Asian Modelica Conference 2022 is sponsored by:

Gold sponsors:

ALTAIR Ansys DASSAULT SYSTEMES Maplesoft Modelon

远思智能 SIMTEK Neorium Technology TOSHIBA TOYOTA TSUSHO SYSTEMS

Silver sponsors:

ARGO GRAPHICS iVH SIEMENS

Asian Modelica Conference 2022 is organized by Modelica Association in cooperation with Sophia University and Modelon K.K.

Proceedings of Asian Modelica Conference 2022
Tokyo, Japan, November 24-25, 2022

Editors:
Prof. Tielong Shen, Dr. Rui Gao and Dr. Yutaka Hirano

Linköping Electronic Conference Proceedings Nr. 193
ISBN: 978-91-7929-578-3   (PDF)
ISSN: 1650-3686
eISSN: 1650-3740
DOI for entire proceeding: https://doi.org/10.3384/ecp193

Organized by:
Modelica Association
c/o PELAB, Linköpings Univ.
SE-581 83 Linköping
Sweden

In cooperation with:
Sophia University
7-1 Kioi-cho, Chiyoda-ku,
Tokyo, 102-8554
Japan

Modelon K.K.
1-10-3-209, Roppongi, Minato-ku,
Tokyo 106-0032
Japan

Conference location:
Sophia University Yotsuya Campus and Virtual platform in Whova

Welcome Message

Following the first and the second conferences in 2016 and 2018 as the Japanese Modelica Conference, and the third conference as The Asian Modelica Conference 2020, The Asian Modelica Conference 2022 takes place in Tokyo again.
The local organizer and Modelica Association work closely to overcome various difficulties caused by continuing COVID-19 pandemic to make the conference available. We are now proud to present a conference with:

- 2 Keynote speeches
- 19 paper presentations
- An exhibition area featuring 9 exhibitors
- Hybrid event at the venue of Sophia University and virtual event platform on Whova

According to Modelica Association standards, all papers are peer-reviewed and will be freely available for download.
We want to acknowledge the support we received from the conference board and program committee. Special thanks to our colleagues at Modelon KK for taking care of all the practical matters. Support from the conference sponsors is gratefully acknowledged. Last but not least, thanks to all authors, keynote speakers, and presenters for their contributions to this conference.
With the hybrid environment of the real venue and the virtual platform, the objective of creating the conference as an arena in Asia for sharing knowledge and learning about the latest scientific and industrial progress related to Modelica and FMI (Functional Mock-up Interface) is unchanged. We wish all participants an enjoyable and inspiring conference!

Tokyo, November 24, 2022

Tielong Shen, Rui Gao & Yutaka Hirano

Keynotes Speakers

Mazen Alamir
Research Director, CNRS

Plenary session I: Model Predictive Control: An attempt to tell an unfinished story!

Abstract
While it is true that any control design methodology is based on some modelling of the underlying dynamics, the most advanced of these techniques, namely Model Predictive Control (MPC) cannot be conceived in the absence of such a dynamic model. The predominant place of MPC in the control engineering landscape since the last two decades is justified by its ability to handle nonlinearities, constraints and optimality concerns on one hand and by the availability of dedicated efficient optimization solvers on the other hand. In this talk, the basics of MPC are introduced and the main related research topics are shortly and progressively discussed including among others: Stability issues, Real-time implementation, Handling uncertainties via stochastic MPC, GPU-based parallel implementation as well as appropriate use of Machine Learning based blocs. Industrial and real-life examples are used throughout the talk in order to illustrate the underlying tools and methods.

Yudai Yamasaki
Professor, The University of Tokyo

Plenary session 2: Toward advanced powertrain control technologies based on models

Abstract
Powertrain control technologies of vehicles are important for carbon neutrality. The powertrain system is getting more complicated for its hybridization, and an internal combustion engine is also a complicated component itself in nature. The control system is also becoming complicated and the conventional control framework using look-up tables based on a huge number of experiments is difficult to continue. Furthermore, the framework of control is changing, as it is now possible to control the powertrain using and combining a variety of information, including connected data and driver's data. Model-based control system is more useful and essential to combine different things, disciplines and researchers for advanced powertrain control. In this presentation, I will introduce our research activities on advanced control systems of automobiles based on models.

Program Committee

General Schedule

Day1: November 24th

Time Start	Time End	Session Title
9:50 AM	10:00 AM	Opening session
10:00 AM	12:00 PM	Tutorial session
12:00 PM	12:50 PM	Lunch, Break
12:50 PM	2:10 PM	Oral session A:　Thermal and power system (1)
2:10 PM	2:20 PM	Coffee Break
2:20 PM	3:40 PM	Oral session B:　Thermal and power system (2), Tools
3:40 PM	4:00 PM	Coffee Break
4:00 PM	4:45 PM	Sponsor session I
4:45 PM	5:35 PM	Plenary session 1:
5:35 PM	6:30 PM	Break
6:30 PM	20;30	Conference dinner

Day2: November 25th

Time Start	Time End	Session Title
9:00 AM	9:50 AM	Plenary session 2:
9:50 AM	10:00 AM	Coffee Break
10:00 AM	11:40 AM	Oral session C; Thermal and power system (3), Mechanical system
11:40 AM	1:00 PM	Lunch, Break
1:00 PM	1:45 PM	Sponsor session II
1:45 PM	2:00 PM	Lunch, Break
2:00 PM	4:00 PM	Oral session D: Tools, FMI related
4:00 PM	4:15 PM	Break
4:15 PM	5:00 PM	Sponsor session III
5:00 PM	6:00 PM	Break, Withdrawal
6:00 PM	8:00 PM	Farewell Reception & Closing Ceremony

Scientific Program

November 24th

Tutorial session: How Modelica works for plant modeling and model-based control design

10:00 AM-12:00 PM

Yutaka Hirano, Woven Planet Holdings, Inc.

Oral session A: Thermal and power system (1)

Chair: Yutaka Hirano

A-1: Dynamic simulation of an oxygen-hydrogen combustion turbine system using Modelica

12:50 PM-1:10 PM

Speaker: Yutaka Watanabe

A-2: Techno-economic assessment of an industrial project towards carbon neutrality

1:10 PM-1:30 PM

Speaker: Arne Köppen

A-3: Towards modeling and simulation of dynamic overconstrained connectors in Modelica

1:30 PM-1:50 PM

Speaker: John Tinnerholm

A-4: Solving flow balancing problem for hybrid-electric aircraft cooling systems

1:50 PM-2:10 PM

Speaker: Clément Coïc

Oral session B: Thermal and power system (2)

Chair: Guillaume Viry

B-1: High-fidelity multiphysics FCEV bus study with detailed HVAC, cabin, and hydrogen fuel cell models

2:20 PM-2:40 PM

Speaker: Theodor Ensbury

B-2: slPCMlib: A Modelica library for the prediction of effective thermal material properties of solid/liquid phase change materials (PCM)
2:40 PM-3:00 PM
Speaker: Tilman Barz

B-3: [Industrial Paper] Digital Twin applications using a cloud native Modelica platform
3:00 PM-3:20 PM
Speaker: Stéphane Velut

B-4: Advanced open source data formats for geometrically and physically coupled systems
3:20 PM-3:40 PM
Speaker: Andreas Naumann

Sponsor session I

Neorium Technology solution introduction
4:00 PM-4:15 PM
Toshiba Digital solution introduction
4:15 PM-4:30 PM
Modelon solution introduction
4:30 PM-4:45 PM

Plenary session I: Model Predictive Control: An attempt to tell an unfinished story!

4:45 PM-5:35 PM
Mazen Alamir, Research Director CNRS

November 25th

Plenary session 2: Toward advanced powertrain control technologies based on models

9:00 AM-9:50 AM
Yudai Yamasaki, Professor, The University of Tokyo

Oral session C: Thermal and power system (3), Mechanics system

Chair: Moritz Hübel

C-1: Optimal design for thermodynamic system with OpenModelica and MATLAB/Simulink

10:00 AM-10:20 AM

Speaker: Kazuki Yoshida

C-2: Study on BEV concept design based on data driven approach

10:20 AM-10:40 AM

Speaker: Juhyeong Park

C-3: Modelling and optimal design of gas engine CCHP system in hospital

10:40 AM-11:00 AM

Speaker: Zheng Qian

C-4: [Industrial Paper] eMobility & energy management, development for sustainable mobility on 3DEXPERIENCE platform

11:00 AM-11:20 AM

Speaker: Guillaume Viry

C-5: Physical modeling of daily electric appliances for education by Modelica

11:20 AM-11:40 PM

Speaker: Yutaka Hirano

Sponsor session II

Maplesoft solution introduction

1:00 PM-1:15 PM

Altair Engineering solution introduction

1:15 PM-1:30 PM

Ansys solution introduction

1:30 PM-1:45 PM

Oral session D: Tools, FMI related

Chair: Adrian Pop

D-1: Importing FMU-3.0: challenges in proper handling of clocks

2:00 PM-2:20 PM

Speaker: Masoud Najafi

D-2: [Industrial Paper] Performance measurement and finding challenges in using FMUs to perform scenario-based testing in a cloud environment

2:20 PM-2:40 PM

Speaker: Katsuya Tsuzuki

D-3: Simulation scheduling of variable-structure models in OpenModelica

2:40 PM-3:00 PM

Speaker: Rahul Paknikar

D-4: Hybrid physical-AI based system modeling and simulation approach demonstrated on an automotive fuel cell

3:00 PM-3:20 PM

Speaker: Moritz Hübel

D-5: A Dymola-Python framework for data-driven model creation and Co-simulation

3:20 PM-3:40 PM

Speaker: Sandra Wilfling

D-6: Towards continuous simulation credibility assessment

3:40 PM-4:00 PM

Speaker: Maurizio Ahmann

Sponsor session III

SimTEK solution introduction

4:15 PM-5:00 PM

Dassault Systemes solution introduction

4:30 PM-4:45 PM

Toyota Tsusho Systems solution introduction

4:45 PM-5:00 PM

Contents

Author Index

Dynamic Simulation of an Oxygen-Hydrogen Combustion Turbine System Using Modelica

Yutaka Watanabe Toru Takahashi Kojun Suzuki

Central Research Institute of Electric Power Industry, Japan, {yutaka,toru-tak,s-kojun}@criepi.denken.or.jp

Abstract

Introducing hydrogen power generation in the power industry may contribute to achieve carbon neutrality by 2050. Hydrogen mixed-fuel gas turbines are available, and pure hydrogen-fueled gas turbines are being developed. Meanwhile, power generation systems based on oxygen–hydrogen combustion turbines have been devised. Such system uses hydrogen as fuel and oxygen as oxidizer, yielding only water vapor as the byproduct of combustion. In addition, as the system performs a semi-closed cycle involving the Brayton and Rankine cycles, high efficient zero-emission power generation is expected with higher thermal efficiency than that of the combined cycle in conventional gas turbines. Basic technologies for oxygen-hydrogen combustion turbines in power generation systems are being developed in Japan as part of the research and development at NEDO for hydrogen utilization. In this study, a dynamic model of the entire system for a 1400 °C-class rationalization system was constructed using a Modelica-based tool developed by the Central Research Institute of Electric Power Industry, Japan. The dynamic behavior considering preliminary load following control was then characterized based on simulation results.

Keywords: Closed cycle, Hydrogen, Dynamic simulation, Load following

Nomenclature

A	flow area [m^2]
K	heat transfer coefficient [$kW/(m^2 \cdot K)$]
cp	heat capacity [$kW/(kg \cdot K)$]
F	mass flow rate [kg/s]
H	specific enthalpy [kJ/kg]
LHV/HHV	lower/higher heating value [kJ/kg]
k	specific heat ratio [-]
M	mass [kg]
Q	heat transfer rate [kJ/s]
P	pressure [MPa]
v	specific volume [m^3/kg]
W	power [kW]
η	adiabatic efficiency [-]

Subscripts

ad	adiabatic change
c	cold flow
cb	combustion
cp	compressor
f	fuel
h	hot flow
i	inlet
j	segment number
m	metal
o	outlet
st	steam turbine

1 Introduction

To address climate change, actions toward decarbonization and net-zero CO2 emissions in the global energy sector must be implemented by 2050 (IEA, 2021). A carbon-neutral society will widely use hydrogen. Thus, hydrogen supply infrastructures and power generation systems are being increasingly demanded. Fuel cells are a well-known power generation method using hydrogen, but gas turbines (GTs) are more suitable for large-scale power generation. The impact of CO2 reduction will be significant if hydrogen is used at large-scale electric power generation plants. Currently, hydrogen mixed-fuel GTs are available, and pure hydrogen-fueled GTs are expected to be developed by 2030 (ETN, 2021). Meanwhile, a power generation system based on oxygen-hydrogen combustion turbines that perform direct combustion of these gases has been devised, seeming promising for high-efficiency hydrogen power generation. In this system, the sole byproduct of combustion at a very high temperature is water vapor if hydrogen and oxygen are combusted at the theoretically compatible ratio. As the conventional Rankine cycle is intended for external combustion engines, it is difficult to achieve higher temperatures than those in GTs, which perform internal combustion, owing to limitations in the heat resistance of heat transfer tubes in the boiler. In addition, in a conventional GT, the amount of generated NOx increases with increasing temperature. In contrast, in oxygen-hydrogen combustion, NOx generation is prevented owing to the absence of nitrogen in the reaction gas. Therefore, oxygen-hydrogen combustion turbines may be suitable for both combustion at higher temperatures and highly efficient zero-emission power generation. Hence, they may achieve higher power generation efficiency than conventional GTs with a combined cycle.

Oxygen-hydrogen closed cycles and their system parameter optimization have been widely studied, leading to systems such as those based on the Graz cycle (Sanz et al., 2018), a new Rankine cycle (Fukuda and Dozono, 2000; Soufi et al., 2004; Bannister et al., 1998; Schouten and Klein, 2020), and even topping with a fuel cell (Millewski, 2015). In Japan, the WE-NET project (Mitsugi et al., 1998) was conducted between 1993 and 1998 to achieve an efficiency above 60% (HHV) (approximately 71% regarding LHV) at a turbine inlet temperature of 1700 °C. A topping regeneration cycle and new Rankine cycle were proposed, demonstrating that the target efficiency could be achieved through optimization (Sugisita et al., 1998).

Since 2018, the development of basic technology for a power generation system based on oxygen-hydrogen combustion turbines is underway as part of a research and development project at the New Energy and Industrial Technology Development Organization (NEDO) for leading hydrogen utilization (NEDO, 2022). By 2020, the oxygen-hydrogen combustion power generation was reviewed considering the knowledge obtained from the WE-NET project, and a parameter study of the turbine inlet conditions (i.e., steam pressure and temperature) for achieving 75% (LHV) thermal efficiency was conducted using the Graz cycle. Simultaneously, a system that leverages the cycle by rationalizing the factors that increase costs while minimizing the decrease in thermal efficiency was investigated, leading to identify technical research problems to be addressed for obtaining a high-efficiency system. Technical studies on rationalization systems are being conducted based on results from the abovementioned pioneering studies.

Despite the advancements, most studies on oxygen-hydrogen combustion turbine cycles have been focused on the optimization of the cycle configuration and system performance. Such studies have demonstrated the possibility of achieving high efficiency, but few studies on operational characteristics such as start-up, shutdown, and load change are available, with an exception being a study on the start-up operation of the new Rankine cycle (Funatsu et al., 1998). When variable renewable energies are introduced into the power system in large quantities in the future, power generation systems are expected to have more opportunities to operate at partial load. Therefore, in addition to their rated performance, their operability should be evaluated. In fact, determining the dependence of the system performance on different load changes is important. For example, condensate generation, rapid temperature fluctuation, and excess temperature may occur during partial load operation. Hence, a system-level dynamic model should be established. Physical modeling is especially useful to predict both the dynamic behavior and off-design conditions for the entire operational range of a power generation system. In addition, the model is important for efficient development, prediction, and evaluation of operational characteristics of the entire system at the conceptual stage as well as understanding operational problems and specifications for components at an early stage. Although many analyses of thermal power generation systems have been performed using Modelica, few have been performed considering oxygen-hydrogen combustion turbines.

To evaluate the dynamic performance under load changes and capture the general behavior of the target system, we constructed a dynamic model of an oxygen-hydrogen combustion turbine system using a Modelica-based tool developed by the Central Research Institute of Electric Power Industry, Japan. Then, simple load following control was implemented, and the dynamic behavior was evaluated based on the corresponding simulation results.

2 Methods

2.1 System Configuration

Figure 1 shows a diagram of the system considered in this study, while Figure 2 shows the temperature–entropy diagram, and Table 1 lists the main performance parameters of the system. Although the system was based on the Graz cycle, it was intended to leverage the benefits of the cycle by omitting the high-pressure turbine and streamlining equipment specifications that increase manufacturing costs by using a combustion temperature of 1400 °C. Therefore, the system mainly comprises a combustor, high-temperature turbine (HTT), compressor, waste heat recovery boiler (HRSG), constant pressure steam turbine (LPT), condenser, and feedwater pump. When hydrogen and oxygen completely react in compatible quantities in the combustor, the only working fluid and combustion product is steam, and the system is characterized by a combined cycle configuration involving the high-temperature Brayton cycle and low-temperature Rankine cycle. The steam circulates in the system as the working fluid, and the water produced during combustion is discharged via a drain after passing through the condenser.

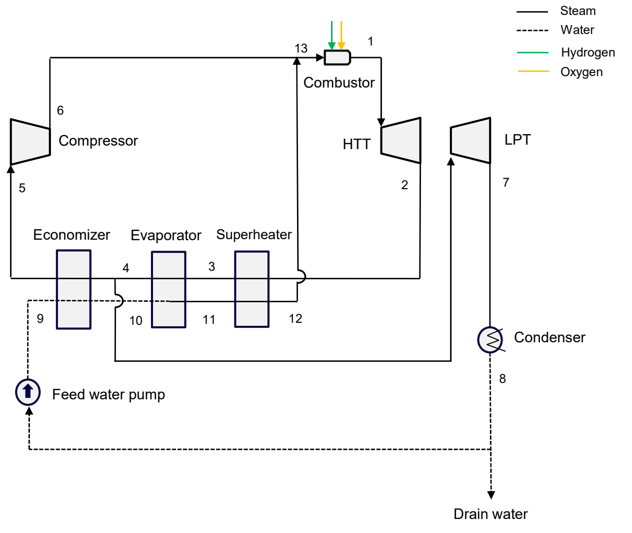


Figure 1. Schematic of target power generation system.

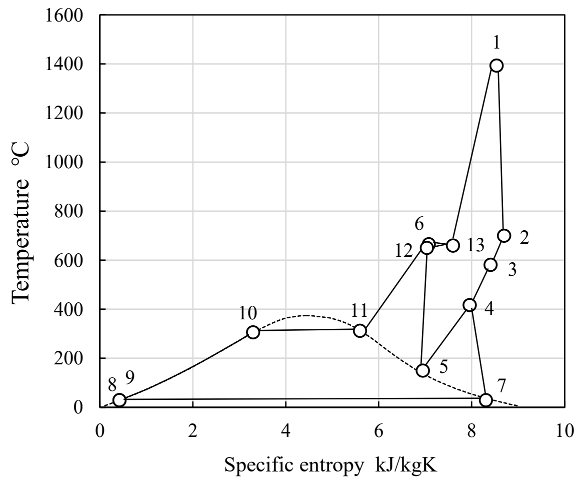


Figure 2. Temperature-entropy diagram of target power generation system.

Table 1. Main performance parameters of target power generation system.

Parameter	*Value*
Power output	102.3 MW
Thermal efficiency	68.3 %LHV
Combustion temperature	1393 °C
HTT inlet pressure	10.0 MPa

2.2 The Dynamic Model

A dynamic model of the system was established using the Modelica-based tool developed by the Central Research Institute of Electric Power Industry (Watanabe et al., 2017). The system model was constructed by combining static and dynamic component models and controllers, as shown in Figure 3. The SimulationX software was used for simulations.

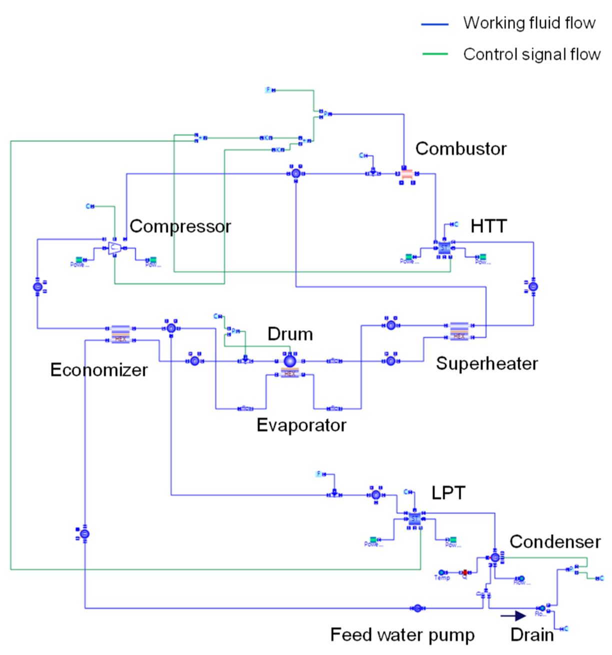


Figure 3. Overview of dynamic model.

We assume that hydrogen and oxygen are combusted in compatible proportions and that the working fluid contains no components other than water. In addition, the effects of residual oxygen and hydrogen are neglected.

2.2.1 Combustor

The equation of the combustion reaction is given by

$$2H_2 + O_2 \rightarrow 2H_2O \qquad (1)$$

The combustion is assumed to be instantaneous, and the heat generated is calculated statically from the calorific value of the fuel (hydrogen) as follows:

$$Q_\mathrm{f} = HHV_\mathrm{f} \cdot F_\mathrm{f} \qquad (2)$$

The combustion components are also calculated statically. The hydrogen flow rate is set as an input condition, and the oxygen flow rate depends on the hydrogen flow rate to meet the conditions for combustion at an equivalent ratio of 1.0. The combustion products are water vapor, and the amount of production is given by

$$F_{\mathrm{H20,cb}} = F_{\mathrm{H2}} + F_{\mathrm{O2}} \qquad (3)$$

The pressure and temperature are calculated for the dynamics of the volume model assuming that the fuel gas enters a vessel of constant volume. A diagram of the Modelica model is shown in Figure 4.

