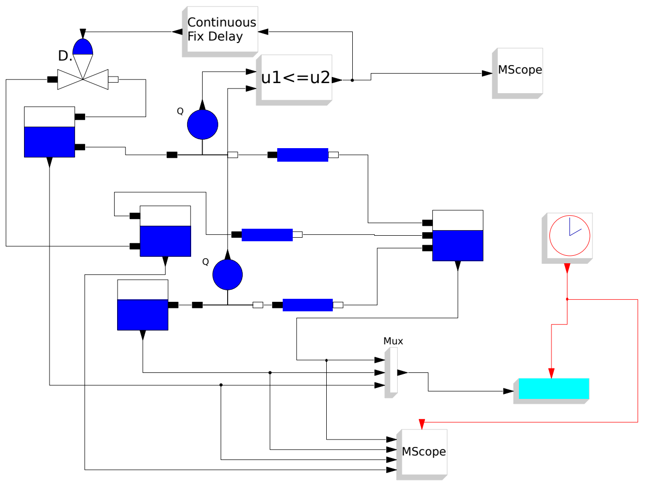


Figure 1. A mixed Scicos-Modelica model as displayed by Scicos's editor

Simulink to Modelica translation chain.

In 2005, the funded project SimPA2 started, having as objective the enhancement of the original Modelicac compiler. Among others, support of multiple-file Modelica libraries and interactive initialization of complex hybrid systems have been added.

As a result, the new Modelicac compiler was able to compete with industrial compilers (it even ranked number two in terms of performance on an industrial thermohydraulic benchmark proposed by EDF in 2009).

Today, the Scicos toolbox with its Modelica-compatible extension is freely available under several environments including Scilab, ScicosLab and Nsp.

2.4 Simport, a Simulink Model Importer for Scicos

2.4.1 Capabilities

The Simport (Chancelier et al., 2016, 2015) development started in 2007 at Inria: it has now turned into a comprehensive Simulink import assistant for the Scicos and Altair Activate block system modelers: Simport reads a textual representation of a Simulink model (MDL or SLX file format) and generates the corresponding equivalent Scicos model.

Based on compilation techniques, Simport is a fast and reliable translator from Simulink models to Scicos or Altair Activate models.

Simport is a free software distributed with Nsp Scicos (Scicos) and Activate (Altair Activate).

2.4.2 Capabilities

Simport aims at preserving the original Simulink model semantics: simport performs passes of semantic analysis to explicit the Simulink model meaning and translate it into an equivalent Scicos model.

In any case, the resulting Scicos model preserves the model hierarchy and diagram topology, and the visual aspects of the original model.

Simport also supports both the MDL and SLX formats as input for Simulink models.

2.4.3 Simulink block translation

Simport maps Simulink bloks to Scicos blocks using the *block translation library*. More precisely, each Simulink source block is translated either into

- a single basic block, if there exists an equivalent Scicos block,
- a *super-block* that implements the Simulink block via a combination of Scicos blocks,
- an empty super-block for user completion, if the Simulink block translation is unsupported

The Simport translation library covers a large subset of Simulink basic blocks, in particular the so-called *action blocks*.

2.4.4 Simport generated code

The Simport back-end translates explicit semantics of Simulink models to concrete code of the Scicos host language (Nsp or Oml). In addition, the concrete code provides the definition of simulation parameters and embeds various outputs to the host language (e.g. Matlab supporting M-files).

2.4.5 Example

Given the Simulink model described in Figure 2 and saved as file `model.mdl`.

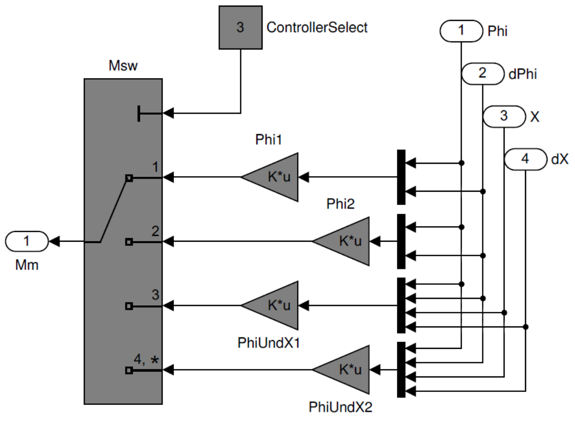


Figure 2. Segway controller as a Simulink model

We translate it into Nsp using the command `simport -tl nsp model.mdl`. We now get file `model.nsp` whose execution in Nsp produce build the Scicos model displayed in Figure 3.

2.4.6 Limitations

Simport indeed supports a large subset of Simulink basic blocks, but exotic blocks from specific Simulink libraries cannot be translated since they have no Scicos equivalent;

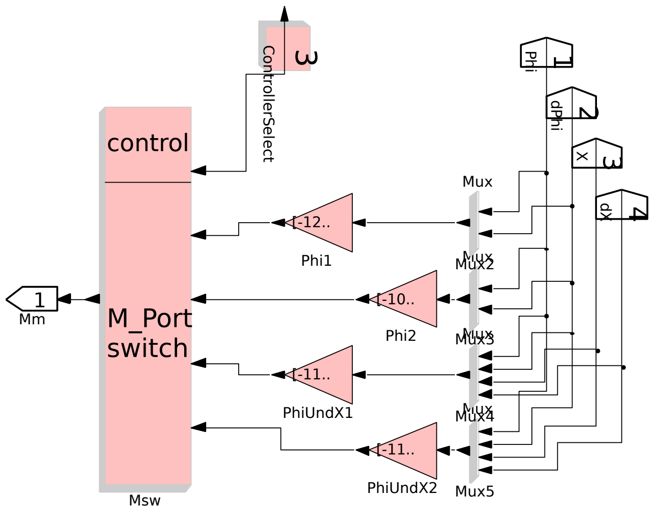


Figure 3. Segway controller as a Scicos model

in such a case, Simport generates an empty Scicos super block to incorporate the mandatory hand written Simulink block translation.

3 Translation from Simulink to Modelica

The Translation from Simulink to Modelica is implemented as a two step process. First, as already described, using Simport, we translate a Simulink model into a Scicos model. Second, using Nsp scripts we convert the Scicos model into an hybrid Modelica-Scicos model. Conversion is obtained by 1) translation of Scicos blocks to Modelica blocks, and 2) addition in the model of converters along the links which connect Scicos Blocks to Modelica-Scicos blocks. The hybrid Modelica-Scicos models can be edited and simulated in Scicos editor; thus, even if during the Translation process we cannot obtain a full Modelica model[1], the resulting hybrid model may still be used for simulation because users have the possibility to complete untranslated parts thanks to the Scicos editor.

Figure 4 is a Scicos model simulating the Lorenz dynamical system. The same model after conversion to Modelica is shown in Figure 5. As it can be seen in Figure 5 the Scopes are not translated to Modelica blocks and converters from Modelica signals to Scicos signals are inserted in the links connected to the entry ports of scopes.

When converters are available, Scicos blocks are replaced by Modelica blocks as a one-to-one process. We have developed a specific library of Modelica blocks to ease the replacement. For example Scicos integrator blocks are replaced by `MB_Integral` Modelica blocks. For some translation we rely on the already available library Coselica (Reusch), but for many blocks a direct Coselica translation fails because of size lim-

[1]In case some blocks are unknown to Simport. Indeed, Simulink blocks are black boxes, so Simport cannot translate blocks or combinations of blocks that are not already described in its translation tables.