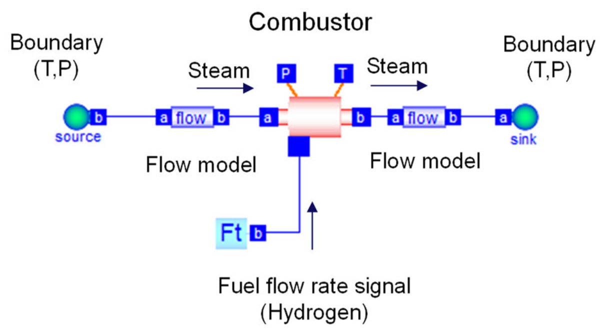


Figure 4. Diagram for calculations of combustor model.

2.2.2 Steam Compressor

The steam compressor is also modeled in the static mode. The specific enthalpy at the turbine outlet and power output are respectively given by

$$H_\mathrm{o} = H_\mathrm{i} + \frac{H_\mathrm{o,ad} - H_\mathrm{i}}{\eta_\mathrm{cp}} \tag{4}$$

$$W_\mathrm{cp} = F_\mathrm{cp}(H_\mathrm{o} - H_\mathrm{i}) \tag{5}$$

As the design of the steam compressor is not complete, a simple functional relation between the adiabatic efficiency and mass flow rate is assumed. The compressor flow is approximated based on the appropriate compressor map in terms of pressure ratio, rotation speed, and opening degree of the inlet guide vane.

2.2.3 Steam Turbine

HTTs and LPTs are described by static models. The enthalpy at the turbine outlet and power output are respectively given by

$$H_\mathrm{o} = H_\mathrm{i} - \eta_\mathrm{st}(H_\mathrm{i} - H_\mathrm{o,ad}) \tag{6}$$

$$W_\mathrm{st} = F_\mathrm{st}(H_\mathrm{o} - H_\mathrm{i}) \tag{7}$$

As the steam turbine is not completely designed, a simple functional relation between the adiabatic efficiency and mass flow rate is assumed. The steam flow rate is calculated using the nozzle flow equation:

$$G_\mathrm{st} = \begin{cases} A\sqrt{2\dfrac{k}{k-1}\dfrac{P_\mathrm{i}}{v_\mathrm{i}}\left[\left(\dfrac{P_\mathrm{o}}{P_\mathrm{i}}\right)^{\frac{2}{k}} - \left(\dfrac{P_\mathrm{o}}{P_\mathrm{i}}\right)^{\frac{k+1}{k}}\right]}, & \left(\dfrac{P_\mathrm{o}}{P_\mathrm{i}} \geq \left(\dfrac{2}{k+1}\right)^{\frac{k}{k-1}}\right) \\ A\sqrt{k\left(\dfrac{2}{k+1}\right)^{\frac{k+1}{k-1}}\dfrac{P_\mathrm{i}}{v_\mathrm{i}}}, & \left(\dfrac{P_\mathrm{o}}{P_\mathrm{i}} \leq \left(\dfrac{2}{k+1}\right)^{\frac{k}{k-1}}\right) \end{cases} \tag{8}$$

2.2.4 Heat Exchanger

For the superheater and economizer, finite volume models divided along the vertical direction are used, as illustrated in Figure 5. The effects of mass and energy conservation laws are given by Eqs. (9)-(13), and the heat transfer equations are given by Eqs. (14) and (15).

$$\frac{d}{dt}M_{\mathrm{h},j} = F_{\mathrm{h},j-1} - F_{\mathrm{h},j} \tag{9}$$

$$\frac{d}{dt}M_{\mathrm{c},j} = F_{\mathrm{c},j+1} - F_{\mathrm{c},j} \tag{10}$$

$$\frac{d}{dt}(M_{\mathrm{h},j}H_{\mathrm{h},j}) = F_{\mathrm{h},j-1}(H_{\mathrm{h},j-1} - H_{\mathrm{h},j}) - Q_{\mathrm{h},j} \tag{11}$$

$$\frac{d}{dt}(M_{\mathrm{c},j}H_{\mathrm{c},j}) = F_{\mathrm{c},j+1}(H_{\mathrm{c},j+1} - H_{\mathrm{c},j}) + Q_{\mathrm{c},j} \tag{12}$$

$$cp_{\mathrm{m},j}M_{\mathrm{m},j}\frac{d}{dt}T_{\mathrm{m},j} = Q_{\mathrm{h},j} - Q_{c,j} \tag{13}$$

$$Q_{\mathrm{h},j} = AK_{\mathrm{h},j}(T_{\mathrm{h},j} - T_{\mathrm{m},j}) \tag{14}$$

$$Q_{\mathrm{c},j} = AK_{\mathrm{c},j}(T_{\mathrm{m},j} - T_{\mathrm{c},j}) \tag{15}$$

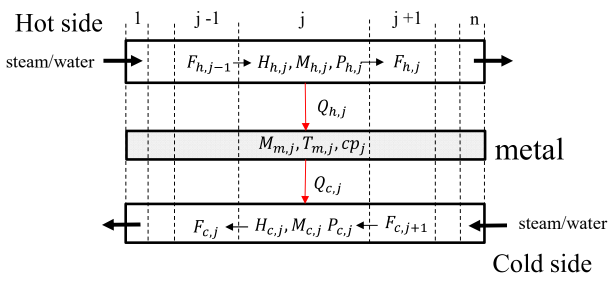


Figure 5. Schematic of finite element heat exchanger model.

The evaporator is modeled as a finite volume model, while the drum is modeled as a pressure vessel element satisfying the two-phase condition. Level control is considered to maintain a constant water level in the drum by adjusting a water supply valve. The heat transfer coefficient is also considered along with the variations at partial load with respect to mass flows.

2.2.5 Condenser

The condenser is modeled as a pressure vessel, where water and steam are mixed in a separated state. Dry or wet vapor enters the inlet and is discharged into water at the outlet. Again, level control is introduced to maintain a constant water level in the condenser and adjust the amount of condensate water by adjusting a drain valve.

To calculate the properties of water and steam in the model, we use the original function model created based on the IAPSW-IF97 formulation (Fernandez-Prini and

Dooley, 1997). The equipment performance parameters are configured based on heat balance and computed results. The weights of the heat transfer metal surface of the heat exchanger and piping volume can also be configured.

2.3 Load Change Strategy

In this study, the fuel flow rate was controlled based on the difference between the desired and computed power outputs, as illustrated in Figure 6. Other possible operating variables included the compressor inlet guide vane and steam governor valves, which were assumed to operate at a constant point.

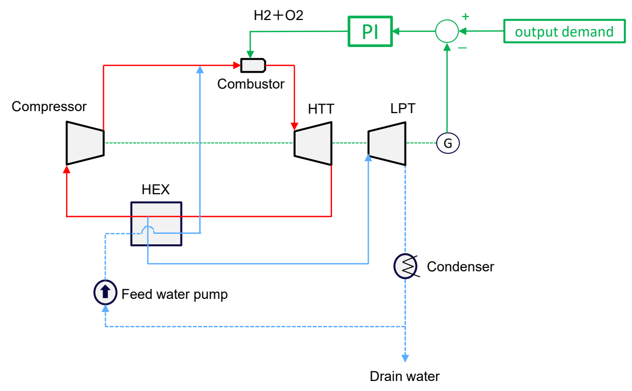


Figure 6. Diagram of load change strategy.

3 Results and Discussion

The system characteristics were examined when the load dropped and increased between 100% and 50%. The simulation input was a load command, and the load change rate was set to 10% per minute. This scenario was set up according to the operational conditions referred to existing and future power generation systems using GTs with combined cycles.

3.1 Load Reduction

Figure 7 shows the simulation results for load reduction. The values for power in figure 7(a) are given relative to the rated power output, and the values in the other figures are given relative to the value of the rated power output. The HTT output followed the load command, while the LPT output changed with a delay attributable to the delayed response of the steam caused by the heat capacity of the heat exchanger. The CP power also changed with a delay for the above reason. The thermal efficiency was lower at partial load than at the rated output. In addition, the fuel flow rate to the GT followed the load command, thereby reducing the combustion temperature, while the turbine outlet temperature showed large fluctuations, and it took time for the system state in the cycle to stabilize. The compressor inlet temperature and pressure also fluctuated, but the compressor inlet state showed less overshooting and fluctuations than the HTT outlet state. This is attributable to both the heat capacity in the heat exchanger and suppression of the effect of short-time fluctuations.

3.2 Load Increase

As shown in Figure 8, the HTT output followed the load command quickly as the fuel flow rate to the GT increased during load increase. On the other hand, the LPT output and CP power changed with a delay, like in the case of load reduction. In addition, the system state at the turbine outlet and compressor inlet fluctuated, and the steam state in the cycle took time to stabilize. Countermeasures (e.g., installing sprays) should be considered to prevent overshooting in parameters such as the temperatures of the combustor inlet steam and HTT outlet steam.

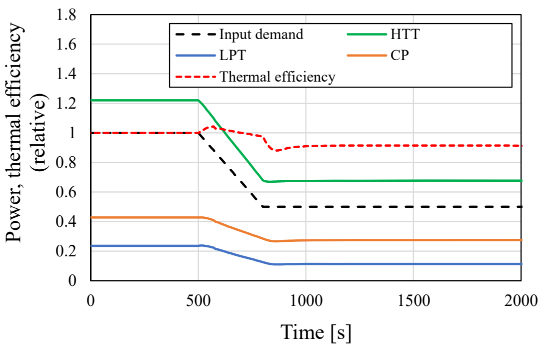
3.3 Problems and Countermeasures for Load Changes

The load change results indicate some sites where the system state considerably fluctuates using this control strategy. Such fluctuations can destabilize plant conditions and increase the heat load on equipment. Therefore, they should be minimized for stable operation through countermeasures taken by changing component specifications and control methods.

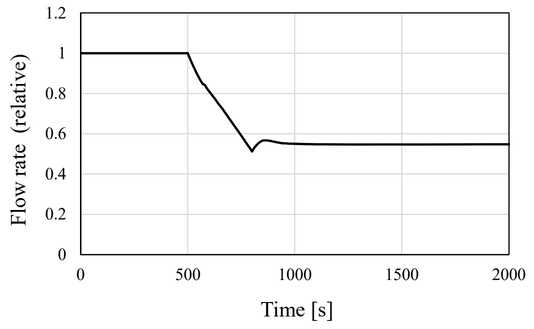
To evaluate the effect of the HRSG heat capacity on the system response, simulation results of load following when the heat capacity was set to half of the original were obtained, as shown in Figure 9. The undershoot width of the HTT outlet temperature reduced by decreasing the HRSG heat capacity. This can be attributed to the ratio of the delay in the LPT output being compensated by the reduced GT output, thereby mitigating the change in the fuel flow rate supplied to the GT. Thus, the HRSG specifications are important for system stability under changing load. In addition, other control methods should be considered to reduce fluctuations in the compressor inlet and HTT outlet for stable operation.

4 Conclusion

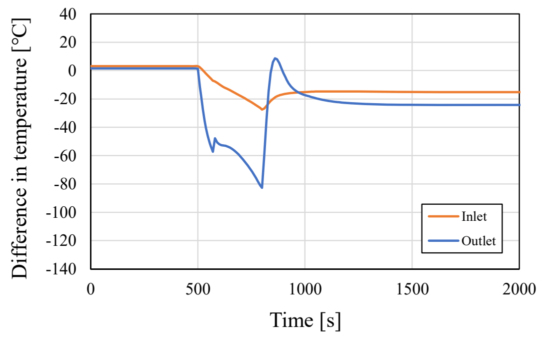
We established a dynamic model for a power generation system based on oxygen–hydrogen combustion turbines to evaluate its performance and dynamic behavior based on load following control. This study was preliminary and included various assumptions. In future work, we will improve the accuracy of the analytical model by considering the detailed designs of the system components. In addition, we will evaluate other types of load changes.



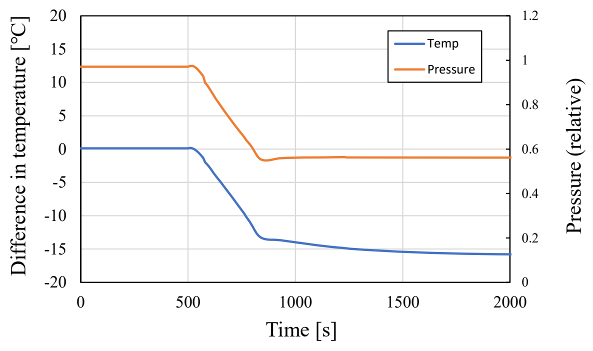
(a) Power output and thermal efficiency



(b) Fuel flow rate

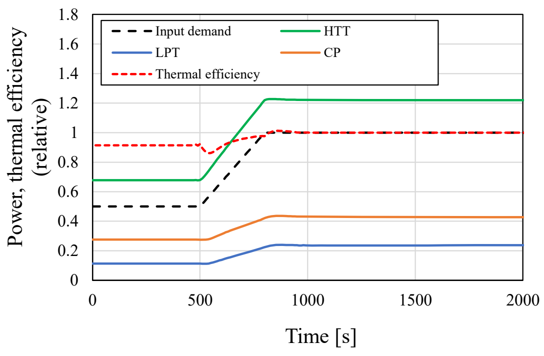


(c) HTT inlet and outlet temperature

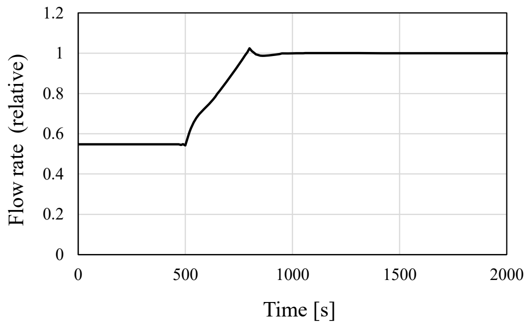


(d) Compressor inlet temperature and pressure

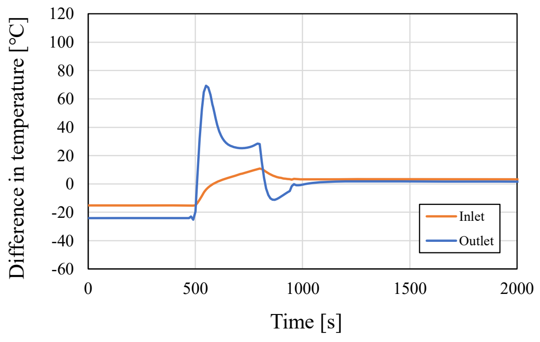
Figure 7. Simulation results during load reduction.



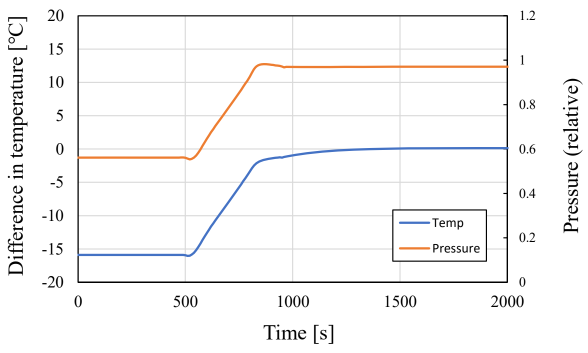
(a) Power output and thermal efficiency



(b) Fuel flow rate

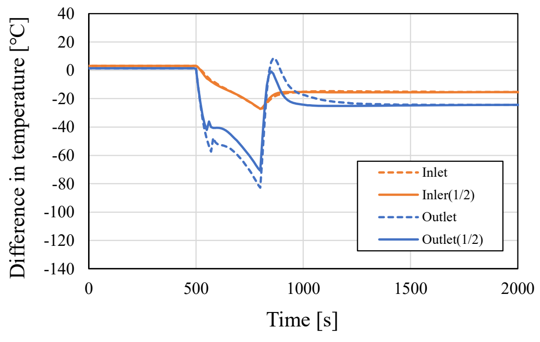


(c) HTT inlet and outlet temperature

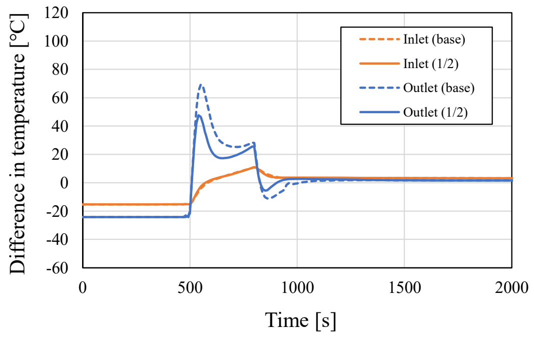


(d) Compressor inlet temperature and pressure

Figure 8. Simulation results during load increase.



(a) Load reduction



(b) Load increase

Figure 9. Simulation results of HTT inlet and outlet condition

Acknowledgements

This study is based on results obtained from a project, JPNP14021, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

R. L. Bannister, R. A. Newby, and W. C. Yang. Development of a hydrogen-fueled combustion turbine. Journal of Engineering for Gas Turbines and Power, 120(2):276–283, 1998. doi: 10.1115/1.2818116.

ETN. Hydrogen gas turbines. https://etn.global/wp-content/uploads/2020/02/ETN-Hydrogen-Gas-Turbines-report.pdf (accessed: September 5, 2022).

R. Fernandez-Prini and R. B. Dooley. Release on the IAPWS industrial formulation 1997 for the thermodynamic properties of water and steam. The International Association for the Properties Water and Steam, Erlangen, Germany, 1997.

Masafumi Fukuda and Yoshikazu Dozono. Double reheat Rankine cycle for hydrogen-combustion, turbine power plants. Journal of Propulsion and Power, 16(4):562–567, 2000. doi: 10.2514/2.5639.

T. Funatsu, M. Fukuda, and Y. Dohzono. Start up analysis of a H2-O2 fired gas turbine cycle. Proceedings of the ASME 1997 International Gas Turbine and Aeroengine Congress and Exhibition, 97-GT-491, 1997. doi: 10.1115/97-GT-491.

IEA. Net Zero by 2050. https://www.iea.org/reports/net-zero-by-2050 (accessed: September 5, 2022).

Jaroslaw Millewski. Hydrogen utilization by steam turbine cycles. Journal of Power Technologies, 95(4):258–264, 2015.

Chiba Mitsugi, Arai Harumi, and Fukuda Kenzo. WE-NET: Japanese hydrogen program. International Journal of Hydrogen Energy, 23(3):159–165, 1998. doi: 10.1016/S0360-3199(97)00042-6.

NEDO. Advancement of hydrogen technologies and utilizati on project. https://www.nedo.go.jp/english/activities/activi ties_ZZJP_100068.html (accessed September 5, 2022).

Wolfgang Sanz, Martin Braun, Herbert Jericha, and Max F. P latzer. Adapting the zero-emission Graz Cycle for hydroge n combustion and investigation of its part load behavior. In ternational Journal of Hydrogen Energy, 43(11):5737–574 6, 2018. doi: 10.1016/j.ijhydene.2018.01.162.

Bram Schouten and Sikke Klein. The optimization of hydrogen oxygen cycles. Proceedings of ASME Turbo Expo 2020, GT2020-14592, 2020. doi: 10.1115/GT2020-14592.

Mohammed G. Soufi, Terushige Fujii, Katsumi Sugimoto, and Hitoshi Asano. A new Rankine cycle for hydrogen-fired power generation plants and its exergetic efficiency. International Journal of Exergy, 1(1):29–46, 2004. doi: 10.1504/IJEX.2004.004732.

H. Sugisita, H. Mori, and K. Uematsu. A study of thermodynamic cycle and system configurations of hydrogen combustion turbines. International Journal of Hydrogen Energy, 23(8):705–712, 1998. doi: 10.1016/S0360-3199(97)00081-5.

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. doi: 10.1115/GT2017-64699.

Techno-economic assessment of an industrial project towards carbon neutrality

Arne Köppen[1] Fredrik Magnusson[2] Stéphane Velut[2]
Rui Gao[1] Koji Moriyama[3] Robert Uyeki[3]

[1]Modelon KK, Japan, {arne.koppen,rui.gao}@modelon.com
[2]Modelon AB, Sweden, stephane.velut@modelon.com
[3]American Honda Motor Co., Inc., USA, kmoriyama@na.honda.com

Abstract

This paper describes the application of the system simulation platform Modelon Impact for techno-economical assessment of energy projects towards carbon neutrality. The control co-design approach applied in the work allows for rapid assessment of various technology options without the need for deriving complex control laws for the considered assets. The approach is here applied on an industrial use case where the goal is to identify the technology options that minimize the total cost of ownership while achieving carbon neutrality.

Keywords: process design, controls, optimization, hybrid energy systems, renewables, hydrogen, energy storage systems

1 Introduction

Organizations have made a goal of significantly reducing CO_2 emissions to mitigate climate change. Not only governments and energy companies but also industries are now moving away from fossil fuels, investing in new technologies towards carbon neutrality. There are however many options, ranging from diverse renewable energy sources to carbon capture and utilization, hydrogen technologies and storage energy systems. The different alternatives vary significantly in terms of technical performance (efficiency, degradation, longevity) and in terms of economy (initial investment, operational cost, incentives). It is therefore not an easy task for the decision makers to efficiently explore the different paths and choose a cost-efficient implementation that meets the environmental targets (Venkatraman and Khaitan, 2015).

As shown in (Windahl *et al*, 2019) and (Fathima and Palanisamy, 2015), system simulation and optimization can efficiently help exploring and pruning the various options in a systematic way. The specificity of the approach proposed by the authors in (Velut *et al*, 2020) is twofold. It relies on the open modeling language Modelica and makes it thereby possible to model and simulate potentially any hybrid energy system. Secondly, the models can be used to formulate and solved dynamic optimization problems avoiding the need to derive & implement complex controllers for all considered configurations. Instead, optimal control and design problems can be setup to quickly assess the limits of performance and the cost of various alternatives.