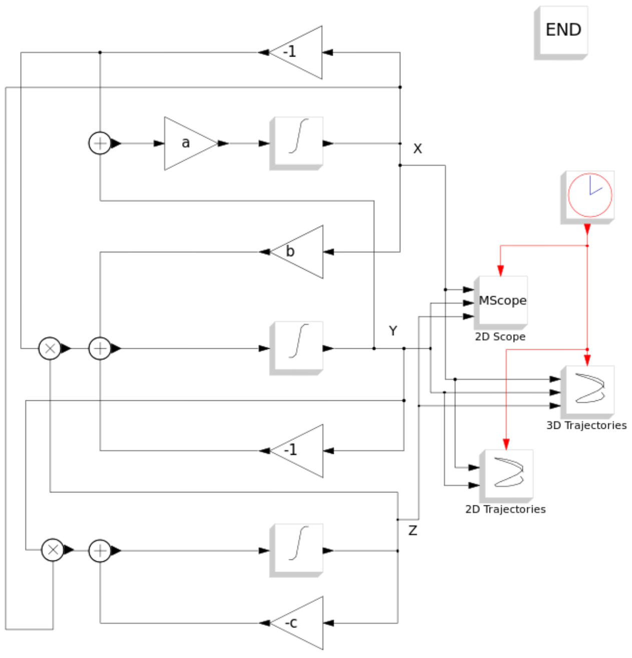


Figure 4. A Scicos model as displayed by Scicos's editor

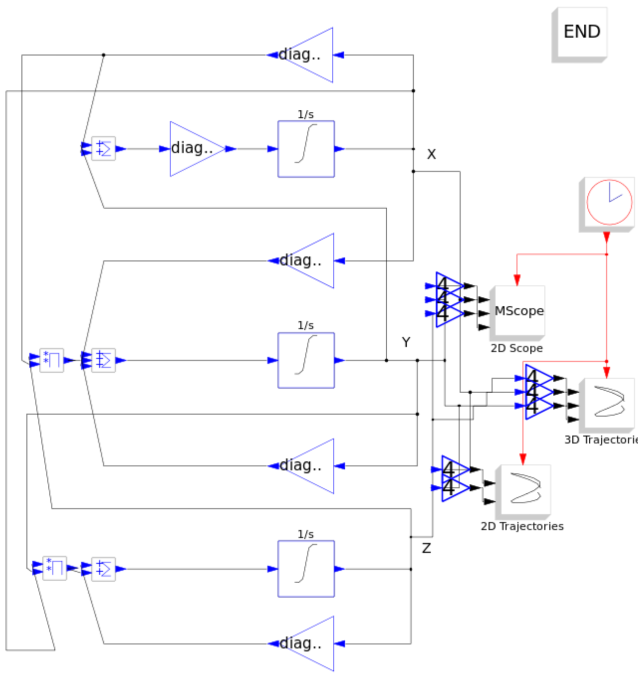


Figure 5. Scicos model after Modelica conversion as displayed by Scicos's editor

itations of Coselica blocks. For example the Coselica `Integrator` is limited to 1-dimensional signals while the Scicos `INTEGRAL_m` block may have n-dimensional entries. One way to encompass that difficulty is to generate super blocks for enabling n-dimensional block operations from 1-dimensional basic blocks (See Figure 7 for an example with adder). We have chosen this approach for the converter blocks (See Figure 6) as explained below, but we also implemented specific blocks to deal with n-dimensional signals. For example, the `MB_Integral` block is a special purpose Modelica-Scicos block which produces at compile time a new Modelica model for each instance of the block in a specific model. As an example, in Figure 5 each `MB_Integral` Modelica block integrate a 4-dimensional variable without saturation and thus the generated code will be given by

```
model integral2
 parameter Real xinit[   4,1 ] = {{   20 },{   19.9900 },{
      20.0100 },{   20.0110 }};
 RealInput u[4];
 RealOutput y[4](signal(start=xinit[:,1]));
equation
  der(y[1].signal) = u[1].signal;
  der(y[2].signal) = u[2].signal;
  der(y[3].signal) = u[3].signal;
  der(y[4].signal) = u[4].signal;
end integral2;
```

Most of the one-to-one block conversion follows the same mechanism. Building a library of Modelica-Scicos blocks is an on-going work and for the time being it only contains about 20 blocks. Indeed, this Library can also be used to directly build models in the Scicos editor, it complements the set of Modelica block available in Scicos giving access to Modelica counterpart of known Scicos blocks.

The one-to-one block conversion is in fact also a multi-step process. We proceed as follows.

First block-to-block conversions are performed but converted Modelica-Scicos blocs are not fully usable because they lack local information (for example the final matrix sizes are unknown at first step). Notice that this first step requires Nsp evaluation of block parameters since they may be used to infer types and dimensions. For example the sizes of a Gain block parameter gives the input/output port sizes of the block, except when the parameter size if 1. But in order to obtain the sizes of a given Gain block parameter we need to evaluate Nsp expressions, since parameters can be given through context (produced by Simport from Matlab companion files).

In a second step, links are modified and converters are inserted where appropriate. Notice however that converters sizes are also unknown.

In a third step, sizes and types are obtained by calling the Scicos model compiler. However, since the Scicos model compiler only infers types and dimension for Scicos blocks this step requires a hidden conversion of the hybrid Modelica-Scicos model into a pure Scicos model before trying to infer sizes and types. When sizes and types are inferred for a Modelica-Scicos block, its internal Modelica code can be generated. The code is thus consistent with respect to sizes, types and parameters.

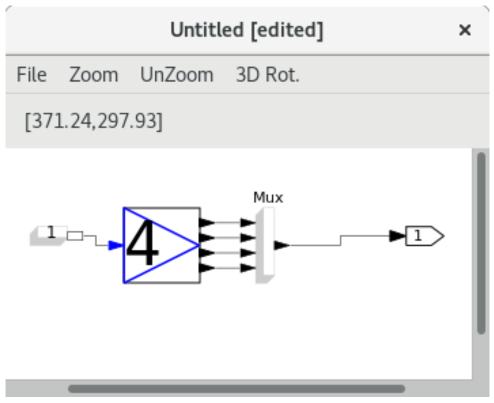


Figure 6. Scicos internal model of a 4-dimensional Modelica to Scicos converter

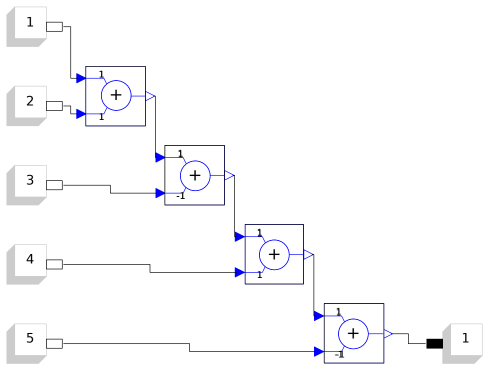


Figure 7. Scicos internal model of a generated 5-dimensional addition block

The fact that models can be manipulated and generated programmatically is also used in the conversion process. We illustrate this point by describing more precisely `MB_MO2Sn` the block used to convert Modelica signals to Scicos Signals. The communication between Scicos and Modelica can only be realized using scalar links (for historical reasons, not because of limitations of any of the languages), thus to have converters on links which transfer n-dimensional signals we have implemented a block named `MB_MO2Sn` as a super-block. That is, the `MB_MO2Sn` block contains a model and this model is generated dynamically when the link signal size is known. We give in Figure 6 the internal model of a 4-dimensional Modelica to Scicos converter as used in the model displayed in Figure 5. It contains four 1-dimensional Modelica to Scicos converters. To illustrate the possibility to generate models by program, we give in Figure 7 an example of a model which performs a 5-dimensional addition of Modelica signals. Implementing a n-dimensional adder block could be implemented that way, even if we chose to directly embed the Modelica n-dimensional adder block in a unique block.

As already pointed above, during the conversion from Scicos models to Modelica-Scicos models, inferring types