The strategy has been applied in (Velut *et al*, 2020) for microgrid design and operation. In (Magnusson *et al*, 2021), the models have been further extended to include hydrogen components such as electrolyzer, fuel cell or hydrogen tanks. The current paper presents a major extension of the framework that is now able to assess the technical and economic feasibility of complex long term energy projects involving the production, conversion or supply of products like heat, electric power, hydrogen, synthetic methane or CO_2.

2 Framework

2.1 Tools and methods

Modelon Impact (Modelon, 2022) is used to model, simulate, and optimize the hybrid energy system in Modelica. Modelon Impact is a system modeling and simulation platform leveraging the benefits of web and open standard technologies. With openness at its core, Modelon Impact supports standards such as Modelica, FMI, Python and REST. The user-friendly browser interface provides modeling experts the tools they need to create, simulate, and experiment. The Modelon Impact API enables scripting of advanced analyses using Python through Jupyter notebooks. The optimization problem formulation has been written in

Optimica, a Modelica language extension (Åkesson, 2008). The API of Modelon Impact gives access to the Optimica Compiler Toolkit and its dynamic optimization framework (Magnusson *et al*, 2015), which is used to solve the dynamic optimization problem using direct collocation.

2.2 Physical modeling

A sketch of the system to be modeled and optimized is shown in Figure 1. The sketch represents a Honda-owned factory in the US that assembles cars.

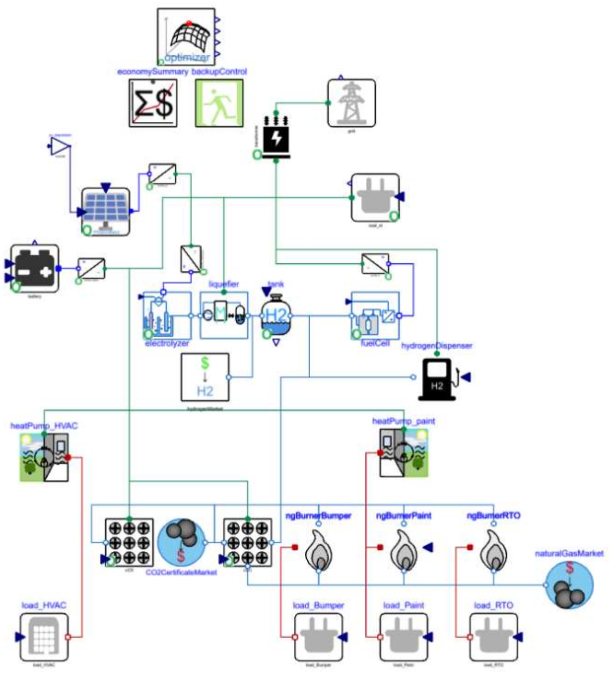


Figure 1 Overview of the system model

The plant model has been built by connecting component models from the Microgrid package in Thermal Power Library (Modelon, 2022). The Modelica package contains optimization-friendly models targeting optimal design and control. The models are typically static, semi-empirical and described by efficiency curves. Dynamics is mainly present in the storage components.

The plant consists of buildings, vehicle fleets, industrial equipment that can all be seen as some type of load:

- Electric loads for buildings' HVAC system, lighting, EV fleet, etc.
- Hydrogen loads for fuel cell-based transportation and logistics systems (forklifts, trucks, railroad)
- Thermal energy loads for heating and cooling in the buildings as well as industrial equipment such as burners

The goal of this work has been to assess different carbon neutral and sustainable options to satisfy the various loads. The model shown in Figure 1 represents a possible configuration where hydrogen technologies have replaced fossil fuel ones:

- Electrolyzer for onsite hydrogen production
- Hydrogen tank (for either gas or liquid hydrogen)
- Liquefaction plant
- Stationary fuel cell for power back-up or peak shaving
- Hydrogen dispenser

Compared with the work presented in (Magnusson *et al*, 2021), additional models have been derived:

- Heat pump, converting electric power to heat in buildings
- Burners to produce heat from the combustion of methane
- Carbon capture technologies to achieve carbon neutrality
  - A CCS block to capture the CO_2 produced in the combustion processes
  - A CCUS block for methane production from captured CO_2 and hydrogen
  - If CC(U)S is not applied, CO_2 is released to the ambient at a cost given by carbon taxes

Electric power, hydrogen and other fuels can also be imported using

- A power grid component acting as a voltage source
- A discrete delivery hydrogen market component that implements a controllable supply at a fixed frequency and the amount for every delivery being a degree of freedom in the optimization.
- A continuous fuel delivery, which can deliver methane (or other fuels) on demand, in case of pipeline delivery infrastructure.

Finally, energy can be stored either in batteries or as hydrogen to shave the power peaks and cope with the variations in renewable power. The goal is to assess the most economical alternative.

2.3 Economy modeling

2.3.1 Fixed and variable costs

The main goal of the work has been to perform a techno-economic analysis of the future plant. Economy information has therefore been added to every component to keep track of both the capital and operational cost of all equipment.

The considered time horizon for the optimization is the project lifetime T_{proj} , close to the components' lifespan.

The total cost of ownership of a component i has been divided into 2 parts:

- the fixed total cost $TCO_{fix,i}$, the sum of the capital cost and the fixed operational cost, i.e., due to maintenance
- the operational cost

If the fixed costs (capital cost and fixed operational cost) scale linearly with the assets size, the fixed cost can be computed using the following parameters:

- Lifetime L_i
- The specific capital expenditure CapEx C_i, i.e. normalized by the rated output or size $s_{opt,i}$
- The fixed yearly operational expenditure $o_{fix,i}$ (e.g. fixed maintenance cost) also normalized by the rated output or size $s_{opt,i}$

The fixed total cost component for component i over the project lifetime is then

$$TCO_{fix,i} = T_{proj} \cdot s_{opt,i} \left(\frac{C_i}{L_i} + o_{fix,i} \right)$$

The variable OpEx based on the usage of a component is typically computed by integrating the resource cost (power, fuel cost…) $o_{var,i}$over time. Since this system has centralized energy markets, the variable OpEx is typically calculated on the respective markets. The project total cost of ownership of the system can be finally computed as:

$$TCO_{tot} = \sum \left[T_{proj} s_{opt,i} \left(\frac{C_i}{L_i} + o_{fix,i} \right) + \int_0^{T_{proj}} o_{var,i} dt \right]$$

The summation over the components is automatically done by aggregating all the costs in a single "Economy Summary", which makes it convenient and compact. Any system configuration change does not require any manual update in the overall cost computation.

Although time-varying grid prices can be considered in the optimization, a fixed price was used to describe a virtual power purchase agreement.

2.3.2 Long term aspects: degradation and money value

While the data profiles used in the optimization only cover one year, to account for seasonal variations, the project lifetime is several decades. When considering such long periods, degradation of components as well as changes to the value of money need to be considered as well. Since the computational cost of dynamic grows superlinearly with the time horizon, optimization over the entire time horizon is computationally tractable. Two simplifications have been considered to account for these factors:

1. **Year separation**: By fixing the design and disregarding the storage between years, the TCO of each year can be calculated independently. By sampling the entire design space, the optimal design and control can be found.
2. **Mean year**: Lump degradation and NPV factors across all years to create a "Mean" year that allows simultaneous optimal design and control. This reduces the time horizon of the project optimization to a single year.

While the first approach is more accurate, it is still computationally expensive. It has been used to verify that the second approach yields satisfactory accuracy.

2.3.3 Demand charges and peak shaving

The objective of this model incorporates many aspects from previous projects that have been introduced and described in detail in (Velut *et al*, 2020) and (Magnusson *et al*, 2021). Similar to previous models, a form of peak shaving was being implemented. In this case, the utilities contract applied a demand charge on the maximum of the power demand from the grid during the summer months from June to September. The grid model has therefore been adapted to optimize the peak only during these months, so that both size and operation of relevant components results in an economically optimal peak shaving behavior.

The demand charge is a constant per-kW charge applied to the electricity grid and is being handled by use of a slack variable.

3 Optimization problem

As previously mentioned, the goal is to assess the technical and economic feasibility of the project of making the Honda-owned facility carbon neutral. For this purpose, some clean replacement technologies listed in Section 2.2 have been considered to meet the various needs in heat and power. Optimization will now be used to find the best options and size the equipment appropriately.

3.1 Objective function

The objective function is the total cost of ownership computed over the project lifetime T_{proj} as described in Section 2.3.1.

3.2 Discretization

The optimization problem is solved over a full calendar year. The sampling rate of the collocation algorithm is one hour (equal to the boundary conditions' sampling rate), which means that every control trajectory is described by 8760 degrees of freedom.

As the dynamics of the plant is relatively simple (integrators of the storage components), implicit Euler has been chosen in the collocation.

3.3 Co-design - simultaneous control and process design

The plant consists of many assets that interact with each other and some need to be controlled such as battery, electrolyzer, fuel cell. Instead of developing controllers for all these components, a control co-design approach has been chosen, where the optimizer operates the controllable assets at the same time as it sizes all equipment. If the design is solved for a specific control strategy, it leads to sub-optimal design, i.e. a higher total cost of ownership. By solving simultaneously for the assets' operation and their size, it is possible to minimize the asset's size and the overall cost. The strategy makes it also very convenient as no time is spent on deriving empirical controllers for all considered technologies and system configuration.

3.4 Degrees of freedom

The degrees of freedom in the co-design optimization problem are:

1. Parameters that define the size of the equipment (design problem)
2. Time trajectories for all control inputs of the controllable assets (control problem)

The list of degrees of freedom can be found in the tables below. Note that the discrete deliveries of hydrogen *hydrogenMarket.deliveries[i]* have been implemented as a vector of deliveries, the period being fixed to a week. All control signals have been normalized to operate between 0 (no output) to 1 (rated output).

With the chosen discretization level, the optimization problem contains **78905** degrees of freedom, 65 for the parameters and 78840 for the input trajectories (9 inputs and 8760 hourly values).

3.5 Constraints

Apart from the equality constraint of fulfilling the plant model equations, several inequality constraints have been considered in the formulation.

The first set concerns all degrees of freedom that need to lie within given bounds as shown in the previous section. Another set concerns the storage components (battery and tank) whose state of charge, eventually in terms of pressure or level, must be kept within reasonable limits. Export to the power grid was also prevented using an inequality constraint.

The last operational constraint that has been implemented ensures that power can be supplied in case of a full blackout. No black-out scenario is considered in our optimization and therefore we need to design and operate the system in a way energy is always available to handle that event. Since several backup solutions exist in the system, even a mix of technologies is conceivable. In this system, the backup power can be provided by either battery or fuel cell, i.e. the sum of rated output power P_{backup} should be greater than the minimum requirement for emergency power P^0_{backup}

$$P_{backup} \geq P^0_{backup}$$
$$P_{backup} = \Sigma P^i_{backupProvider} = P_{batt} + P_{fc}$$

This power needs to be available for a certain time t^0_{backup}:

$$t_{backup}(t) \geq t^0_{backup} \quad \forall t$$

The back-up time is expressed in terms of the total energy stored in the back-up providers and the power level it needs to be provided at:

$$t_{backup}(t) = \frac{\Sigma\left(\mathrm{r}^{\mathrm{i}} * E^i_{store}(t)\right)}{\mathrm{P}^0_{\mathrm{backup}}} > \mathrm{t}^0_{\mathrm{backup}}$$

Where r^i is a factor that accounts for the efficiency of the discharging process and the discharging power of the back-up assets. The constraint formulation has been generalized to allow for easy integration of alternative backup power solutions in case both battery and fuel cell turn out to be unfeasible.

3.6 Initialization

The solver for the nonlinear program needs reasonable initial guess of the solution for reliable convergence. This is typically generated by a dynamic simulation of the plant model where initial guesses for all degrees of freedom have been applied.

While the initial component size was typically constrained by simple physical considerations (available area for photovoltaics, yearly total energy demand, etc), the control signals' initial trajectories were derived using simple control laws. The battery charging and discharging rates were controlled by a PI controller driven by the renewable energy surplus. The electrolyzer was controlled using a PI controller to maintain a constant state of charge in the hydrogen tank. Concerning the other components, pre-defined trajectories were applied as initial guess.

4 Results

This optimization problem solves many tasks at the same time:

- It selects technology options (discrete choices)
- It sizes components (continuous choices)
- It operates assets to achieve the minimal total cost of ownership (continuous choices)
- It estimates the minimal total cost of ownership

In this paper, we review the results of a single optimization run. It is required in the future to perform a sensitivity analysis to assess the robustness of the optimization results with respect to uncertainties in all forecasted data (prices, loads and weather).

4.1 Technology selection

In the plant configuration shown in Figure 1, all technology options to be assessed have been modeled. This means that there are redundancies in the way the loads can be met. If the optimization results in an asset of size zero, it means that the corresponding technology is neither technically nor economically viable. In some cases, the optimization finds it optimal to fulfill a need by investing in different technologies. Here are the technology options that have been investigated:

1. *Import versus on-site generation* for hydrogen and power
2. *Fuel cell versus battery* for backup-power and peak shaving
3. *CCS vs. CCUS vs. carbon tax* to deal with the emissions from the combustion processes
4. *Conventional burner versus heat pump* for paint drying

In the following sections, the technologies selected for the first 2 items will be presented and discussed.

4.1.1 Import versus onsite generation

With the given price structure for energy and the given CapEx and OpEx for electrolyzer and photovoltaic power plant, both hydrogen and electricity can economically be produced on-site to a significant amount. However, due to the different pricing of electricity in winter and summer, both technologies have been selected by the optimization:

1. While the CapEx cost for the electrolyzer and the required liquefier is rather high, and also the energy losses from well to wheel are considerable, electricity cost in this model is low enough to produce liquid hydrogen at a lower price than

Table 1 Optimizable parameters of the system

Parameter name	*Min Value*	*Max Value*	*Unit*
battery.capacity	0.01	2000	MWh
CCS.m_flow_rat	1e-5	1e5	kg/s
CCU.m_flow_rat	1e-5	1e5	kg/s
electrolyzer.n_cell	0.01	350	MW
fuelCell.n_cell	0.01	212	MW
grid.P_peak	0		MW
heatPump2.P_rat	1e-7	1e4	MW
hydrogenMarket.deliveries[i]	1e-7	1e4	MW
ngBurner2.P_rat	0.01	68.9	MW
photovoltaics.scale	0.1	1e06	m^3
tank.V	40	3e5	MW
transformer.P_max	0.01	2000	MWh

Table 2 Optimizable control signals of the system

Input name	*Lower bound*	*Upper bound*
I_electrolyzer__opt	0	1
I_fuelCell__opt	0	1
P_battery_charge__opt	0	1
P_battery_discharge__opt	0	1
P_processHeat__opt	0	1
m_flow_CCS__opt	0	1
m_flow_CCU__opt	0	1
ndot_H2_boiloff_vent	0	
pv_curtailment__opt	0	1

purchasing from an external supplier. In summer, the comparatively high peak demand charge means, that reducing the output of the electrolyzer is the most economical mode of operation. As a result, the optimizer determined that producing the hydrogen on-site and purchasing the hydrogen to almost equal parts is the most cost-effective option, see the Sankey diagram in Figure 2.

2. Photovoltaics cannot provide electric energy at a lower cost than the utility company with the available pricing structure and the given insolation profile for the plant location. But during peak-hours in the summer, photovoltaics can significantly reduce the demand charge, and drives the electrolyzer for a longer time at its rated power, decreasing the overall cost for hydrogen.

4.1.2 Fuel cell vs. battery

In this model, the fuel cell and the battery can fulfill similar tasks: both can provide power and lower the demand charge during peak hours. Both options can also both provide emergency power in case of a blackout in the grid.

The considered batteries are second-use and they come therefore with limited performance. Apart from a limitation on the usable SOC range, they cannot be fully discharged faster than 2 hours.

The optimization results show that the fuel cell is the far better alternative for both use cases. We attribute this to two main root causes:

1. While the price-per-kW of the battery is here lower than that of the fuel cell, the backup power time requirement t^0_{backup} is much larger than 2h, meaning we will need to install several times as much power as P^0_{backup} to achieve the required energy needs, which results in a battery that is more expensive than the fuel cell for this purpose.
2. Furthermore, the constraint on t^0_{backup} means, that the battery needs to have enough energy stored to provide P^0_{backup} for t^0_{backup} at all times. A battery that is just big enough to fulfill the requirements for backup power needs to be kept completely charged and cannot be used for other purposes like peak-shaving without violating the backup power requirement.

While the fuel cell size is the significant parameter for the backup *power*, the time $t_{backup}(t)$ this power can be provided is largely determined by the available hydrogen in the tank, meaning that the tank will need to always retain t^0_{backup} worth of hydrogen in the tank. In our model, hydrogen is delivered at fixed time intervals and the delivered amount is variable and optimal. At each delivery, the tank is filled just enough to have a sufficient backup of hydrogen in the tank before a new delivery arrives.

The investment cost (CapEx) for a fuel cell would be too high to justify its use as a peak-shaver. However, if the fuel cell doubles as emergency power provider, the operational cost make operation during peak hours economically viable. A detailed explanation of this behavior can be found in 4.3.

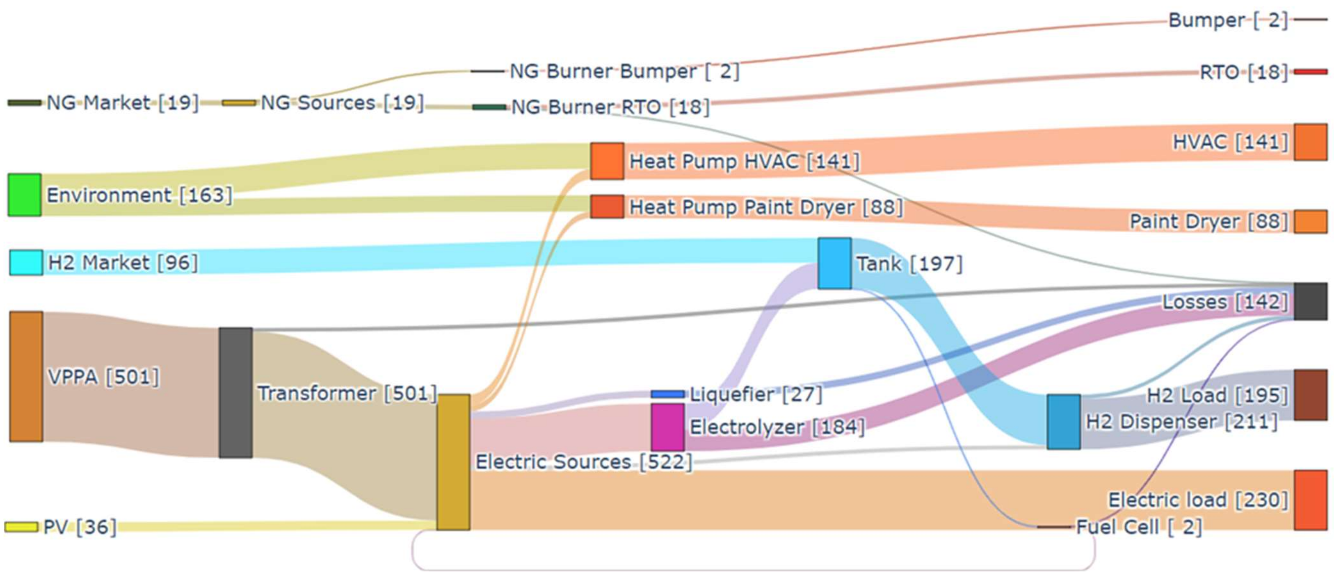


Figure 2 Sankey diagram of the energy flows [GWh] in the optimized system

Table 3 Optimized parameters

Parameter Name	***Optimal size***	***Min***	***Max***	***Unit***
battery.capacity	0.01	0.01	2000	MWh
cCS.m_flow_rat	1e-5	1e-5	1e5	kg/s
cCU.m_flow_rat	1e-5	1e-5	1e5	kg/s
electrolyzer.n_cell	22.6	0.01	350	MW
fuelCell.n_cell	3.1	0.01	212	MW
grid.P_peak	54.44	0		MW
heatPump2.P_rat	10.87	1e-7	1e4	MW
ngBurner2.P_rat	1e-7	1e-7	1e4	MW
photovoltaics.scale	22.26	0.01	68.9	MW
tank.V	1069	0.1	1e06	m^3
transformer.P_max	92.73	40	3e5	MW

The resulting fuel cell is just sized big enough to be the single provider of backup power P^0_{backup} Since the battery does not provide any backup power, the hydrogen tank's level is directly proportional to the available emergency backup time t_{backup} (Figure 4)

4.2 Components' size

In this system, a total of 11 parameters have been optimized, most of them are related to the rated output or physical size of these components. Additionally, 53 weekly hydrogen deliveries have been computed (Figure 3). The results of the optimal sized components is shown in Table 3.

The following is worth to be noted:

- The resulting photovoltaics is not covering all the available area for the reasons previously exposed
- The fuel cell is sized mainly to match the backup power requirements.
- The battery capacity is minimal, reaching practical zero size
- Carbon taxes are more economical than carbon capture technologies

4.3 Optimal control

Interesting findings can be done by visualizing the control trajectories.

The state of charge of the tank is limited by the back-up requirements and not by its bound parameter, see Figure 4. The back-up requirements have therefore a direct impact on the amount of the weekly hydrogen delivery and the tank size.

The power from the photovoltaics is in principle never curtailed because the maximum output of the

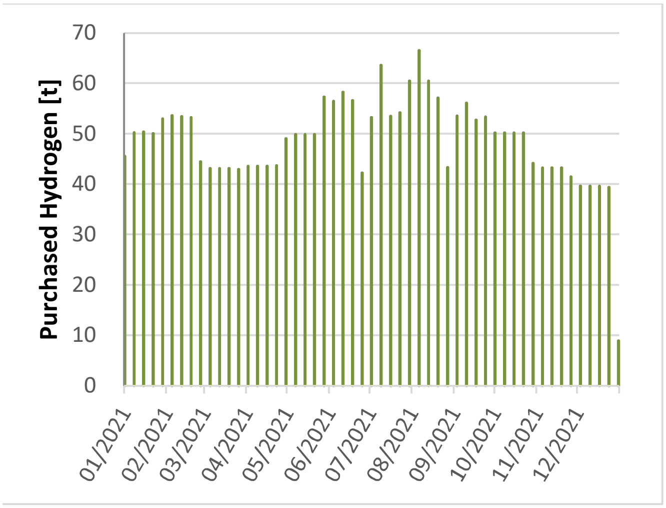


Figure 3 Optimal hydrogen deliveries over the course of a full year

photovoltaic power plant is below the rated power input of the electrolyzer (Table 3), which means that the optimizer is always able to utilize the surplus energy either for the electricity load directly or for hydrogen production.