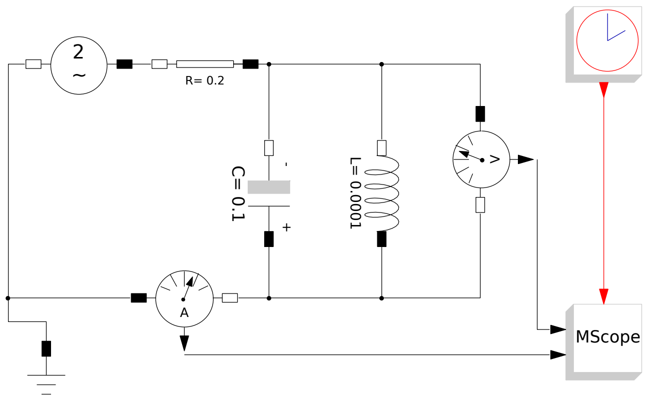


Figure 8. A mixed Scicos-Modelica model of a RLC circuit

and sizes is of utmost importance and it partially relies on Nsp block parameter and context expression evaluation. This is mostly why the conversion cannot completely be performed by Simport. Indeed, inferring types and sizes could have been implemented directly in Simport, if evaluation of Matlab expression were not required in the process.

3.1 Translation from Modelica to C

Modelica source code is translated to C thanks to the Modelicac compiler. The idea is as follow. Once the model is being run by the user, Scicos gathers all the blocks whose execution semantics is described by means of Modelica code into a unique Modelica program whose source code is given to Modelicac. This program is actually a high-level description of the Modelica part of the model. The following listing illustrates what such a description looks like. It contains the Modelica description generated by Scicos for the model of Figure 8:

```
model RLC_circuit_test_im
  parameter Real VA_VsourceAC_(fixed=false) =
    2.000000e+00  "VA_VsourceAC_";
  parameter Real f_VsourceAC_(fixed=false) =
    1.000000e+00  "f_VsourceAC_";
  parameter Real R_Resistor_(fixed=false) =
    2.000000e-01  "R_Resistor_";
  parameter Real C_Capacitor_(fixed=false) =
    1.000000e-01  "C_Capacitor_";
  parameter Real v_Capacitor_(fixed=false) =
    0.000000e+00  "v_Capacitor_";
  parameter Real L_Inductor_(fixed=false) =
    1.000000e-04  "L_Inductor_";
  VsourceAC     VsourceAC_(VA=VA_VsourceAC_,
                           f=f_VsourceAC_);
  Resistor      Resistor_(R=R_Resistor_);
  Capacitor     Capacitor_(C=C_Capacitor_,
                           v(start=v_Capacitor_));
  Inductor      Inductor_(L=L_Inductor_);
  CurrentSensor CurrentSensor_;
  Ground        Ground_;
  VoltageSensor VoltageSensor_;
  OutPutPort    OutPutPort_;
  OutPutPort    OutPutPort_1;
equation
  connect (CurrentSensor_.n,VoltageSensor_.n);
  connect (Capacitor_.p,VoltageSensor_.n);
  connect (Inductor_.p,VoltageSensor_.n);
  connect (Ground_.p,VsourceAC_.n);
  connect (CurrentSensor_.p,VsourceAC_.n);
  connect (VoltageSensor_.p,Resistor_.p);
  connect (Inductor_.n,Resistor_.p);
  connect (Capacitor_.n,Resistor_.p);
  connect (Resistor_.n,VsourceAC_.p);
  CurrentSensor_.i = OutPutPort_.vi;
  VoltageSensor_.v = OutPutPort_1.vi;
end RLC_circuit_test_im;
```

Modelica programs generated by Scicos contain five (possibly empty) sections declaring respectively:

- the parameters of the model,
- the components appearing in the model (i.e. Modelica “blocks” used to build the model),
- the connectors to and from the Scicos world (declared as `InPutPorts` and `OutPutPorts`),
- the connection equations (corresponding to links between components of the model, introduced by means of the `connect` keyword), and
- the correspondence between some Scicos ports and some Modelica connectors used to exchange information between both worlds (introduced by means of an equal sign).

From such Modelica programs Modelicac generates native, C-based Scicos blocks. It starts by resolving the names appearing in the Modelica description and instantiates required classes (found in libraries) to form the set of all equations governing the dynamics of the Modelica part of the model. It then flattens the structure of the Modelica model, simplifies equations, and generates C code. The following listing is the result of calling Modelicac with previous Modelica code:

```
/* Scicos block's entry point */

void RLC_circuit_test_im(
  scicos_block *block,
  int flag)
{
  int     *ipar = GetIparPtrs(block);
  double *rpar = GetRparPtrs(block);
  double *z    = GetDstate( block);
  double *x    = GetState(block);
  double *xd   = GetDerState(block);
  double *res  = GetResState(block);
  double **y   = GetOutPtrs(block);
  double **u   = GetInPtrs(block);
  double **work= GetPtrWorkPtrs(block);
  double *g    = GetGPtrs(block);
  double *alpha= NULL;
  double *beta = NULL;
  int *jroot   = GetJrootPtrs(block);
  int *mode    = GetModePtrs( block);
  int nevprt   = GetNevIn( block);
  int *xprop   = GetXpropPtrs(block);

  /* Intermediate variables */
  double v0;

  if (flag == 0) {
    res[0] =
      (x[1]+x[0]*
       (*GetRealOparPtrs(block,3))+
       sin(6.28318530718*
           GetScicosTime(block)*
           (*GetRealOparPtrs(block,2)))*
       (*GetRealOparPtrs(block,1)))*(1.0);
    res[1] = x[2]+ xd[1]*(*GetRealOparPtrs(block,4))-x[0];
    res[2] = xd[2]*(*GetRealOparPtrs(block,6))-x[1];
  } else if (flag == 1) {
    if (!areModesFixed(block)) {
      y[0][0] = x[0]; /* OutPutPort_.vo */
      y[1][0] = -x[1]; /* OutPutPort_1.vo */
    } else {
      y[0][0] = x[0]; /* OutPutPort_.vo */
      y[1][0] = -x[1]; /* OutPutPort_1.vo */
    }
  } else if (flag == 2 && nevprt < 0) {
  } else if (flag == 4) {
      x[0] = 0.0; /* Resistor_.i */
      x[1] = (*GetRealOparPtrs(block,5)); /* Capacitor_.v */
      x[2] = 0.0; /* Inductor_.i */
      if (GetNopar(block)<6){
        SetBlockError(block,-21);
        return;
      }
      SetAjac(block,1);
    } else if (flag == 5) {
    } else if (flag == 6) {
    } else if (flag == 7) {
      xprop[0] = -1; /* Resistor_.i (algebraic) */
      xprop[1] = 1; /* Capacitor_.v (state) */
      xprop[2] = 1; /* Inductor_.i (state) */
    } else if (flag == 9) {
    } else if (flag == 10) {
      alpha=GetAlphaPt(block);
      beta =GetBetaPt(block);
      res[0] = (*GetRealOparPtrs(block,3))*(1.0)*alpha[0];
      res[1] = -alpha[0];
      res[3] = (1.0)*alpha[1];
      res[4] = (*GetRealOparPtrs(block,4))*beta[1];
      v0 = -alpha[1];
      res[5] = v0;
      res[7] = alpha[2];
      res[8] = (*GetRealOparPtrs(block,6))*beta[2];
      res[9] = alpha[0];
      res[12] = v0;
    }

    return;
  }
```

Finally, a new native Scicos block running the generated C code is created by Scicos and connected to the Scicos part of the original model in place of the Modelica blocks. Scicos then performs the simulation of the resulting model.

4 Working with Acausal Models

Although supported Simulink models are limited to explicit input-output blocks, one should not conclude that our tool chain only deals with "causal" modeling. Indeed, Scicos provides a powerful editor (written in Nsp's language) by means of which imported Simulink models can be graphically connected to Modelica "acausal" models. This allows, for instance, control models written in Simulink to be imported in Nsp and connected to Modelica models (handled by Modelicac). Complete models featuring both "causal" and "acausal" aspects can then be simulated and possibly exported as Modelica code (Figure 1 and Figure 8 show examples of Modelica "acausal" models with control and displays modeled as block-diagrams).

5 Conclusion and Future Work

In this paper we have shown that NSP can be used as a powerful environment to translate, possibly edit, and simulate some Simulink models. The tool chain comprises two external tools, namely Simport and Modelicac, and the Scicos toolbox with its hybrid Scicos-Modelica library Coselica.