Peak shaving in the summer was mainly expected to be performed by engaging storage. As shown in Figure 5, the optimizer found that it is more cost effective to act on the significant controllable load that is the electrolyzer. Hydrogen is imported to a higher degree in the summer month when this happens, see Figure 3.

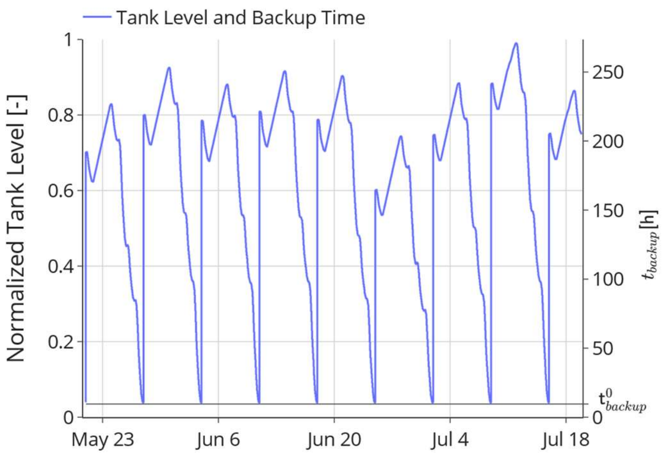


Figure 4 Backup reserve time t_{backup} and tank level (normalized)

The fuel cell is only used in 2 situations:

1. It takes over from the electrolyzer in its peak shaving activity when the electrolyzer power is completely shut down, see Figure 6. As explained in 4.1.2, due to its high CapEx, the fuel cell is sized to fulfill the backup requirements. Making thus up only roughly 15% of the electrolyzer's rated power demand, the impact on peak-shaving of the fuel cell is relatively small.
2. Besides its operation as peak shaver, the fuel cell converts the otherwise unused boil-off hydrogen into electricity. When demand for hydrogen becomes lower, the fraction of boil-off gas (BOG) at the tank outlet increases. During the weekends, the hydrogen-load is provided entirely as BOG (Figure 6). At the same time, the electrolyzer generates additional hydrogen, which results in a steady increase in the amount off boil-off hydrogen until the demand recovers during working hours at the plant.
   Reliquefying the hydrogen (zero boil-off) may be a more cost-effective option in such cases but is currently not supported in this model.

4.4 Performance

The optimization problem considered in the paper is complex and large-scale. Modelon's Thermal Power library, as well as the component models developed for this project, have been designed with dynamic optimization in mind. Thanks to a more efficient formulation and model improvements, it is now possible to solve the TCO optimization problem for a full year in a reasonable time. On an entry level PC (i3), initializing the problem takes about 5 minutes, with an additional 10 minutes to find a solution. Using parallelization, it is possible to run parameter sweeps in not more than 10 minutes.

5 Conclusion

In this paper, we have presented a framework that allows for the techno-economic assessment of complex hybrid energy projects. The benefit of the approach relies in the simultaneous design of the controls and the process, which lead to lower cost and a more systematic way of handling new configurations and technologies. The method has been applied on a car manufacturing plant to minimize the total cost of ownership of the transition towards carbon neutrality. The technique was able to estimate the overall cost and select the most viable technology options. S*ome* results are unexpected and cannot be found by considering a part of the system in isolation, but rather require a holistic system model.

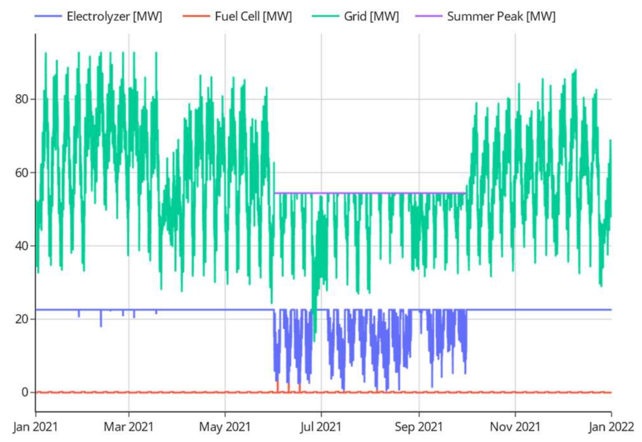


Figure 5 Demand reduction during summer months and peak shaving over the course of one year

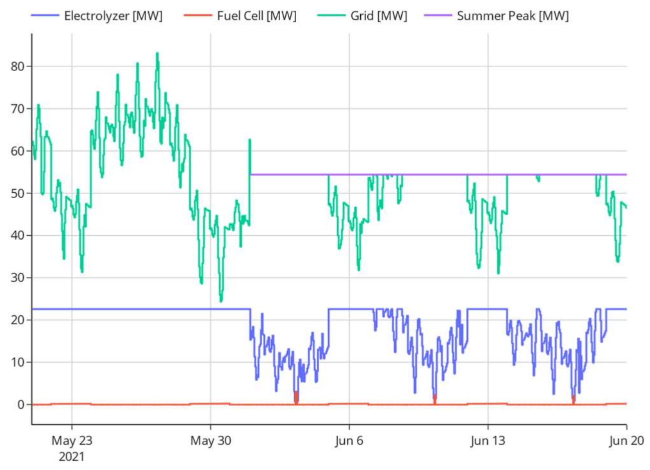


Figure 6 Demand reduction and peak shaving (detail)

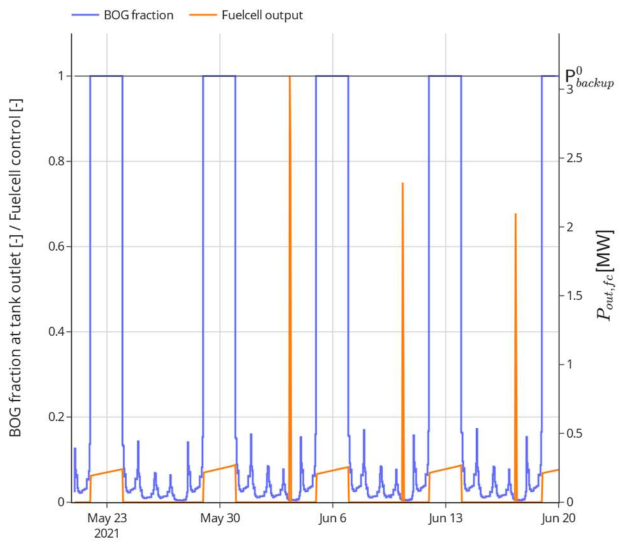


Figure 7 Boil-off consumption and peak-shaving

Abbreviations

EV	Electric Vehicle
HVAC	Heating, Ventilation, Air-Conditioning
TCO	Total Cost of Ownership
BOG	Boil-off gas (hydrogen)
CapEx	Capital Expenditure
OpEx	Operational Expenditure
FC	Fuel cell
PV	Photovoltaic
CCS	Carbon Capture and Storage
CCUS	Carbon Capture, Utilization and Storage
NPV	Net Present Value
FMI	Functional Mock-up Interface

References

Fathima A. and Palanisamy K. “Optimization in microgrids with hybrid energy systems – A review”. Renewable and Sustainable Energy Reviews, 45, 431-446, 2015

Magnusson F., Velut S., Åberg M. and Gao R. Optimal Design of Microgrids with Hydrogen Components Using Modelica. Proceedings of 1DCAE-MBD JSME symposium, December 2021, Tokyo, Japan

Åkesson, Johan. Optimica-An Extension of Modelica Supporting Dynamic Optimization, Proceedings of the 6th International Modelica Conference, Bielefeld, Germany, 2008

Magnusson, Fredrik, and Åkesson, Johan. "Dynamic optimization in JModelica.org." Processes, vol. 3, no. 2, 2015, pp. 471-496.

Velut S., Andreasson J., et al. ”A Modelica-based solution for the simulation and optimization of microgrids”, Proceedings of Asian Modelica Conference 2020, Tokyo, Japan, October 08-09, 2020

Windahl, Johan, et al. “Platform for Microgrid Design and Operation.” Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019, Linköping University Electronic Press, 1 Feb. 2019. Crossref, doi:10.3384/ecp19157405.

“Thermal Power Library: Modelica Library Built by Modelon.” Modelon, https://www.modelon.com/library/thermal-power-library/. Accessed on August 17, 2022.

“Modelon Impact: Turn simulation results into business decisions with confidence.” Modelon, https://modelon.com/modelon-impact/. Accessed on August 17, 2022.

• Ramakrishnan Venkatraman and Siddharta Kumar Khaitan. A Survey of Techniques for Designing and Managing Microgrids. IEEE Power & Energy Society General Meeting, 2015

Towards Modeling and Simulation of Dynamic Overconstrained Connectors in Modelica

John Tinnerholm[1] Francesco Casella[2] Adrian Pop[1]

[1]Department of Computer Science, Linköping University, Sweden, {john.tinnerholm, adrian.pop}@liu.se
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,
francesco.casella@polimi.it

Abstract

Cyber-Physical Systems are ever-increasing in complexity and new methods and tools for developing them are needed. To support these highly dynamic systems, increasing the flexibility of the modeling languages is desirable. This paper proposes and examines a Modelica language extension to support dynamic overconstrained graphs with reconfiguration at runtime. Two applications of this new feature are also discussed: synchronous AC power systems and incompressible fluid networks. Reported findings suggest that supporting dynamic overconstrained graphs might yield performance benefits and provide the possibility of simulating systems that can not currently be simulated in existing Modelica tools.

Keywords: dynamic overconstrained connection graph, runtime reconfiguration

1 Introduction & Motivation

Overconstrained connector semantics was introduced in 2004 in version 2.1 of the Modelica Language Specification (The Modelica Association, 2004). It allows to add non-flow variables on connectors that are dependent on each other, which can lead to overconstrained equation systems when loops are formed in the connection graph. It also makes topological information about the connection graph available to the modeller, via the `Connections.*()` operators.

Overconstrained connectors have found at least two notable applications so far. The first is in the MultiBody package of the Modelica Standard Library (Martin Otter, 2003), were such a feature allowed to design of the library in a truly object-oriented way compared to previous versions. The second one is in the PowerSystems library (Rüdiger Franke, 2014), where overconstrained connector variables are used to carry around a reference phase signal for efficient numerical simulation.

As of the current Modelica Language Specification (The Modelica Association, 2021), it is only possible to define static connection graphs, which can be processed at compile time. This makes the implementation of overconstrained connectors in a Modelica compiler rather straightforward. However, it introduces a significant limitation when modelling AC power systems using phasors. In these models, the phase (or frequency) reference is generated by one component of the synchronous system (an infinite bus or a large synchronous generator for islanded systems) and then distributed throughout the entire connected synchronous system by the overconstrained connector variables. In this context, it is possible to have multiple independent synchronous systems in the same Modelica model that correspond to structurally disconnected connections sub-graphs, e.g., two national grids such as Germany and Denmark connected by an undersea DC link; each statically connected synchronous network gets its reference phase or frequency from the root node of its connection graph. However, in this case, their topology is fixed at compile time and cannot change at runtime.

When modeling AC transmission systems, particularly large ones, it is possible that, in case of severe perturbations, some key circuit breakers are switched open, effectively splitting a single synchronous system into multiple independent synchronous islands, which can permanently rotate at different frequency. For example, when modelling the European ENTSO-E synchronous system, the Spanish grid can become isolated by opening a few line breakers on the French border. Note that modelling this scenario requires no structural changes in the grid equations; it just needs some numerical admittance values to be set to zero.

When this happens, the two (or more) ensuing islands can settle down into new steady states with different steady-state frequencies. Hence, if a single, whole-system-wide reference is still used, the phase angle of currents and voltages of the islands that do not contain the root node of the static connection graph will end up rotating permanently, with a frequency that is the difference between the local island frequency and the root node frequency. As a consequence, when the steady-state is reached, the phasors of the new island will continue to change sinusoidally. This is very inconvenient from a performance point of view because it prevents variable step-size solvers from increasing the step size, once the system settles into the new steady state. It also triggers very frequent recomputations of the system Jacobian if implicit stiff solvers are used.

This problem could be avoided by allowing to dynamically add or remove the unbreakable branches corresponding to `Connections.branch()` statements in the connection graph (of Section 9.4.1 in the Modelica Specification[1]), based on the status of the corresponding circuit breaker components, and to add or remove the equations that show up in the same if-equation branches where the `Connections.branch()` statements are declared. It would then be possible to break up the original synchronous connections established by transmission lines when their admittance is brought to zero, thus modelling the effects of circuit breakers on the synchronous subsystem topology.

As a consequence, two or more disjoint connection graphs would be formed at the time of the breaker openings, each corresponding to a new synchronous island. Therefore, the new graph topology should be analyzed at this point, picking a new root node for each newly formed island in the grid. Then, instead of having a single phase reference for the entire system (which is no longer adequate), one would now have two or more independent phase references, one for each island, which would ensure that the phasor variables of each island reach a steady state, thus avoiding the persistent sinusoidal oscillations found in the case of a statically determined connection topology.

In this paper, we investigate the effects of relaxing the constraints imposed on If-Equations (in Section 8.3.4 of the Modelica Specification[2]) by allowing a special If-equation construct where the `Connectors.branch` operator is allowed within these equations, thus making it possible to change the connection graph dynamically at runtime.

The applicability and usefulness of this concept are demonstrated in Section 2, using simple conceptual models in two different application domains: AC power systems and closed incompressible fluid networks. It is shown that in these two cases, the structural variability of the system of equations brought by the proposed extension is indeed very limited and can be handled by small extensions of existing Modelica compilers. A prototype implementation of this feature in the OpenModelica.jl Julia framework is presented in Section 3. Simulation results obtained with the prototype implementations are discussed in Section 4, while Section 5 concludes the paper with final remarks and suggestions for future work.

2 Dynamic Overconstrained Connectors in Modelica

A current limitation of Overconstrained Connectors in Modelica is that they cannot be used in If-Equations[3]. In this paper we relax this condition by allowing the `Connections.branch()` operator to occur within If-Equations, hence allowing conditional Connection operators.

To showcase this feature we developed an experimental package called *DynamicOverconstrainedConnectors*[4], containing three sub-packages. The first contains conceptual models of AC grids, using Complex types to represent phasors. The second contains the very same models, albeit with separate Real variables for the real and imaginary part - this is meant for experimental compiler frameworks that cannot handle operator records. Finally, the third contains conceptual models of incompressible fluid networks.

2.1 Use case: AC power systems

The main simplifying assumptions for this use case are:

- Purely inductive transmission lines.
- Idealized synchronous generators that impose a voltage at their port with fixed magnitude and a phase equal to the rotor angle.
- Droop-based primary frequency control of the generators.
- The reference frame for the phasors is rigidly connected to the rotor of the generator that is selected as the root node in the connection graph.

Listing 1 contains the definition of the overconstrained connectors; all quantity are in per-unit, to avoid any scaling issues.

Listing 2 shows two alternative implementations for the transmission line model with an embedded line breaker. The first is made possible by the current static connection graphs, whereby the unbreakable branches are always active, and the frequency reference is always the same at the two ports. The second uses the proposed extension, whereby the unbreakable branch is only active when the breakers are closed; so is the equality constraint between the phase reference on the two ports.

Listing 3 shows the conceptual synchronous generator model, which sets the overconstrained reference frequency variable to its own frequency if selected as the root node of the connection graph.

Listing 1. AC overconstrained connector.

```
type ReferenceAngularSpeed
  extends SI.PerUnit;
  function equalityConstraint
    input ReferenceAngularSpeed omega1;
    input ReferenceAngularSpeed omega2;
    output SI.PerUnit residue[0];
  end equalityConstraint;
end ReferenceAngularSpeed;
```

[1]https://specification.modelica.org/maint/3.5/connectors-and-connections.html#overconstrained-equation-operators-for-connection

[2]https://specification.modelica.org/maint/3.5/equations.html#if-equations

[3]https://specification.modelica.org/maint/3.5/connectors-and-connections.html#restrictions-of-connections-and-connectors

[4]https://github.com/looms-polimi/DynamicOverconstrainedConnectors

```
connector ACPort
  SI.ComplexPerUnit v(re(start = 1));
  flow SI.ComplexPerUnit i;
  ReferenceAngularSpeed omegaRef;
end ACPort;
```

Listing 2. AC Transmission line models.

```
partial model TransmissionLineBase
  parameter SI.PerUnit B = -5.0;
  discrete SI.PerUnit B_act;
  Boolean closed;
  Boolean open;
  Boolean close;
  ACPort port_a;
  ACPort port_b;
equation
  port_a.i = Complex(0,B_act)*
             (port_a.v - port_b.v);
  when open then
    closed = false;
    B_act = 0;
  elsewhen close then
    closed = true;
    B_act = B;
  end when;
  ...
end TransmissionLineBase;

model TransmissionLine
  extends TransmissionLineBase;
equation
  port_a.omegaRef = port_b.omegaRef;
  Connections.branch(port_a.omegaRef,
                     port_b.omegaRef);
end TransmissionLine;

model TransmissionLineVariableBranch
  extends TransmissionLineBase;
equation
  if closed then
    port_a.omegaRef = port_b.omegaRef;
    Connections.branch(port_a.omegaRef,
                       port_b.omegaRef);
  end if;
end TransmissionLineVariableBranch;
```

Listing 3. AC generator model.

```
model Generator
  parameter SI.PerUnit V = 1;
  parameter SI.Time Ta = 10;
  parameter SI.PerUnit droop = 0.05;
  parameter Integer p = 0;
  ACPort port;
  SI.PerUnit Ps = 1, Pc, Pe;
  SI.Angle theta(start=0, fixed = true);
  SI.PerUnit omega(start=1, fixed = true);
equation
  der(theta) =
    (omega - port.omegaRef)*omega_n;
  Ta*omega*der(omega) = Ps + Pc - Pe;
  port.v = CM.fromPolar(V, theta);
  Pe = -CM.real(port.v*CM.conj(port.i));
  Pc = -(omega-1)/droop;
  Connections.potentialRoot(
    port.omegaRef, p);
  if Connections.isRoot(port.omegaRef) then
    port.omegaRef = omega;
  end if;
end Generator;
```

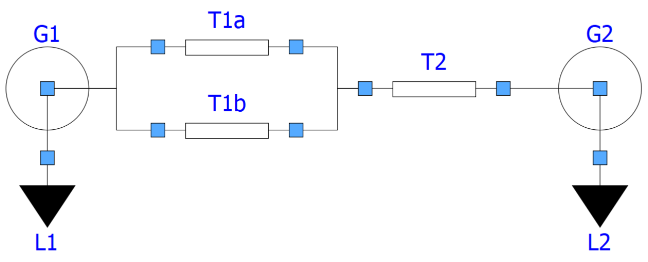


Figure 1. `System3` model diagram.

These components are used to build several test cases in the demonstration package; this paper focuses on the most relevant ones.

The base system model `System3` is built as shown in Fig. 1, using standard `TransmissionLine` components with static overconstrained connector semantics. Its connection graph is shown in Fig. 2, and it contains three unbreakable branches, seven connections, one broken connection, and one root node.

The system is initially fully connected, and undergoes an initial transient to get to its steady-state. At $t = 10$, the break of line `T2` is tripped open, so two synchronous island are formed, one containing `G1`, `L1`, `T1a`, `T1b`, and the left connector of `T2`, the other containing `G2`, `L2`, and the right connector of `T2`. The two islands settle to different steady-state frequency, but unfortunately there is only one reference frequency (set by `G1`), so the phasors of the right-hand-side island keep on rotating forever.

The next test model `System4` uses dynamic overconstrained `TransmissionLineVariableBranch` components instead. In this case, the connection diagram is initially the same as in Fig. 2, but after $t = 10$ it becomes as shown in Fig. 3: the deactivation of the unbreakable branch of `T2` splits the graph into two disconnected graphs, each with its own root node. From the point of view of the equation count, the additional equation brought in by the extra root node is balanced by the de-activated conditional equality equation of `T2`. In this way, also the phasors of the right-hand-side island eventually settle down to a constant value, because they are now referred to their proper reference, namely `G2.port.omegaRef`.

The test model `System6` is the same as `System4`, except that `T1a` is tripped open at $t = 10$, instead of `T2`. In this case the system remains fully connected, and a single root node can be used throughout the entire transient.

The model `System7`, shown in Fig. 4, demonstrates the use of root node priority. Two synchronous islands are formed when line `T2` is tripped open. The right-hand-side one contains two generators, `G2` and `G3`, which both contain potential root nodes. In this case, `G3` is selected as root node, since it has a higher priority than `G2`. This mecha-

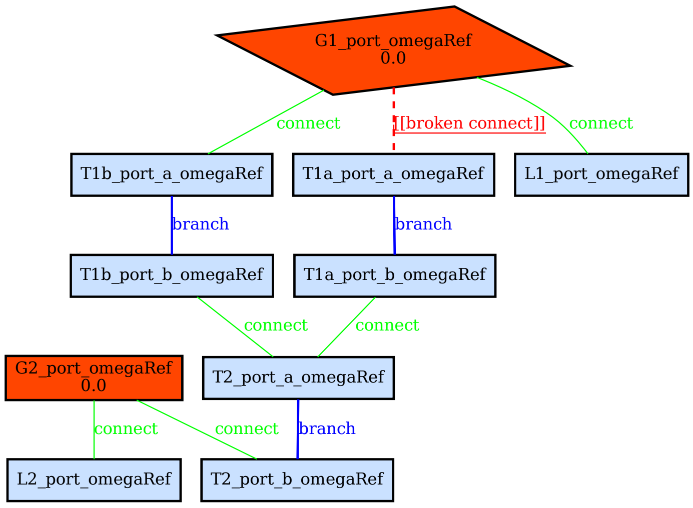


Figure 2. Static connection graph for `System3`.