This addition to NSP allows users to simulate many Simulink models with few changes, if any. Moreover, original models can be edited after translation, and possibly connected to native Scicos models, or to Modelica models. This opens many interesting perpectives for users willing to run heterogeneous models at a very high level, using the graphical editor provided by Scicos as the sole GUI.

Several enhancements can be made to this preliminary work. The most significant enhancement would probably consists in enriching the Scicos to Modelica translation table, to allow more Simulink models to be translated automatically.

6 Acknowledgements

We would like to thank Ramine Nikoukhah from Altair for his considerable help in the design and implementation of the tools we used, in particular Scicos of course, but also Modelicac and Simport.

References

Altair Activate. Multi-Disciplinary System Simulation. URL https://solidthinking.com/product/activate.

Stephen Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. Springer, 2006. ISBN: 978-0-387-27802-5.

Jean-Philippe Chancelier, Franḉois Delebecque, Clément Franchini, Ramine Nikoukhah, and Pierre Weis. Simport: A Simulink Model Importer for Scicos. In *The 3rd International Workshop on Simulation at the System Level*, École Normale Supérieure de Cachan, France, 2015.

Jean-Philippe Chancelier, Franḉois Delebecque, Clément Franchini, Ramine Nikoukhah, and Pierre Weis. Simport. In *ISC'2016 Bucharest*, 2016.

Mike Dempsey. Automatic translation of simulink models into modelica using simelica and the advancedblocks library. In *Proceedings of the 3rd International Modelica Conference, Linköping, Sweden*, pages 115–124, 2003.

Modelica. The modelica language specification. URL https://modelica.org/documents.

Ramine Nikoukhah and Sébastien Furic. Towards a full integration of modelica models in the scicos environment. In *Proceedings of the 7th International Modelica Conference, Como, Italy*, pages 641–645, 2009.

Ramine Nikoukhah and Masoud Najafi. Initialization of modelica models in scicos. In *Proceedings of the 6th International Modelica Conference, Bielefeld, Germany*, pages 37–46, 2008.

Nsp. A Numerical computing environment (GPL). URL http://cermics.enpc.fr/~jpc/nsp-tiddly/mine.html.

Dirk Reusch. Coselica toolbox für scicoslab. URL http://www.kybdr.de/software#coselica_toolbox_fuer_scicoslab.

Scicos. Block diagram modeler/simulator. URL http://www.scicos.org/.

ScilabGtk. Gtk+ version of Scilab. URL http://www.scilabgtk.org.

Simulink. System modeling and simulation. URL https://www.mathworks.com/products/simulink.html.