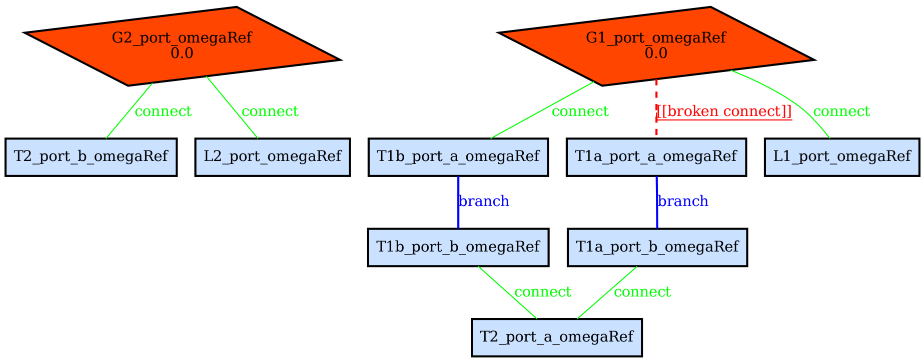


Figure 3. The connection graph for `System4` after $t = 10$.

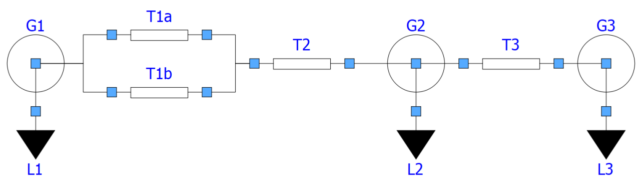


Figure 4. `System7` model diagram.

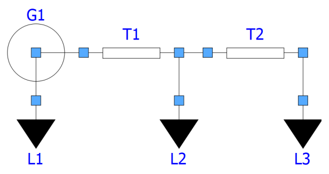


Figure 5. `System9` model diagram.

nism can be used to ensure that the larger generators are selected as reference nodes, by setting priorities correlated to the generator size.

Finally, another interesting modelling feature can be implemented with dynamic overconstrained connectors in synchronous AC system models. Consider the system shown in Fig. 5, with the three loads consuming 0.8, 0.1 and 0.1 p.u. power, respectively. Suppose that the total consumption threatens to cause a network breakdown, because the generator `G1` does not have enough power to supply it reliably. In such a case, the network operator can opt for some load shedding, i.e., it can trip open the line `T1`, preserving the service for the largest load `L1`, instead of risking a complete system blackout.

When this happens, `L2` and `L3` remain connected to an island without any power generation capability. Since load models normally prescribe a certain active and reactive consumption, the absence of any generation capacity in the island means that the system equations have no feasible solution, like in the equation $x^2 + 1 = 0$. As a consequence, the simulation aborts as soon as `T1` is tripped open, and it is not possible to continue the simulation of the left-hand-side island, even though it is perfectly functional and capable to carry on operating.

However, if the `TransmissionLineVariableBranch` model is employed for line `T1`, and the load model of Listing 4 is used, the simulation can be continued.

Listing 4. Extended load model.

```
model LoadVariableRoot
  extends LoadBase;
equation
  if port.omegaRef > 0 then
    port.v*CM.conj(port.i) = Complex(P,Q);
  elseif Connections.isRoot(port.omegaRef)
      then
    port.v = Complex(0);
  else
    port.i = Complex(0);
  end if;
  Connections.potentialRoot(port.omegaRef,
                             10000);
  if Connections.isRoot(port.omegaRef) then
    port.omegaRef = 0;
  end if;
end LoadVariableRoot;
```

As long as the load is connected to a connection graph that contains at least one generator, this will be selected as root node, because of the much higher priority, and it (greater than zero) frequency will show up as the overconstrained `omegaRef` variable; hence, the first branch of the If-equation will be active, setting the active and reactive power consumption to the given `P, Q` values.

However, if there are no generators in the dynamically formed island after the line tripping, then one load in the island will be selected as root node, and it will set both `omegaRef` and the port voltage to zero, thus conceptually connecting its port to ground. If the load finds itself in a generator-less island, which is characterized by a zero `omegaRef` value, but is not selected as root node, then its equations will set the absorbed current to zero. As a result, all voltages and all currents of the generator-less island will be computed to zero, describing a switched-off subnetwork, while allowing the simulation of the other island to continue undisturbed.

2.2 Use case: Closed incompressible fluid networks

Another use case for dynamic overconstrained connectors is incompressible fluid networks. This is demonstrated by the `IncompressibleFluid` sub-package. For the sake of brevity, only some short code fragments are reported in this paper; the reader is referred to the full Modelica code on GitHub for more details.

Any closed incompressible fluid systems, that is not connected to any pressure source of sink (e.g. the atmosphere, or a fixed pressure representing the supply point of a water supply system), needs to be connected to a component known as expansion tank or vessel. In real life the purpose of this component is to set the pressure level of the circuit, which would otherwise be floating freely, and also accommodate for the thermal expansion of the fluid without blowing the circuit up.

When modelling incompressible fluid systems, the thermal expansion effect is normally not explicitly included, because it is very well compensated by the presence of such expansion tanks and has a negligible effect on the actual flow rates, so the density of the fluid is assumed to be a constant. A very simple expansion tank model can then just set the pressure at its port to a fixed value; the entering flow will eventually turn out to be zero, due to the overall mass conservation of the closed circuit. This is demonstrated by the `System1` model in the `IncompressibleFluid` sub-package.

Consider now the `System2` fluid model, shown in Fig. 6. The system is closed and circulates a fluid in the left and right meshes, as well as through the two valves in case the pressure distribution is not fully symmetric.

As long as at least one of the two valves is open, the

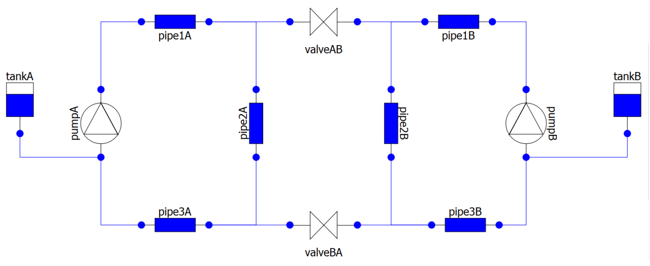


Figure 6. `System2` fluid system model diagram.

closed system is fully connected, so a single expansion vessel (e.g. `Tank1`) is required to make sure that all the pressures around the circuit are well-defined. However, when both valves are closed, i.e., their flow coefficients set to zero, the system is effectively split into two independent closed system. At that point, a second expansion tank is needed in the right-hand-side half of the circuit, to make sure that all the pressures there remain well defined.

Without that provision, the equations corresponding to that part of the system will have a unique solution concerning the mass flow rates, but infinitely many when looking at the pressures. Although some tools allow to manage such a situation by picking one of them and continuing the simulation, it remains a fact that the model in those conditions is not well-posed.

Note that, as in the case of AC networks, the splitting into independent sub-systems only depends on numerical values of some coefficients (line admittances there, flow coefficients here), not on structural changes of the system of equations such as, e.g. disabling some connections or some models in the system.

Dynamic overconstrained connectors allow solving this problem. The overdetermined connector can be defined as shown in Listing 5

Listing 5. A overconstrained fluid connector.

```
type CircuitIdentifier
  extends SI.PerUnit;
  function equalityConstraint
    input CircuitIdentifier id1;
    input CircuitIdentifier id2;
    output SI.PerUnit residue[0];
  end equalityConstraint;
end CircuitIdentifier;

connector FluidPort
  SI.Pressure p;
  flow SI.MassFlowRate w;
  CircuitIdentifier id;
end FluidPort;
```

In this case, there is no need to carry around any information throughout connected components, as in the previous case; the only thing that is needed for the modelling is the information about the dynamic connection graph topology. However, since the connection graph is always referred to some overconstrained connector variable, one possible choice is to define it as an Integer *connected circuit identifier*.

The `ValveDynamicBranch` model is analogous to the `TransmissionLine` model; in particular, it has a conditional activated `Connections.branch()` statement and a conditionally activated equation stating the equality of the ID on both connectors; both are only active when the valve is open, see Listing 6.

Listing 6. Valve model.

```
model ValveDynamicBranch
  extends BaseValve;
equation
  if closed then
    Connections.branch(inlet.id,outlet.id);
    inlet.id = outlet.id;
  end if;
end ValveDynamicBranch;
```

The `ExpansionTank` model is shown in Listing 7. If the tank is selected as root node, then it means the tank is the only component having such a property in the effectively connected circuit; in this case, it sets the port pressure to a fixed parameter value, and the overconstrained `id` connector variable to an ID parameter.

If instead, it is not selected as the root node, then it just acts as a plug, i.e., it sets the port flow rate to zero. This avoids getting inconsistent systems of equations, which would arise if two or more expansion tank components tried to set their port pressure in a connected circuit.

Listing 7. Valve model.

```
model ExpansionTank
  parameter SI.Pressure p0;
  parameter Integer id = 0;
  parameter Integer priority = 0;

  FluidPort inlet;
equation
  Connections.potentialRoot(inlet.id,
      priority);
  if Connections.isRoot(inlet.id) then
    inlet.p = p0;
    inlet.id = id;
  else
    inlet.w = 0;
  end if;
end ExpansionTank;
```

When this dynamic overconstrained component is used for the tanks, the model is always well-posed. Initially, when the valves are open, only one tank is selected as a root node and sets the pressure at its port, while the other tank behaves as a plug. As soon as both valves are closed, the connection graph is split into two disconnected graphs, each having its own root node tank. Therefore, each newly formed sub-circuit ends up with a tank setting its pressure level.

2.3 Outlook

The two presented modelling scenarios have two important factors in common. One is the need to identify effectively connected connection graphs, when some components that normally establish a branch in the graph actually do not do so in some cases, when parameters such as admittance or flow coefficients are brought to zero. The other is the need to set some value in the root node of the effectively connected sub-graph and then propagate it through the actually connected sub-network. Many other models in different domains could have the same requirement and, therefore, a similar structure.

As will be discussed in the next section, this modelling pattern, tackled with dynamic overconstrained connectors, leads to a very restricted structural variability of the corresponding equations, where the overconstrained connector variables are set by conditional equations that are activated when the variable is selected as a root node, and then propagated throughout the actually connected sub-network. However, this can be handled in terms of code generation and runtime code, without requiring general-purpose handling of structural variability, which is still an open problem for Modelica tools.

3 Implementation

A typical Modelica Compiler first translates a textual representation of a Modelica model into executable simulation code through a series of phases, and the semantics for overconstrained connectors in the Modelica language is handled statically during the preprocessing of a model before the generation of simulation code, see Figure 7.

We choose to implement our extension within *OpenModelica.jl* (Tinnerholm et al., 2022). OpenModelica.jl is an experimental Modelica implementation implemented in the Julia language and compiles Modelica code to ModelingToolkit (MTK) (Ma et al., 2021) and is capable of runtime reconfiguration of models.

To handle the proposed extension for dynamically overconstrained connectors, we extended the flat Modelica representation to also contain a self-reference before connections are resolved. Furthermore, we reused the dynamical capabilities of *OpenModelica.jl* described in (Tinnerholm et al., 2022).

When the condition for a *DOCC-If-Equation* such as the one for the dynamic transmission line in Listing 2 is fulfilled at the time of the change t_Δ the following steps are taken:

- The simulation halts, and a *Connection* operator is either inserted or removed, and the virtual connection graph is updated.
- A new equation system is derived from the resulting connections and the changed overconstrained connection graph.
- The system is recompiled with the new equation system
- The simulation restarts using the previous values before the event at t_Δ.

It should be noted that this could allow for a more general treatment of other types of structural variability, e.g., conditional `connect()` statements, since the OpenModelica.jl framework allows for dynamic reconfiguration of Modelica models.

However, as discussed previously, in some cases, recompilation is not necessary. The examples included in the DynamicOverconstrainedConnectors package all fall in this category; they are static[5], while the virtual overconstrained connection graph, for `System4` as depicted in Figure 2 and Figure 3, changes its structure during the simulation.

In the applications showcased by the exemplary library, the overconstrained variables carry around a scalar value that is relevant to the behavior of the connected subsystems. For power systems, it is the AC phase or frequency. For incompressible fluid networks, it is the network ID. As noted in Section 2.3, this means that the selected root node sets the value of the overconstrained connector variable, which is then propagated through connection equations and the conditional equality equations when the corresponding `Connections.branch()` statement is either activated or deactivated.

For `System4` this variable is `G1_port_omegaref` before the changes in the virtual connection graph. After this change, this value is provided by `G1_port_omegaref` and `G2_port_omegaref` as depicted in Figure 3. This means that some of the general steps of handling this solution described earlier can be omitted. Instead of recompiling the system at time t_Δ, the set of variables that are part of the virtual connection graph can instead be reinitialized at that time t_Δ using the root value. Hence, recompilation of the system is not needed.

In `System4` these roots are `G1_port_omegaRef` and `G2_port_omegaRef` after $t = 10.0$ as depicted in Figure 3. Instead of recompiling, the second and third steps are as follows:

- The required modification is derived from the resulting connections and the changed overconstrained connection graph.
- The causality is changed for the equations involving the new roots

That is, the difference between the reinitialization approach and the recompilation approach is that instead of recompiling the system and regenerating the equations, we change the reference values of the roots based on the overconstrained connection graph in the simulation runtime. In this example this is done by identifying the new roots and the corresponding root sources. For `System4` this is `G2_port_omegaRef` with the source being `G2_omega`. In the case of `System4` only one equation is modified, that of `G2_port_omegaRef`.

[5]The causality changes, but the number of equations and variables remains the same

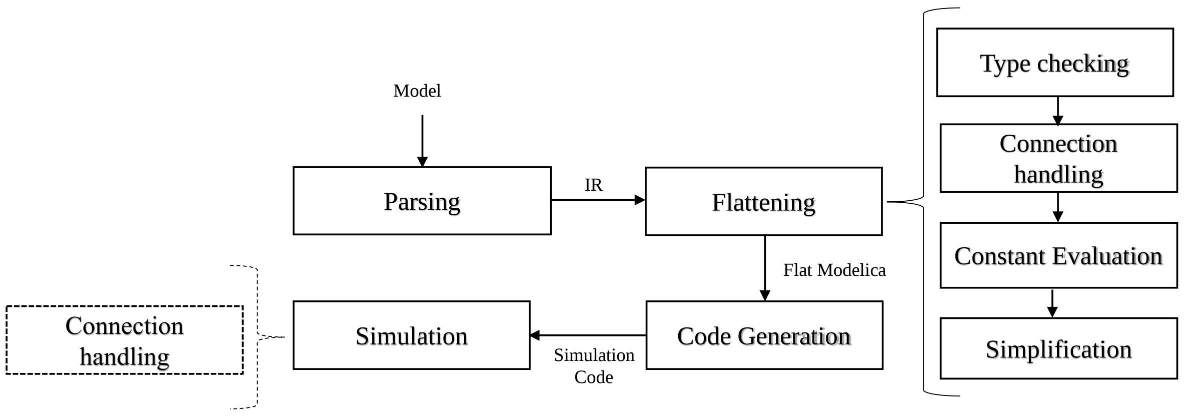


Figure 7. The translation process of a Modelica Compiler. The model is first translated to an internal intermediate representation (IR) where typing and type checking is performed and where the declared connections are handled and expanded before the simulation code is generated. The dashed box to the left shows where the new extension is handled in the compilation process.

4 Simulation Results

Handling overconstrained connector variables dynamically at runtime allows for the successful simulation of models that existing Modelica tools cannot currently handle because of model singularities. It also allows stiff solvers to increase the step size in some situations, which leads to improved simulation performance.

In this section, we demonstrate these benefits on a selection of the models presented in Section 2.1.

The systems were simulated using the RODAS5 solver available from DifferentialEquations.jl (Rackauckas and Nie, 2017).

4.1 Synchronous Power Grid models

The `System3` and `System4` models are equivalent with respect to the generator frequency and power variables `G1.omega`, `G2.omega`, `G1.Pe`, `G1.Pc`, `G2.Pe`, and `G2.Pc` . Indeed, the transients of those variables turn out to be identical in the simulations of the two models.

As explained in Section 2.1, this is not the case for the voltage and current phasors. Figures 8 and 9 show the real part of the voltage phasor of `G2`. During the first 10 seconds (left-hand-side plots), the grid is fully connected in one synchronous system using the frequency of `G1.omega` as reference, so both phasors settle down to a steady state after about 8 s.

However, when the breaker `T2` is tripped open, the right-hand-side island, to which `G2` belongs, settles down to a slightly different frequency than the left-hand-side one. As anticipated, the voltage phasor of `G2` continues to oscillate forever in the model with static overconstrained connectors, while it remains practically constant in the model with dynamic overconstrained connectors, thanks to the correct choice of reference frequency after the splitting into two islands.

This allows a stiff solver such as *RODAS5* to take much longer steps, completing the simulation with less steps and less Jacobian calculations, as shown in Table 1.

The situation is similar when comparing the simulations of `System7`, which has a static connection graph, and `System8`, which has a dynamic connection graph. When the breaker of line `T2` is opened, two synchronous islands are formed, one including `G1`, and one including `G2` and `G3`.

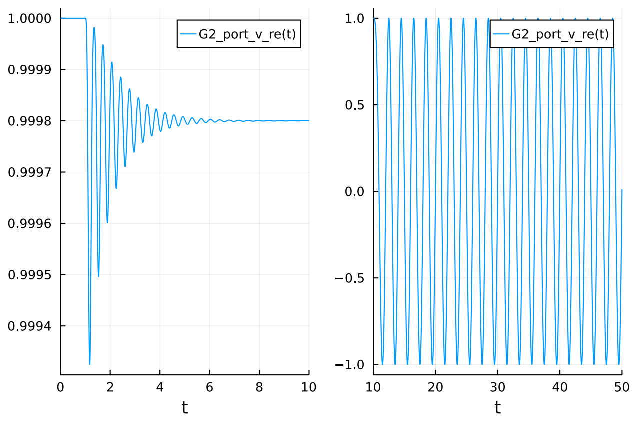


Figure 8. Plots of the `G2.port.v_re` variable in `System3` before and after the susceptance of line `T2` is brought to zero at $t = 10$. The phasor oscillates forever because the system only has one root node also after the network splitting.

Figures 10 and 11 show again the real part of the voltage phasor of `G2` for the two models, before and after the splitting. In this case, once the island containing `G2` and `G3` is formed, these two generators oscillate against each other for a while, but eventually end up rotating at the same speed. When using dynamic overconstrained connectors, the frequency of `G3` is used as a reference, so the voltage phasor of `G2` eventually becomes constant, while it does not in the static overconstrained connector case, for which the frequency of `G1` is still used as a reference.

As in the previous case, using dynamic overconstrained connectors has beneficial effects in terms of less integration steps and less Jacobian computations, see Table 1.

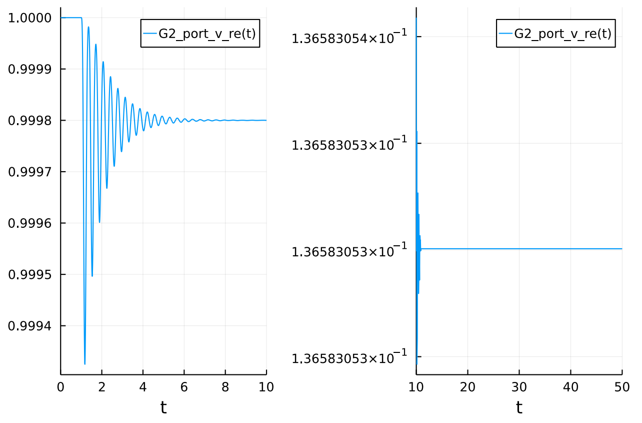


Figure 9. Plots of the `G2.port.v_re` variable in `System4` before and after the susceptance of line `T2` is brought to zero at $t = 10$. The phasor oscillates remains practically constant after the splitting thanks to the correct choice of reference after the splitting.

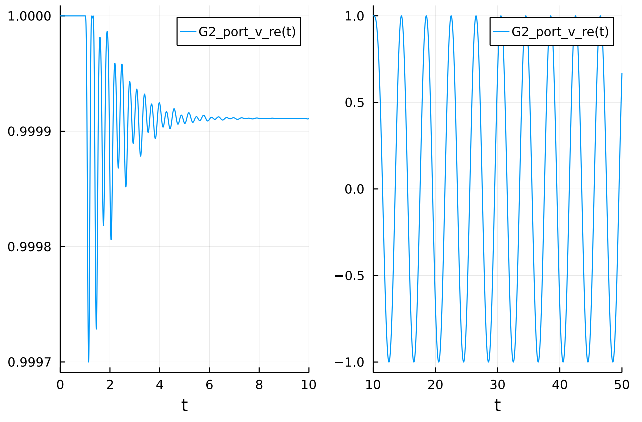


Figure 10. Plots of the `G2.port.v_re` variable in `System7` before and after the susceptance of line `T2` is brought to zero at $t = 10$. The phasor oscillates forever because the system only has one root node also after the network splitting.

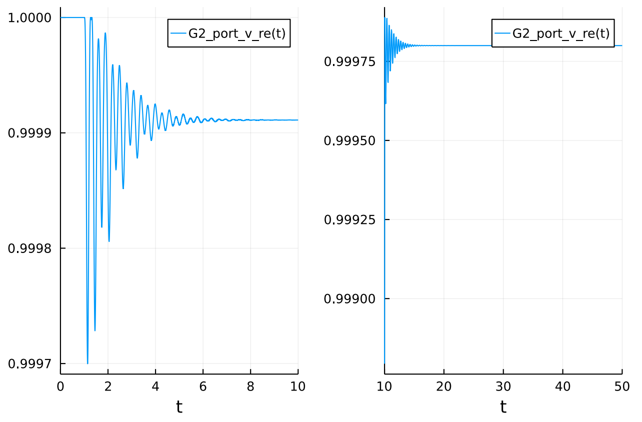


Figure 11. Plots of the `G2.port.v_re` variable in `System8` before and after the susceptance of line `T2` is brought to zero at $t = 10$. The phasor oscillates for a while but then settles to a constant value after the splitting, thanks to the correct choice of reference after the splitting.