SPONSORS & EXHIBITORS

Bronze Sponsors & Exhibitors

ANSYS®

asim

dSPACE

Modelon

OpenModelica

schlegel simulation

SCIL

SYSTEMS & CONTROL
INNOVATION LAB

Silver Sponsors & Exhibitors

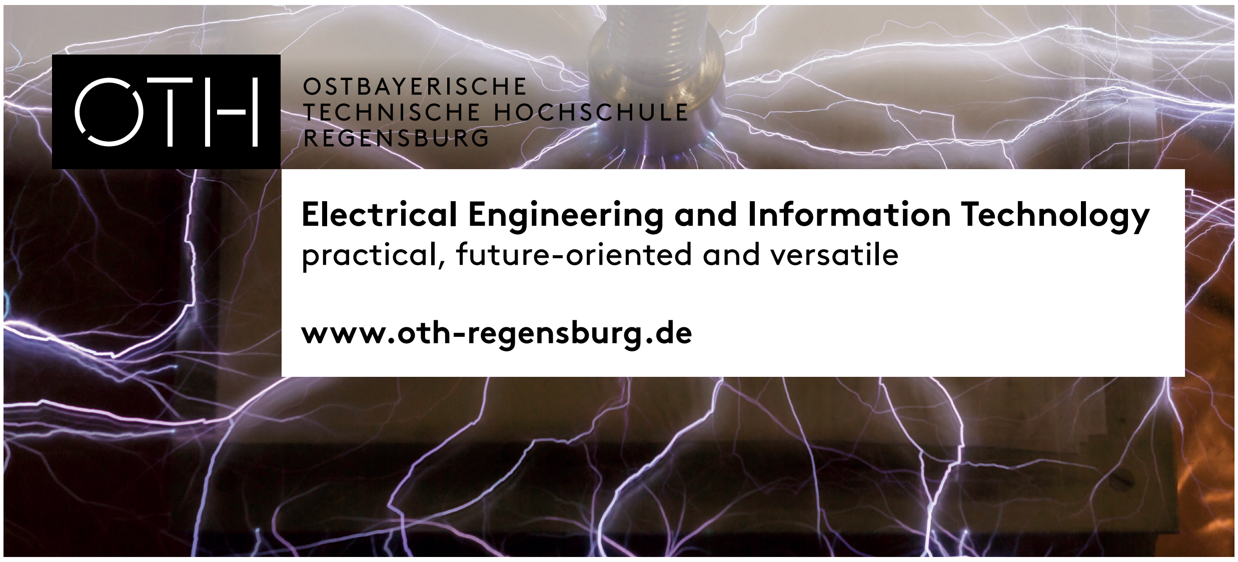
CLAYTEX

concurrent
REAL-TIME

esi
get it right®

Maplesoft™
Mathematics • Modeling • Simulation
A Cybernet Group Company

WOLFRAM
SYSTEMMODELER™



OTH
OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Electrical Engineering and Information Technology
practical, future-oriented and versatile

www.oth-regensburg.de

Gold Sponsors & Exhibitors

Gaio Technology provides tools and engineering services for Model-based development and System/Software Verification and Evaluation.
Our biggest customer is the automotive industry, including both classic domains as well as the newer fields like AD/ADAS.
We developed our own simulator for more than a hundred different MPU/MCUs, and use this technology to provide tools and solutions for Virtual Testing and ECU Simulation. We have enough experience and many successful projects using mathematical, E/E or physical models allowing coupled simulation.
For more details and/or tool demos please come visit our booth or our Vendor Session on Monday March 4th, 18:30.

GAIO
TECHNOLOGY

SIEMENS
Ingenuity for life

Today's dreams need tomorrow's engineering.

Simcenter: Engineer innovation.

Getting a dream rolling has never been more challenging. Products are smarter. Manufacturing processes are more complex. And design cycles are shorter than ever. Simcenter software can help. With its unique combination of multi-disciplinary simulation, advanced testing and data analytics, Simcenter gives you the power to explore alternatives faster, predict performance more accurately... and deliver innovation with greater confidence.

siemens.com/simcenter

Platin Sponsors & Exhibitors

An Open Integration Platform for Multi-Disciplinary System Simulation

Download a free trial at altair.com/Activate

Altair

DYMOLA and MODELICA

Since 1999

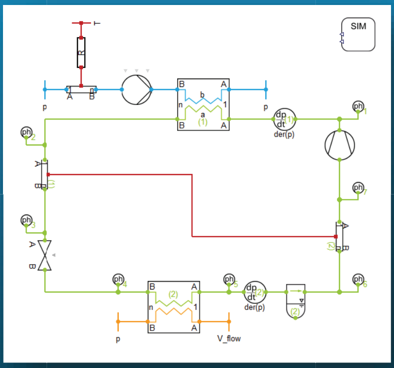
DASSAULT SYSTEMES | The **3D**EXPERIENCE® Company

LTX Simulation GmbH

Simulation Tools and Modeling Services by Modelica Experts

- **Modelica Libraries** from DLR, TLK, XRG and others
- Sales & technical support for **Dymola**
- Automation and post-processing based on
  SCORE
  DaVE
  MoBA Lab
- Model development for **thermal systems**
- Training on Modelica and Dymola

Please visit our booth for hands-on experience of our tools.



www.ltx.de | sales@ltx.de | +49 89 20970055
LTX Simulation GmbH | Wohlfartstr. 21b | 80939 Munich | Germany
Dymola is a registered trademark of Dassault Systèmes.
Modelica is a registered trademark of the Modelica Association.