Table 1. Number of accepted steps and the total number of Jacobians created for Systems 3, 4, 7 and 8. Systems 3 and 4 are identical except for the use of dynamically overconstrained connectors in System 4. Systems 7 and 8 have the same relationship.

System	Accepted Steps	Jacobians Created
System 3	565	605
System 4	125	132
System 7	374	389
System 8	169	175

4.2 Incompressible Fluid networks

When simulating `System3`, as described in Section 2.2, the first valve is closed at $t = 2$, and then the second is closed at $t = 4$. From that point in time, the algebraic system of equations determining the circuit pressures and flows is reported as singular when simulating the system using the OpenModelica tool (Fritzson et al., 2020).

As anticipated, the reason is that the pressures in the right-hand-side part of the system have infinitely many solutions[6]. Model `System4` instead uses the proposed extension and, as expected, the system shows no singularity for $t \geq 4$.

5 Conclusion and Future Work

In this paper, we have illustrated and discussed the benefits if some of the current constraints of the Modelica language are lifted, allowing for dynamic overconstrained connection graph, with application in synchronous AC system models and closed incompressible fluid system models. With reference to phasor-based models of synchronous AC systems, one benefit is that solvers can take much larger steps and subsequently need to create fewer Jacobians if the system is split into multiple, independent synchronous sub-systems by opening breakers on strategic transmission lines, as was shown in Section 4. Another benefit is handling the formation of islands without generation capacity, avoiding the termination of the simulation because of unsolvable equations. With reference to the models of closed hydraulic systems with an incompressible fluid, the benefit is that it is possible to handle the formation of independent closed sub-systems by closing valves that separate two or more parts of the circuit without leading to singular systems of equations.

Furthermore, the reconfiguration approach described in Section 3 could be improved. Currently, runtime reconfiguration in OpenModelica.jl requires the system to keep the equation structure before and during simulation; hence we have to omit important optimisation phases such as removing trivial equality constraints. Consequently, there is currently a trade-off between optimisation and the speed of system reconfiguration. Still, if we consider the cost of

[6]It should be noted that existing Modelica tools can handle this scenario by selecting one of the several solutions.

recompilation, this approach should be more efficient for small systems even though it currently only works without some optimisation phases.

In general, it seems that the extension proposed in this paper can be implemented with a reasonable effort in mainstream Modelica compilers, as long as the structure of the system using it is similar to that of the presented use cases, which will indeed be the case for a significant number of real-life use cases.

Although the current study is based on a small set of examples from a conceptual library, the findings suggest that dynamically overconstrained connectors could be employed to simulate real-life systems that is not possible to simulate in existing Modelica tools and provide possible performance benefits. Therefore, a direction for future work would involve implementing support for dynamically overconstrained connectors in the OpenModelica Compiler to investigate the general applicability of this construct on larger systems, for example using the PowerGrids library (Bartolini et al., 2019).

The final goal is to get this extension into a future version of the Modelica Language Specification, so that it gets eventually supported by a growing number of Modelica tools, allowing library developers to use it without concerns about limited support.

Acknowledgement

This work has been supported by the Swedish Government in the ELLIIT project and has been partially funded by Vinnova in the context of the ITEA project EMBRACE. Furthermore, the authors gratefully acknowledge the financial support of the French Transmission System Operator RTE to the Open Source Modelica Consortium, which made this initial development possible.

References

Andrea Bartolini, Francesco Casella, and Adrien Guironnet. Towards pan-european power grid modelling in Modelica: Design principles and a prototype for a reference power system library. In Anton Haumer, editor, *Proc. 13th International Modelica Conference*, pages 627–636, Regensburg, Germany, Mar 4–6 2019. doi:10.3384/ecp19157627.

Peter Fritzson, Adrian Pop, Karim Abdelhak, Adeel Ashgar, Bernhard Bachmann, Willi Braun, Daniel Bouskela, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Rüdiger Franke, Dag Fritzson, Mahder Gebremedhin, Andreas Heuermann, Bernt Lie, Alachew Mengist, Lars Mikelsons, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Vitalij Ruge, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, John Tinnerholm, and Per Östlund. The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Modeling, Identification and Control*, 41(4):241–295, 2020. doi:10.4173/mic.2020.4.1.

Yingbo Ma, Shashi Gowda, Ranjan Anantharaman, Chris Laughman, Viral Shah, and Chris Rackauckas. Modelingtoolkit: A composable graph transformation system for equation-based modeling, 2021.

Sven Erik Mattsson Martin Otter, Hilding Elmqvist. The new Modelica MultiBody library. In *Proceedings 3rd International Modelica Conference*, pages 311–330, Linköping, Sweden, Nov 3–4 2003.

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(1), 2017.

Hans-Jürg Wiesmann Rüdiger Franke. Flexible modeling of electrical power systems – the Modelica PowerSystems library. In *Proceedings 10th International Modelica Conference*, pages 515–522, Lund, Sweden, Mar 10–12 2014. The Modelica Association. doi:10.3384/ecp14096515.

The Modelica Association. Modelica - A unified object-oriented language for physical systems modeling - Language specification version 2.1. Online, Jan. 30 2004. URL `https://modelica.org/documents/ModelicaSpec21.pdf`.

The Modelica Association. Modelica - A unified object-oriented language for physical systems modeling - Language specification version 3.5. Online, Feb. 19 2021. URL `https://modelica.org/documents/ModelicaSpec35.pdf`.

John Tinnerholm, Adrian Pop, and Martin Sjölund. A Modular, Extensible, and Modelica-Standard-Compliant OpenModelica Compiler Framework in Julia Supporting Structural Variability. *Electronics*, 11(11), 2022. ISSN 2079-9292. doi:10.3390/electronics11111772.

Solving Flow Balancing Problem for Hybrid-Electric Aircraft Cooling Systems

Clément Coïc Michael Sielemann Nirmala Daniel Andersson

Modelon, {name.surname}@modelon.com

Abstract

A flow balancing problem consists of sizing restrictions on flow branches of a fluid system to match desired flow rates on each branch. The problem is rarely trivial as parallel branches routinely contain many components with nonlinear pressure loss characteristics each. This paper introduces the Physics-based Solving capabilities implemented in Modelon Liquid Cooling library. This new capability enables conveniently solving such flow balancing problems with steady-state requirements. The benefits of this solution are discussed using an aircraft thermal management system as example.

Keywords: Flow Balancing, Cooling System, Liquid Cooling, Thermal Management System, Aircraft, Sizing, Modelon Impact

1 Introduction

Recognizing the need for a sustainable future, the aerospace industry is heavily researching innovative technologies that could reduce its impact on the environment. While disruptive technologies such as hydrogen-powered aircraft are being investigated, more incremental ones such as hybrid-electric designs – combining a conventional propulsion system with an electric one – are expected to enter service before, providing substantial fuel efficiency and emissions improvement.

However, adding the hybrid electric propulsion system has impact: it adds direct and indirect weight. The weight of electric propulsion components – batteries, inverters, electric motors, etc. – directly reduces the aircraft payload. In addition, these components need to be cooled, indirectly increasing the weight of the thermal management system. Therefore, achieving an optimized design, when it comes to hybrid-electric aircraft, involves a careful trade-off between the electric propulsion system and the thermal management system designs.

One challenge to solve when designing both coupled systems is to size flow restrictions on the cooling branches. These restrictions, on each branch, define the nominal flow rate of cooling fluid and, therefore, the cooling capacity of a branch. These cooling requirements are directly derived from heat dissipated by each component – here, the electric propulsion components. If we decide to neglect storage effects of heat capacities in the design of the thermal management system, then the amount of heat to be extracted is thus constant over (a sufficiently long period of) time (in other words a steady-state value). The most demanding of these (quasi) steady-state conditions are identified by engineers as sizing conditions, and are, for instance, function of the components selected for the system to be cooled.

Optimizing the hybrid-electric aircraft design is thus an iterative process which aims at maximizing the aircraft range for a given minimum payload. Each iteration involves assessing the electric propulsion component sizes and associated heat loads in order to solve the flow balancing problem – which is key for the thermal management system sizing. This paper focuses on a robust solution to the steady-state flow balancing problem.

With the goal to give more technical background on the problem, section 2 introduces the flow balancing problem for the specific case of a thermal management architecture of a hybrid-electric aircraft. Section 3 discusses the tools we used to solve the flow balancing problem and introduces Physics-based Solving, a combined symbolic and numerical computation technique for the Modelica language. Section 4 shows how the flow balancing problem is solved. Section 5 concludes this publication and draws conclusions.

2 A Thermal Management System for Hybrid-Electric Aircraft

2.1 Architecture Selection

The scope of this paper is not to propose a new hybrid-electric aircraft architecture with an innovative thermal management system but rather to solve a recurrent problem in the currently suggested architectures. To support this argument, a typical architecture , presented by Gkoutzamanis (2022), is used in this paper.

The selected hybrid-electric aircraft architecture is targeting regional commuter aircraft. It consists of two conventional turbo-propellers, from which mechanical power is also extracted to feed electric generators that, in turn, power an electric motor connected to an aft Boundary Layer Ingesting (BLI) fan. For electric power management reasons, the alternating current (AC) electricity generated is converted to direct current (DC), potentially stored into a battery, and inverted back to AC, prior to consumption by the electric motor. **Figure 1** illustrates this architecture.

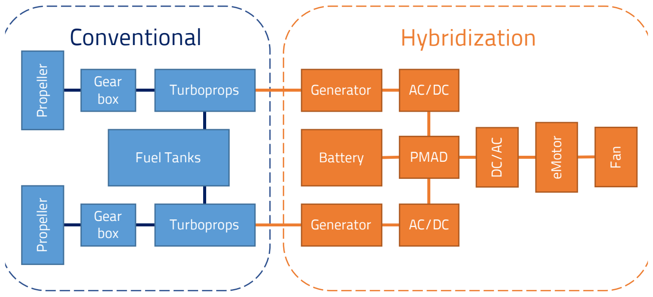


Figure 1. Selected hybrid-electric aircraft architecture.

A highlight of this architecture is that hybridization is added to the conventional propulsion. A typical engineering problem is to estimate differences with respect to the conventional propulsion architecture ("baseline"). How much weight does the hybridization add? One must consider the electrical power system, the thermal management system, and snowball effects on the aircraft structure. Beyond this, engineers face many more questions such as for installation constraints, safety aspects, availability, maintenance, etc. We focus on the thermal management system cooling the electric propulsion system – depicted in **Figure 2**.

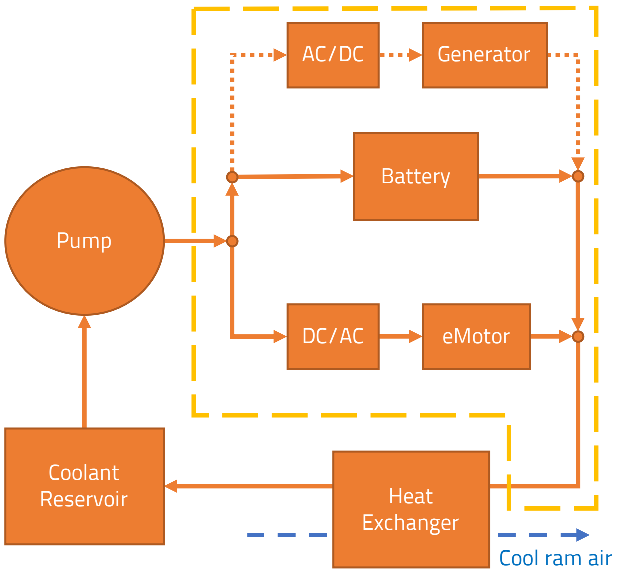


Figure 2. Added Thermal Management System.

The thermal management system on such hybrid electric aircraft must cool both the conventional components and the thrust-generating electric power system. Based on the location of the components on the aircraft, two main routes are identified:

1. Top two branches near the electric power generation and storage – near the turboprops, and thus wings. The dash line represents a branch that can potentially be by-passed, as the electrical components might not always be active, e.g., at take-off.

2. One bottom branch for the electronics and electric motor (eMotor) powering the aft BLI fan – on the rear part of the aircraft.

After capturing the heat loads, both routes merge before dissipating this heat via an air-cooled heat exchanger.

2.2 Flow Balancing Problem Statement

From the thermal management point of view, each component is treated as a heat load to dissipate. While each component can have several different characteristics, only the amount of heat it generates initially matters. In a later step of the design, the pressure loss characteristics and thermal resistance (between coolant bulk flow and heat load) of cooling elements, e.g., cold plates, are refined for each component to be cooled. Nevertheless, the overall need for coolant flow shall not be changed and is function of the heat loads.

Table 1 – also extracted from Gkoutzamanis (2022) – summarizes the heat loads to evacuate for each component and the temperature limits these components should not reach. These temperatures are typically imposed by material constraints. For instance, Budinger (2020) states that the "main design criterion for the motor is the maximum winding temperature" . Every technology might have different criteria driving these temperature limits but the limit shall not be reached to ensure component integrity. While heat loads are computed in such a way that the temperature is not reached, it is still relevant to monitor the component temperature in the simulation outputs and ensure that these requirements are met.

Table 1. Component heat loads.

Component	*Temperature limit* [°C]	*Heat loading* [kW]
eMotor	100	20
Generator	100	20
Inverter	65	10
Converter	65	10
Battery	40	10

To define the flow balancing problem in the thermal management context, we introduce an engineering design rule relating the heat load to the mass flow rate. For its derivation, we start with the First Law of Thermodynamics for a constant volume V, density ρ, internal energy u, enthalpy flow rate at each interface $\dot{H}_a$ and $\dot{H}_b$, and heat flow rate $\dot{Q}$:

$$V\frac{d(\rho u)}{dt} + \dot{H}_a - \dot{H}_b = \dot{Q}$$

If we assume steady state, then the mass flow rates at each interface are identical, $\dot{m}_a = \dot{m}_b = \dot{m}$, and the time derivative vanishes.

$$\dot{m}(h_a - h_b) = \dot{m}\Delta \mathrm{h} = \dot{Q}$$

The engineering design rule for a typical coolant and cooling system technology, is to consider a design factor of, for instance, $\dot{Q}/\dot{m} = \Delta h = 30$ kW/(kg/s) – a mass flow rate of 1 kg/s can evacuate a heat load of 30 kW. This design factor combined with the heat loads of each components gives a first estimate of the mass flow rate required on each branch of the thermal management system to cool the electric propulsion system. It is important to understand that this is first step in an engineering workflow and assumes that, independently of the specific mass flow rate, all heat is transferred *as required* from the device being cooled to the coolant. The detailed design of the cooling surface, e.g., cold plate, can be conceived at a later stage. Here, it is sufficient to assume that the cooling is feasible.

Based on the thermodynamic properties of the coolant and the component inlet temperature, the design factor can be related to temperature change. With the simplifying approximation of constant specific heat capacity, we can deduce that the resulting temperature change can be roughly in the range of 8 K to 17 K for typical coolants propylene glycol water solution, polyalphaolefin, and turbine cooling oil MIL 23699.

The flow balancing problem consists of computing dimensions of calibration restrictions on each branch in order to match the desired mass flow rates, while ensuring that the physics laws are satisfied – e.g. here, the pressure drop of the three parallel branches are identical.

Obviously, each branch, in addition to these flow balancing restrictions, also includes a multitude of routing components – mainly bends and pipes – and aggregates their nonlinear characteristics. For the purposes of this paper, the pump and heat exchanger are substituted by ideal pressure boundaries so that only the branches and routings are considered for the problem solving – part of the system framed with a yellow dashed line. Several pipes and bends are added and parametrized to the system.

It is worth noticing that the flow balancing problem is a steady-state problem which is part of an overall design optimization loop. While Modelon Impact – the platform used in this paper to solve the flow balancing problem – is well suited to solve optimization problems (Coïc 2022), other platforms such as OpenMDAO (Zhao, 2019) (Hecken, 2020) or FAST-OAD (Delbecq, 2021) also proved to support this need. As Modelon demonstrated the optimization capability, with all these platforms, this paper focuses solely on the flow balancing problem solving.

3 Implementation of Physics-based Solving in Modelon Liquid Cooling

Modelon Liquid Cooling library (LCL) is used within Modelon Impact to model the thermal management system. As solving a steady-state problem is not the initial strength of the Modelica Language, Physics-based Solving (PbS) is added to LCL to support this workflow.

3.1 Modelon Impact

Modelon Impact is a next generation system modeling and simulation platform, leveraging the benefits of web and open standard technologies. With openness at its core, Modelon Impact supports standards such as Modelica, FMI, Python and REST (Modelon, 2022-a). The user-friendly browser interface provides modeling experts the tools they need to create, simulate, and experiment. Steady-state or dynamic solutions can be executed from the same model, reducing effort to get an answer (Coïc, 2020-b). Finally, the Modelon Impact API enables user-specific workflows through Python-based custom functions, and deployment of models to non-experts via targeted web applications or Jupyter Notebooks (Coïc, 2020-a).

3.2 Modelon Liquid Cooling Library

The library is used for modeling and simulation of liquid cooling systems in virtual prototyping, component dimensioning and control design.

The library includes more than 80 internal flow components such as pipes, bends and junctions with predictive geometry-based flow resistance correlations. It also includes generic components that can be calibrated from measurement data. More than twenty fluid models are provided in the library, with temperature-dependent properties to support cooling system modeling for water, customizable glycol-water and alcohol-water mixtures, every relevant water-salt mixture (e.g. potassium carbonate), calcium chloride, sodium chloride, potassium acetate, etc. The library also contains thermodynamic property models of aerospace-specific fluids such as turbine oil, polyalphaolefin, hydraulic oil and jet fuels.

Pre-configured templates guide users in creating simplified, high-performance heat exchanger stack models with 3D visualizations for parameter verification and presentation of resulting temperatures.

The Liquid Cooling Library (LCL) can be used effectively in conjunction with geometry-based models from the Heat Exchanger Library.

3.3 Steady-State and Physics-based Solving

When the answer expected from a model is the equilibrium point of the modeled system, steady-state simulation maximizes productivity; you obtain the result directly and faster, by orders of magnitude, and it simplifies post-processing of the results. You can extrapolate the gain on a design exploration, where you run hundreds or thousands of points (Modelon, 2022-b).

Modelon Impact includes steady-state solvers, as well as our Physics-based Solving (PbS) technology, that enables adding engineering insights derived from fundamental physical principles in models so that the steady-state simulation solves faster and in a robust way. Reconfiguring the numerical problem (without recompilation) also allows answering several questions with a single model.

PbS consists of instructions embedded in component models guiding the compiler and solver on iteration variable and residual selection for the steady-state simulation. This language construct enables changing the iteration variables and residuals based on Boolean parameters, without the need for recompilation. The information is stored in an object-oriented fashion, such that modelers can assemble systems graphically, and the desired solving can be deduced from the model topology (model instances and connections). This has been introduced in some detail before in (Coïc, 2020-b).

3.4 PbS Implementation in LCL

Typical Liquid Cooling models have a well-established flow direction. The fluid is directed from the pump toward the parts to cool-down and later cooled down through a heat exchanger before getting back to the reservoir.

As the flow direction is known, the authors implemented a serial approach of PbS in LCL. At the beginning of each branch of the cooling system, the mass flow rate is known or guessed. At the end of each branch, the type of component defines the residual equation of physics – either the user-defined value of a boundary condition, or the identical pressures for a junction.

In addition, the authors simplified the equations where possible, based on the steady-state and unidirectional flow assumptions. Notably, the Modelica language allows switching between such assumptions and thus the same library supports both dynamic and steady-state simulation – only a top-level parameter is changed to switch between the two simulation modes.

4 Solving the Flow Balancing Problem

In this section, a component model is first presented, to give more insights on the orifice sizing. Then, the thermal management system modeling and flow balancing solving are discussed.

4.1 Sizing Orifices

The specific case of the orifice sizing is presented here. PbS is added to the *orifice plate with circular opening* (see **Figure 3**). The component model can be configured into two different modes: orifice sizing and flow simulation. The former case imposes a user-defined flow rate to compute the orifice size $D_{opening}$, while the later enforces the orifice size to compute the pressure or flow unknown.

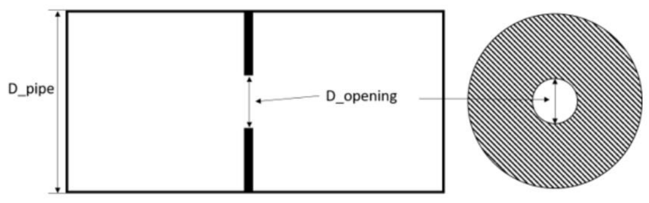


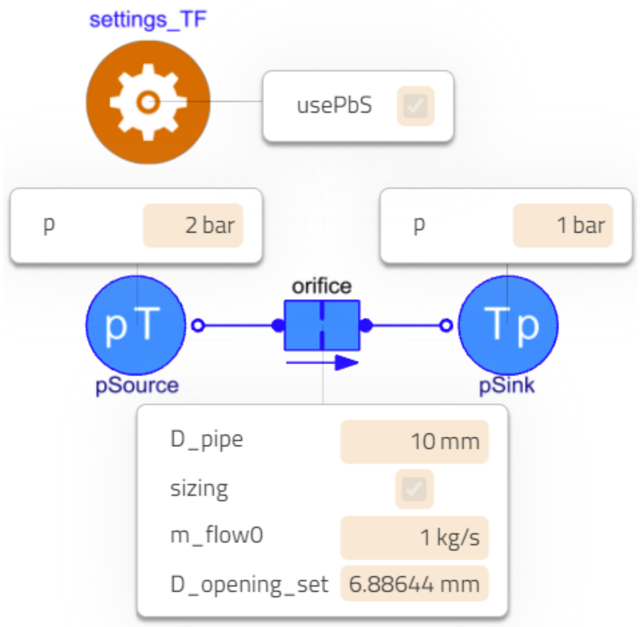
Figure 3. Orifice plate with circular opening.

A couple of statements are relevant to highlight:

- The orifice diameter $D_{opening}$ is of parameter variability. Solving the equations for it only makes sense at initialization or in a steady-state problem. To differentiate the user input parameter from the iteration variable for the sizing problem, $D_{opening_{set}}$ is used instead in PbS mode.

- At system level, it was mentioned that typically the mass flow is iterated on to match boundary conditions. For the case of the orifice sizing, the diameter being an unknown, the mass flow at the sizing point $m_{flow,0}$ shall be specified. Hence, the pressures are computed from the known mass flow rate and the pressure correlation in the branch.

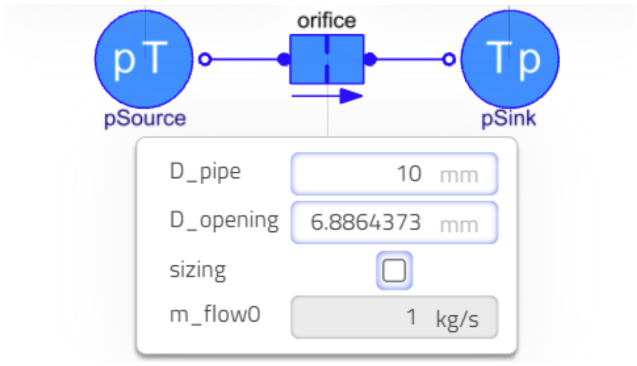
A unitary test of the orifice component could thus be similar to **Figure 4**, where pressure boundaries are set, and the sizing point mass flow rate is prescribed. The pipe diameter D_{pipe} is a necessary parameter to compute the section change and would typically come from a pipe or previous component in a system level model. The greyed-orange background on the values indicates that it displays the results and thus cannot be changed. The orifice diameter $D_{opening_{set}}$ is the result of the simulation



p	2 bar
p	1 bar
D_pipe	10 mm
sizing	✓
m_flow0	1 kg/s
D_opening_set	6.88644 mm

Figure 4. Orifice sizing – unitary test 1.

The solver converged to an orifice diameter of about 6.89 mm for a pipe diameter of 10 mm to satisfy the 1 Bar pressure difference and 1 kg/s mass flow rate. Conveniently, in editing mode, it is possible to turn off the *sizing* mode and specify the desired $D_{opening}$ and simulate a standard flow simulation – without orifice sizing, hence $m_{flow,0}$ is not used and disabled.



D_pipe	10 mm
D_opening	6.8864373 mm
sizing	☐
m_flow0	1 kg/s

Figure 5. Orifice flow simulation – unitary test 2.

The orifice sizing is key in solving the flow balancing. It is now possible to introduce the thermal management system model and solving it is analogous to this section, only at a larger scale and for larger non-linear systems.

4.2 Modeling the System

The thermal management system is modeled in Modelon Impact using the Liquid Cooling Library. The model topology is based on **Figure 2** and shown in **Figure 6**.

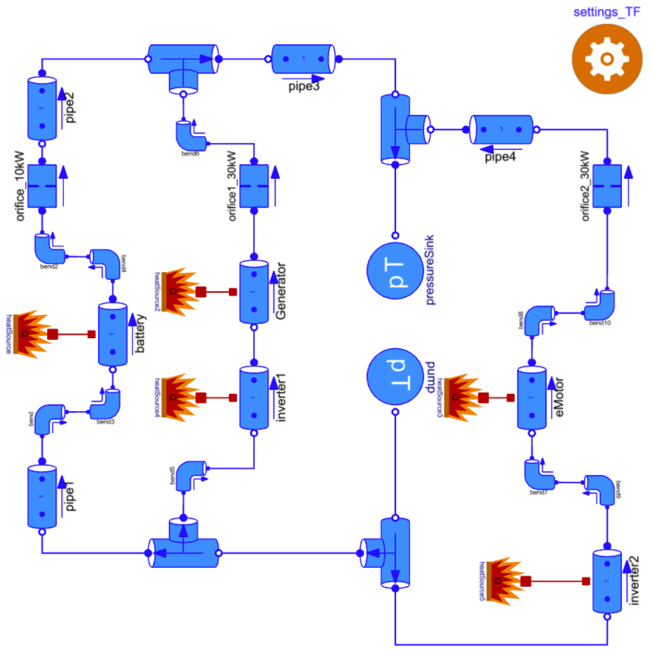


Figure 6. Thermal Management System model.

The following modeling decisions were stated previously but are repeated below for convenience:

- The pump and heat exchanger are substituted by ideal pressure boundaries so that only the branches and routings are considered for the problem solving – part of the system framed with a yellow dashed line.
- Several pipes and bends are added and parametrized to the system.

The components to be cooled are represented by heat sources. Their heat flow rates are set to the values specified in **Table 1** and their cooling interfaces are here modeled by pipes. Typically, a later refinement would involve modeling the cooling interface with higher fidelity, e.g., using several cold plates in parallel.

An orifice is added per branch to be sized as core aim of the flow balancing problem. These are named *orifice* followed by a number and postfix describing the total heat load that the branch needs to cool.

4.3 Solving the System

Once the system is modeled, the orifices are set in sizing mode and the mass flow rates at the sizing points are set to evacuate the heat loads with the considered design factor of 30 kW/(kg/s) – so $m_{flow,0}$ for *orifice1_30kW* is set to 1 kg/s.

The system is simulated with PbS in steady-state. The flow balancing solving happens without further user action. The compiler selects the iteration variables and residual equations – specified in the library at component level – and robustly solves the problem. **Figure 7** shows the results and associated thermal coloring.

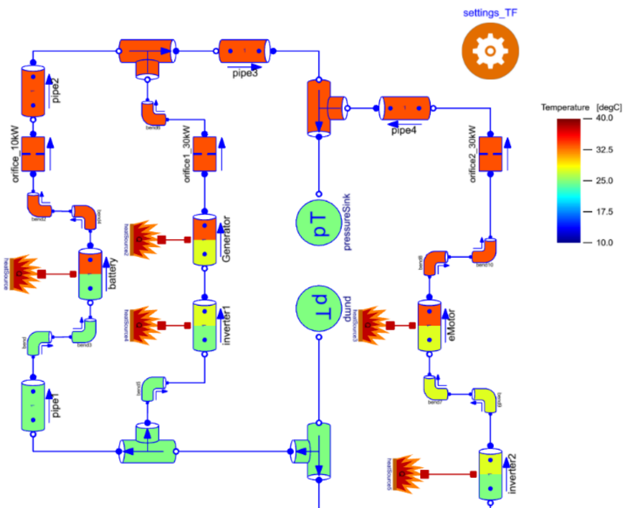


Figure 7. Thermal management system flow balancing solving

The sample value of the design factor results in $\Delta h_{heatload}$ =30 J/kg and, assuming 25 °C pump outlet temperature and propylene glycol water mixture 60%, the component outlet temperatures given in **Table 2**.

Table 2. Component temperatures.

Component	*Temperature inlet* [°C]	*Temperature outlet* [°C]
eMotor	28.0	33.9
Generator	28.0	33.9
Inverters	25.0	28.0
Converter	25.0	28.0
Battery	25	33.9

The computed orifices diameters can easily be applied to the system as Modelon Impact offers the option to start a simulation from the results of a previous one. Should you prefer entering a different value from a catalog, this is obviously also an option. A key result is to find the combined pressure loss characteristics of the branches for off-nominal pump head as shown in **Figure 8**. This then allows investigating further the design for additional requirement validations.

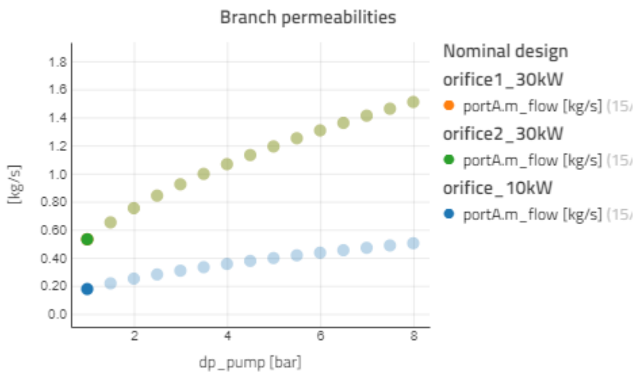


Figure 8. Off-design mass flow vs. pressure loss characteristics.

4.4 Further Requirement Verifications

First, the sizing point may not be defined as a deterministic single point but via ranges of expected heat loads, or it is defined by a deterministic single point but the engineer is interested in studying the impact of increasing or reducing the margin for thermal component performance (i.e., extract the same heat load at lower mass flow rate and increased temperature difference between coolant outlet and inlet). It is thus possible to perform multiple executions of the sizing simulation, covering the appropriate domain, and extracting the resulting ranges of calibrations. Modelon Impact provides dedicated functionalities for design of experiments – from a simple *choices* and *range* operators that enable defining a set of values, to more involved functionalities such as *Latin Hypercube Sampling*. Designing the orifices might thus involve several simulations and the selection of the most constraining design. This is simplified with steady-state simulation as the results are single values that can be easily compared and post-processed.

In the following, we assume that the orifice 2 design factor is swept between 37.5 kW/(kg/s) and 25 kW/(kg/s), i.e., $m_{flow,0}$ for *orifice2_30kW* is set to 0.8 kg/s to 1.2 kg/s. All other parameters are held constant. Orifice dimensions and resulting temperatures can be computed, and the latter are shown in **Figure 9**. The nominal design is highlighted.

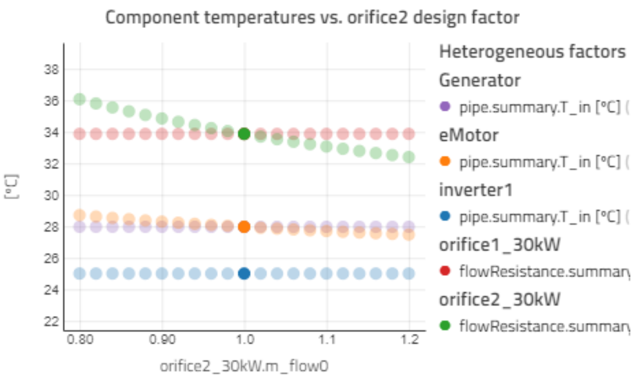


Figure 9. Temperature sensitivities over branch design factor.

Obviously, the thermodynamic model must always be satisfied. Based on the pressure loss characteristics of the branch components, a maximum mass flow rate of each branch must not be exceeded. If this point was reached, it would not be possible anymore to reduce the restriction in the orifice for calibration as the orifice diameter had already reached the pipe diameter. In the given problem, this occurs for single digit design factors and correspondingly high mass flow rate through *orifice2*; see **Figure 10**. In another network with more restriction, this can occur more easily.

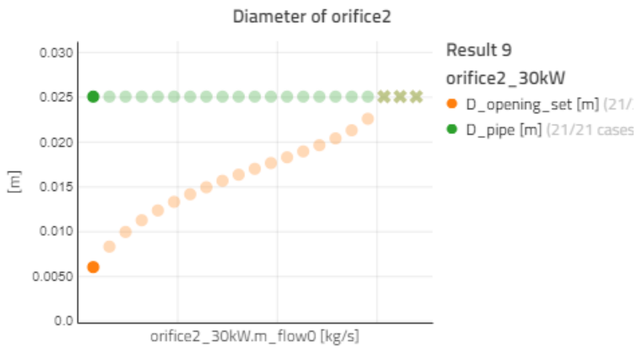


Figure 10. Diameter sensitivities over branch design factor.

Second, while there are heat load requirements on each component, there are also temperature limit constraints to satisfy. It is a good practice to simulate several points in the operational domain to ensure that the constraints are met – and that the requirements are correctly defined. For these simulations, the design of the orifices shall be set by a parameter and the focus would be on flow simulations. As illustrated for a single component in **Figure 5**, this does not require a new model to be developed but simply switching the *sizing* parameters to *false* and setting the diameter values.

Finally, Modelon Impact comes with a Python client (Modelon, 2022-c) that can conveniently define experiments, simulate and return results. System simulation and analysis to verify requirements can thus easily be automated, in a fully integrated manner.

5 Conclusion and Perspectives

This publication shows how the design of hybrid electric aircraft can be simplified with an efficient workflow for solving the flow balancing problem of the thermal management system. Modelon Impact enables automated steady-state solving and requirement verifications. Its openness makes it easy to be integrated in a system-level design loop that also involves the electrical system sizing.

The steady-state and flow balancing capabilities have been developed and tested on larger customer models including more than 300 individual components. The example discussed here has an order of magnitude fewer components and serves the purpose of illustrating the key workflow and modeling principles.

The flow balancing workflow can also be used more widely outside the thermal management system context. Whenever the distribution of fluid is of interest, restrictions can be calibrated to yield the desired split. This methodology is equally applicable to aircraft air distribution ducting from mixer to riser ducts and cabin air outlets, building air supply networks and so on.

While this proved the tool capability, the system model can be further refined to include the closed loop of the cooling system and design-specific cooling devices such as cold plates. A full workflow involving the Jet Propulsion and Thermal Management System – both running in steady-state, as embedded capabilities – would be a next step of this work.

References

Budinger Marc, Aurélien Reysset, Aitor Ochotorena and Scott Delbecq (2020). “Scaling laws and similarity models for the preliminary design of multirotor drones”. In *Aerospace Science and Technology*, 98. 1-15. ISSN 1270-9638.

Coïc Clément, Johan Andreasson, Anand Pitchaikani, Johan Åkesson and Hemanth Sattenapalli (2020). “Collaborative Development and Simulation of an Aircraft Hydraulic Actuator Model”. In *Asian Modelica Conference 2020*, Tokyo, Japan.

Coïc Clément, Moritz Hübel and Matthis Thorade (2020). “Enhanced Steady-State in Modelon Jet Propulsion Library, an Enabler for Industrial Design Workflows”. In *American Modelica Conference 2020*, Boulder, Colorado, USA.

Coïc Clément, Marc Budinger and Scott Delbecq (2022). “Multirotor drone sizing and trajectory optimization within Modelon Impact”. In *American Modelica Conference 2022*, Dallas, Texas, USA.

Delbecq Scott, Marc Budinger, Clément Coïc and Nathalie Bartoli (2021). “Trajectory and design optimization of multirotor drones with system simulation”. In *American Institute of Aeronautics and Astronautics, Inc, SciTech*, DOI: 10.2514/6.2021-0211.

Gkoutzamanis Vasilis G., Spyros E. Tsentis, Orestis S. Valsamis Mylonas, Anestis I. Kalfas, Konstantinos G. Kyprianidis, Panagiotis Tsirikoglou and Michael Sielemann (2020). “Thermal Management System Considerations for a Hybrid-Electric Commuter Aircraft”. In *Journal of Thermophysics and Heat Transfer*, by the American Institute of Aeronautics and Astronautics, Inc, 2020.

Hecken Tobias, Xin Zhao, Michael Iwanizki, Max J. Arzberger, Daniel Silberhorn, Martin Plohr, Konstantinos Kyprianidis, Smruti Sahoo, Giorgio Valente, Sharmila Sumsurooah, Michael Sielemann, Clément Coïc, Andreas Bardenhagen, Annika Scheunemann and Claire Jacobs (2020). “Conceptual Design Studies of “Boosted Turbofan” Configuration for short range”. In *AIAA SciTech Forum*, by the American Institute of Aeronautics and Astronautics, Inc, 2020.

Modelon website – Impact, Lowering barriers and bridging gaps, accessed on July 2022, https://www.modelon.com/modelon-impact-introduction/

Modelon website – Steady State and Dynamic Simulation: What is the difference?, accessed on July 2022, https://modelon.com/steady-state-and-dynamic-simulation-what-is-the-difference/

Modelon Help Center website – Modelon Impact Client, accessed on August 2022, https://help.modelon.com/latest/tutorials/jupyter_mi_client_new/?h=client

Zhao Xin, Smruti Sahoo, Konstantinos Kyprianidis, Jonathan Rantzer and Michael Sielemann (2019). “Off-design performance analysis of hybridised aircraft gas turbine”. In *The Aeronautical Journal*, by Royal Aeronautical Society, Cambridge University Press, 2019.

High-Fidelity Multiphysics FCEV bus study with detailed HVAC, cabin, and Hydrogen Fuel Cell models

Theodor Ensbury[1] Alessandro Picarelli[2] Mike Dempsey[2]

[1]Claytex USA, Inc., United States, `theodor.ensbury@claytex.com`

[2]Claytex Services Ltd. Edmund House, Rugby Road, Leamington Spa, CV32 6EL, UK, `{alessandro.picarelli,mike.dempsey}@claytex.com`

Abstract

Hydrogen Fuel Cells are potentially a viable zero emission propulsion technology for heavy commercial vehicles, like buses. This paper presents a detailed proof of concept Dymola model of an FCEV bus, built using the VeSyMA suite of libraries. Particular attention is paid to the use the high-fidelity Hydrogen Library from Dassault Systèmes for the fuel cell. Representative physical ancillary systems, coupled with detailed thermochemical modelling, enables detailed transient effects on fuel cell performance to be captured. A multizonal cabin model with independent zonal thermal properties, combined with a multi-physics HVAC model provide a realistic current drain on the drive battery. Such detail is important in understanding accurately the conditions the Fuel Cell will experience during operation. A model such as this has many real-world applications, such as component design and selection; concept evaluation; Fuel Cell degradation, maintenance, and durability analysis; accurate range estimation and controller development.

Keywords: Hydrogen, Fuel Cell, HVAC, heating, cooling, bus, coach, EV, FCEV

1 Introduction

Due to the impact of global warming and air quality standards, internal combustion engine (ICE) driven vehicles are being phased out of usage. Passenger Electric Vehicles (EVs) are becoming the norm. Many markets have established sunset dates for the sale of ICE vehicles, triggering the development of electrified vehicles en masse, as intended.

The energy source for this electricity is still a cause for debate. Battery Electric Vehicles (BEVs) have emerged as the dominant technology for passenger vehicles. Such vehicles feature a solid-state electrochemical battery, usually of lithium-ion technology. As commercial vehicles are often in constant use during operational hours, the range limitations of BEVs and time it takes to recharge them, compromise their suitability for commercial applications (Andaloro et al, 2016). A comparatively low energy density of modern EV battery technology versus Hydrogen plays a role in this, with Fuel Cell Electric Vehicles (FCEVs) more suitable, capable of driving ranges greater than 300 miles (Gröger et al, 2015). Procurement of the rare earth minerals needed for BEV batteries also presents its own set of ethical and environmental supply chain concerns (United Nations, 2020).

Transit and coach buses present an interesting use case. In continuous operation during the day, transit buses stop frequently, albeit for very short periods to allow passengers to embark and disembark. In contrast, coach buses stop infrequently, but often travel long distances in a single journey. Neither present a natural use case for the pure BEV concept.

Hydrogen fuel cells (FCs), when powered with green hydrogen, present a potential solution to this predicament. FCEV buses offer a comparable range and can be considered a “one for one” replacement for existing diesel buses (Vock, 2019). Refueling times are and equivalent fuel efficiency are comparable with current diesel-powered vehicles (Eudy and Post, 2021); some manufacturers already claim comparable range and performance to conventional ICE buses (Luxfer, 2022).

1.1 Motivations for study

Beyond reducing the carbon footprint of development by eliminating almost all prototypes, simulation tools feature heavily in vehicle electrification. The multi-physics capability of modern simulation tools and languages such as Dymola and Modelica, are ideally suited to simulating the EV, an inherently complex multi-physics system. Motivations for this study can be broadly broken down into two concepts: theoretical and practical.

Initially undertaken as a technical exercise, from a theoretical perspective this work follows common themes. Primarily, it serves as a proof of concept with regards to integrating the Vehicle Systems Modelling and Analysis (VeSyMA) suite of vehicle simulation libraries from Claytex and the Hydrogen library from Dassault Systèms (DS). Work stemming from the development of this model was directly responsible for the adaptation of the VeSyMA library to interface with the Hydrogen library.

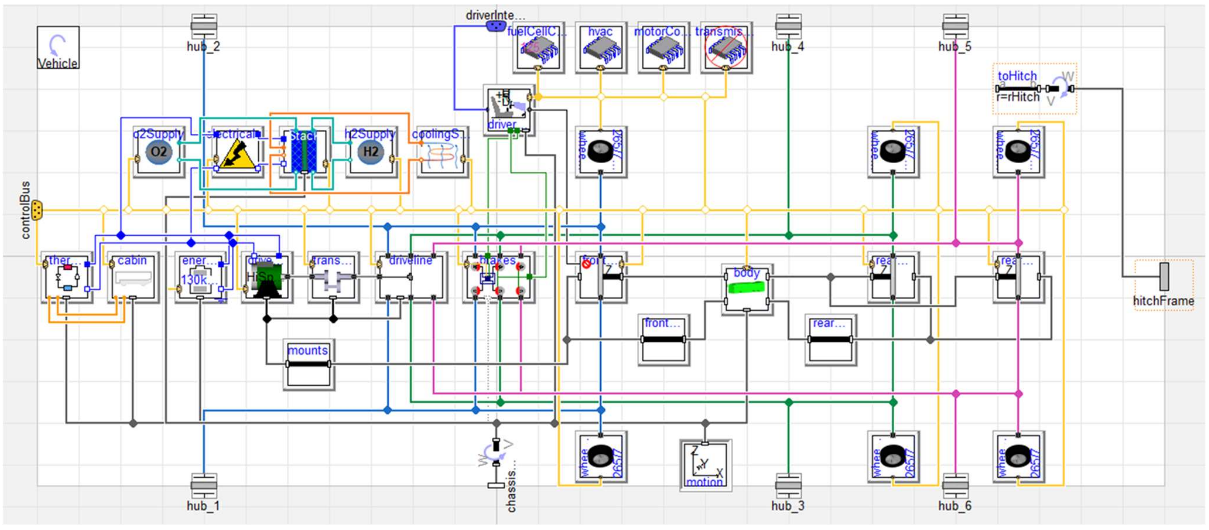


Figure 1 - FCEV Bus model by Claytex. Following the VeSyMA principles, a template system to promote model reuse and replaceability is used, allowing for scalable detail. Note the Fuel Cell components in the top left-hand corner, and the Cabin and HVAC models on the far left-hand side.

On a practical level, there are several real-world benefits. Featuring high fidelity HVAC and FC models, it can be used to accurately size, design and select components for usage and evaluate their suitability in the digital twin. Cooling needs of the FC can be evaluated with regards to the loading cycles it undergoes; needs and benefits of preconditioning can also be determined with a model such as this. With a detailed HVAC and cabin model, the FC in the model operates in realistic, and sometimes compromised, loading conditions. As FCs are very sensitive to operating conditions, the bus model presented can be used as a plant model for controller development. Beyond this, the lifecycle and potential degradation and durability of the FC can be understood; important when optimizing control systems and strategies. Finally, such a model also gives an accurate picture of the potential range and performance of the vehicle, useful in validating a design's effectiveness prior to prototyping.

2 Modelling

The bus model described in this paper was created as an example model for the Vehicle Demos library from Claytex. This library is a showcase, demonstrating how the VeSyMA suite can be interfaced with external third-party libraries (Hammond-Scott and Dempsey, 2018).

In this case, the bus model was created from components and templates in the VeSyMA library, whilst integrating a cabin and HVAC model from the Thermal Systems library from TLK, along with fuel cell components and stack model from the Hydrogen library by Dassault Systèmes. Kormann and Krüger (2019) have elaborated on some of the principles underpinning the Hydrogen library.

2.1 Bus Model

In FCEV applications, the fuel cell itself is mounted in a series configuration; primarily used to replenish a drive battery, which ultimately delivers electricity to the drive motors. Applications of FC technology to buses seem to follow the same logic, with both Tata Motors (Yogesha et al, 2019) and Toyota (Ogawa et al, 2019) producing buses of this type for use in transit scenarios.

Ogawa (2019) describes how originally in 2002, Toyota produced a FCEV bus using a single fuel cell coupled to a single battery, to drive a single motor unit. Sugiura (2016), describes a dual fuel cell bus, where a single battery per fuel cell was used. Later in 2019, a second-generation FCEV bus was presented by Toyota, this time featuring twin fuel cells allied to a pair of batteries in this case. Yogesha (2019) indicates that Tata Motors have taken another approach, coupling a single fuel cell to dual drive batteries and motors units.

Deciding on the optimal layout requires balancing, component mass, size, packaging ability, thermal management, and performance. Twin stacks and batteries would be easier to package, although potentially less efficient. Dual batteries could be a solution to the packaging problem. Simulation is the perfect tool to evaluate this before committing resources and effort to prototyping. As ultimate performance was not the goal, a simple layout of a single fuel cell coupled to a single drive battery was chosen.

Whilst the bus model chassis itself has full multibody capabilities; the purpose of this exercise was to investigate the longitudinal performance of the vehicle power train. Therefore, the principles established in the VeSyMA library simplifying the vehicle dynamic degrees of freedom (DOF) were followed. This means the suspension utilized was rigid as the bus traversed a perfectly flat road model, with the total motion of the model constrained to 3DOF (translational x, rotational y and translational z) to improve simulation performance. Similarly, tyre modelling was limited to linear longitudinal slip, of a 26570/R19.5 size.

Of note, the bus model featured 3 axles, as is commonly encountered with bus type vehicles, one at the front and two at the rear. Drive was supplied directly to the middle (first rear) axle through an ideal open differential model. A 550Nm mapped motor model was used which included the mass and inertia of the stator and core. Finally, a 130kWh idealized battery model was deployed, with varying output voltage and a fixed internal resistance. It is important to note that review of literature suggests that this battery size is too large for a FCEV bus; it is equipped with this battery to enable comparison with the existing pure EV bus model found in the Vehicle Demos library, as presented in section 3 of this paper.

2.2 Fuel Cell Model

The Fuel Cell model is broken down into 6 subcomponents, mimicking the described layouts found in the aforementioned literature. They are:

- Stack
- Anode hydrogen supply system
- Cathode air supply system
- Cooling system
- Boost Converter Circuit
- Control system

Each subsystem occupies a single model slot at the top level of the bus vehicle model. This continues the principle of component "plug and play" replaceability established in the VeSyMA library. All fluid modelling components used in these models are taken from the Hydrogen library; valve and pipe models are included, featuring representative pressure losses across them. Note, thermal rejection to surrounding components from transport elements are neglected in this study, but the model is equipped to include those effects if desired by the user. These effects were neglected in this study owing to the lack of a representative system to base the model upon.

At the heart of the Fuel Cell model (Figure 2) is the stack model. Taken from the Hydrogen library, the stack used is a Proton-Electron Membrane (PEM) parameterised with a polarization curve from 125KW PowerCell S3 Fuel Cell stack. Defined in the Hydrogen Library as a *stackWithCooling_DetailedMembrane*, this stack model simulates the current generation of the stack, fully dependent upon the temperature and pressure of the reactants in the Anode and the Cathode. The effect of humidity on the stack performance is omitted in this model. A thermal model built into the stack enables the effect of fluid cooling to be incorporated into the performance of the stack. Moving

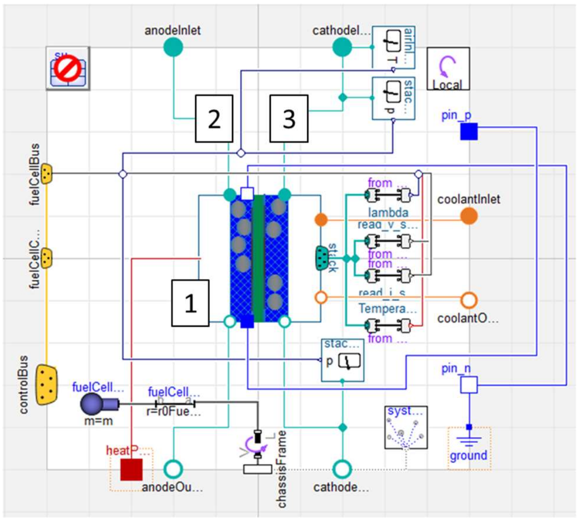


Figure 2 - Fuel Cell stack model. The Electrochemical stack model is deployed with signal routing, connector interface and mass properties, enabling it to feature in the multibody bus model. Note: stack (1), anode (2) and cathode (3).

onto the hydrogen supply system for the stack anode (Figure 3), hydrogen flow is modelled from a fuel tank through a recirculation loop with the anode. Supply pressure is maintained via a valve dependent upon a control signal from the controller model. Hydrogen passes through a humidifier before entering the recirculation loop. To recirculate the hydrogen, a ThomasGardner 907ZC18 pump performance map is used, upscaled 20x to meet the demand of a more powerful stack. Finally, a purge valve on the exit of the loop is controlled to maintain the mass fraction of oxygen in the anode fluid to be less than 0.1.

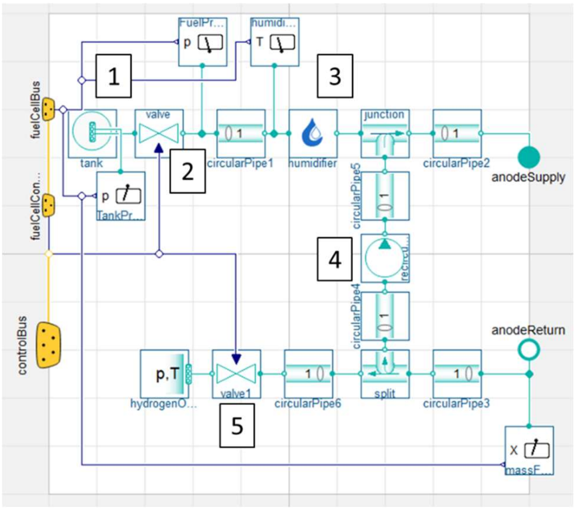


Figure 3 - Hydrogen supply model with recirculation system. Note: hydrogen tank (1), fuel pressure valve (2), humidifier (3), recirculation pump (4) and purge valve (5).

For the air supply system to the stack cathode (Figure 4), a similar modelling philosophy is employed. Here, a model of a Celeroton CT17 700 compressor with efficiency and pressure ratio map is used to compress ambient air; compressor speed is driven via a control signal from the controller. Upon exiting the compressor, the air moves through a heat-exchanger (effectiveness-NTU method) intercooler, to reject heat generated from compression to a 1,2 Propylene Glycol 47% water mix coolant flow, controlled via a command signal. Compressed, cooled air passes through a humidifier before entering the cathode. After passing through the stack, the exhaust flows through a water extractor. This has been included to account for the pressure drop across it due to this commonly used component. Finally, the pressure across the stack is governed by an exhaust valve, once again driven via command signal from the controller.

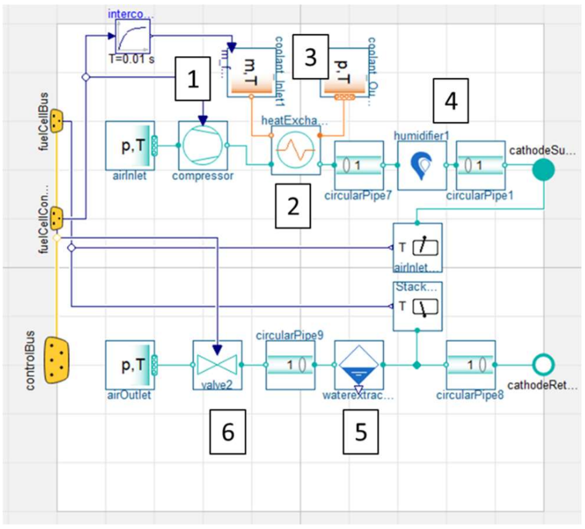


Figure 4 - Air supply system. Note the heat exchanger used to cool the oxidant feed post compression. As the stack model incorporates thermal effects regarding reaction suitibility, then a temperature correct oxidant feed is required. Note: air compressor (1), heat exchanger (2), coolant supply (3), humidifier (4), water extractor (5), exhaust valve (6)

In comparison to other elements of the Fuel Cell model, the cooling and electrical components are simplified. A 1,2 Propylene Glycol 47% water mix coolant flow is specified by the controller model. Flow is regulated to maintain the stack at a specific temperature level. No pumping dynamics or losses are included at this time. Voltage generated by the stack is scaled via a constant efficiency DC/DC boost converter to step up the voltage delivered to the drive battery. A demanded current is used to control the stack output using this component.

The last element of the fuel cell model is the controller responsible for governing the system. This has been broken down into 5 subsystems; one each for the cooling, electrical, air and fuel subsystem models described above, and a state controller. A set of logic gates based on the stack temperature, fuel tank pressure, stack current, stack voltage and battery state of charge determine the stack's state within a rudimentary state machine with 3 modes; "startup", "normal running" and "shut down". This has been designed to enable the stack to enter a "shut down" mode if it exceeds safe operating conditions, fuel supply is diminished, or the battery exceeds a maximum charge threshold to preserve fuel supply. At the time of writing the controller has only been tested in the normal running state.

In terms of actual control signals, a PID approach is used. Cooling demand for intercooler is driven by PID with a target of 50°C, with the stack system having a target of 45°C. A 2D lookup table, dependent on driver torque demand and battery state of charge defines the stack current demand. Intake air compressor is driven by a scalable lambda value of the stack; lambda being the ratio of provided to used oxygen in the stack. Cathode exhaust valve opening is controlled relative to the pressure differential across the inlet/outlet from the stack, with a target difference of 0.5bar. Fuel supply pressure from the tank is maintained at 3.2bar during normal running conditions. The hydrogen purge valve is modulated to keep the mass fraction of oxygen in the anode side of the stack below 0.1.

2.3 Cabin and HVAC Model

The cabin model (Figure 5) is a multi-zone cabin model with partitions including glazing, solid partitions, and internal furniture. The cabin/compartment model is split into front, middle and rear zones and can have a variable number of passengers within it. The orientation or bearing of the compartment influences the angle at which the solar radiation hits the external partitions which affects the thermal loading on the partitions and interior of the compartment.

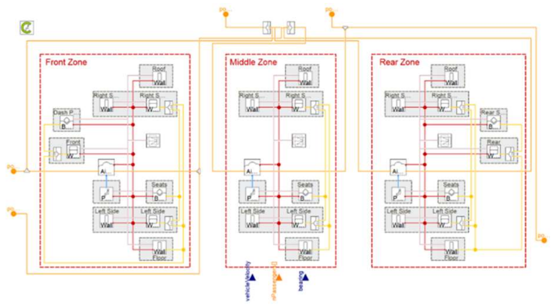


Figure 5 - Multi-zone cabin model with unique elements, such as furniture and glazing properties. The front has a large frontal windshield and dashboard thermal mass, the middle zone has opening doors.

The HVAC model (Figure 6) uses an electrically driven compressor to pump the R134a refrigerant around the two-evaporator, one-condenser system. The electric motor for the compressor uses electrical power from the traction battery via a voltage regulator. Each evaporator branch has its own thermal expansion valve which is independently controlled to achieve a desired superheat value of the refrigerant flowing out of each evaporator.

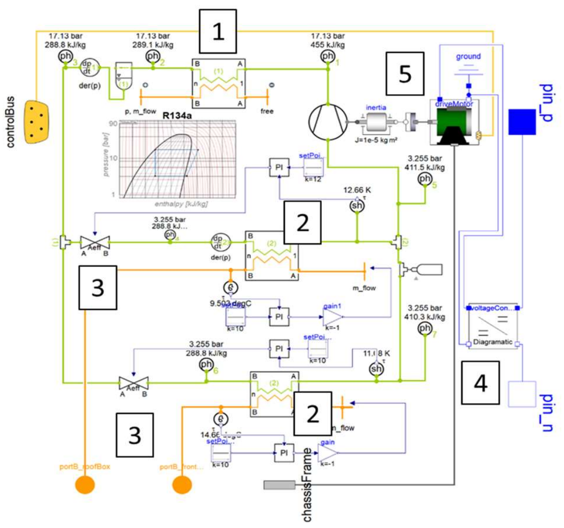


Figure 6 - HVAC model. Note the individual condenser models. Note: condenser (1), evaporators (2), expansion valves (3), voltage regulator (4) and motor (5).

3 Testing & Evaluation

As this paper is concerned with the demonstration of the functioning of the Hydrogen library, Thermal Systems library and VeSyMA, control parameterisation has been geared towards exercising the model rather than replicating a realistic usage. The FCEV bus results are also compared to the EV only version of this bus model found in the Vehicle Demos library, identical as figure 1 minus the FC; to make the comparison valid, the EV bus was also equipped with the same 130KW battery model as the FCEV.

3.1 Drive Cycle Scenario

The Standardized On Road Test (SORT) suburban drive cycle for bus applications is used, as per Figure 7.

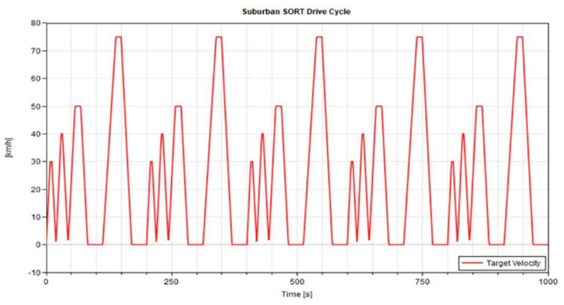


Figure 7 - SORT Suburban drive cycle used to test the bus model.

3.2 Results

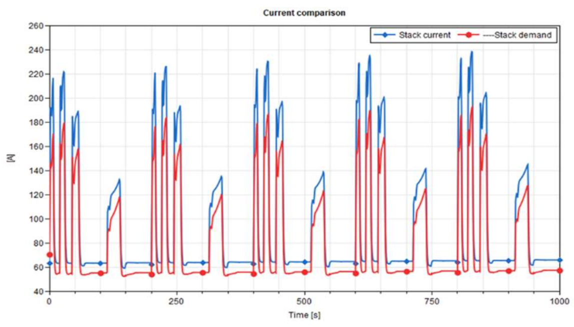


Figure 8 - Current comparison between demand and result. The modulus of the demand current was used for comparisons sake. Blue is stack current, red stack demand.

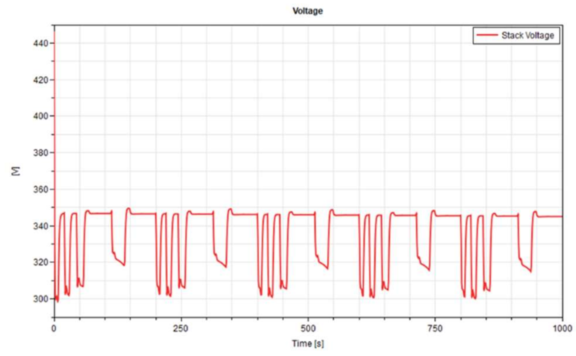


Figure 9 - Voltage produced by the stack.

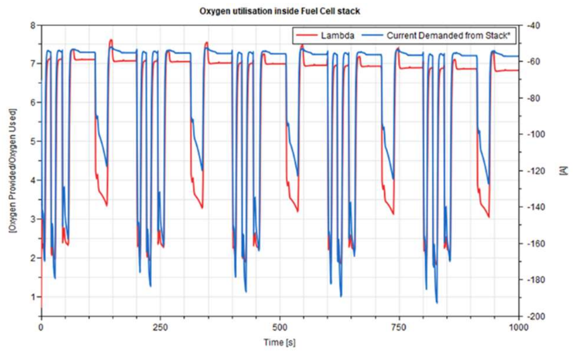


Figure 10 - Oxidant usage in the stack cathode. Red is Lambda, blue the current demand from the stack.

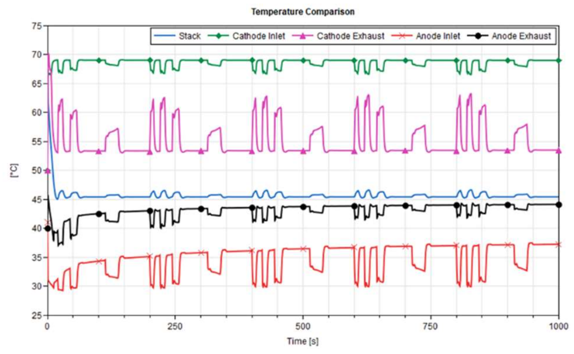


Figure 11 - Temperature comparison across the stack. General stack temperature is blue, the cathode inlet green, cathode exhaust magenta, anode inlet orange and the exhaust black.

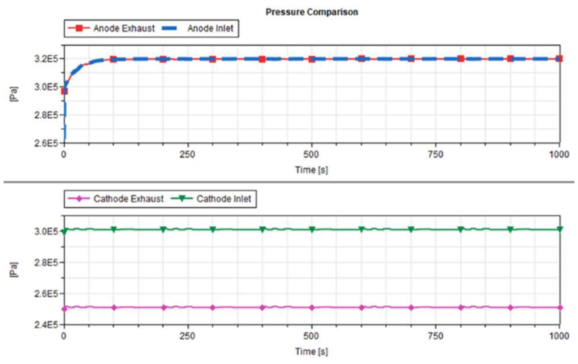


Figure 12 - Pressure comparison across the stack. Anode exhaust is red and the inlet blue; cathode exhaust is magenta and the inlet green.

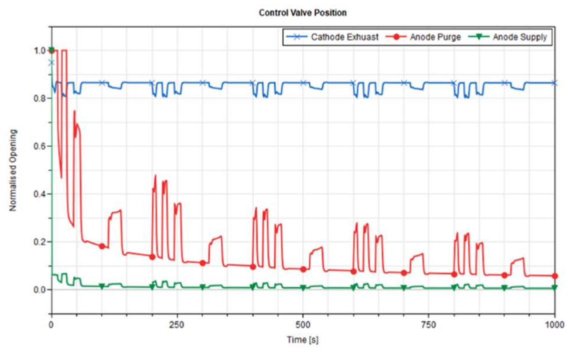


Figure 13 - Control valve position. Large initial transients to steady state can be corrected with improvements to the stack initial conditions. Blue is the cathode exhaust valve, red the anode exhaust valve and green the anode supply valve.

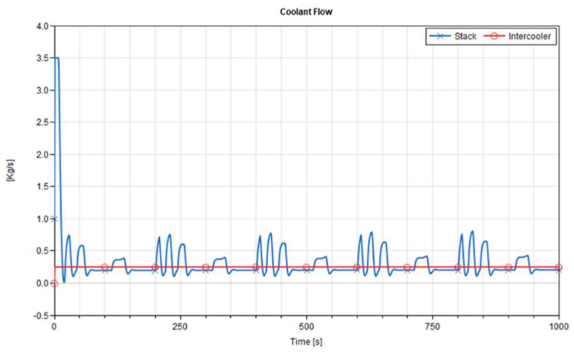


Figure 14 - Coolant flow into the stack (blue) and the intercooler (red).

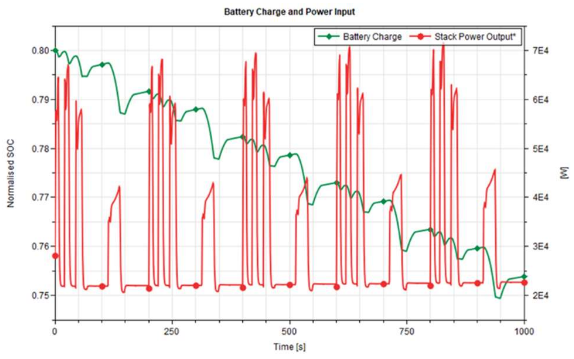


Figure 15 - Battery power input (red) and effect on state of charge (green).

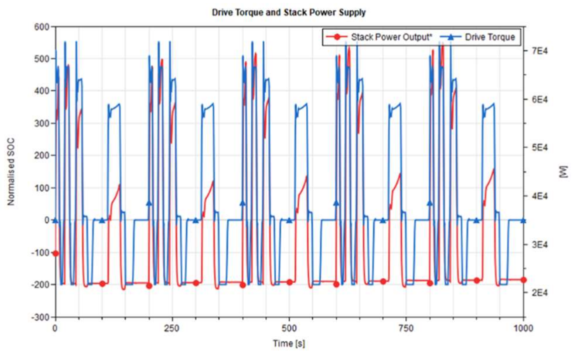


Figure 16 - Stack power output (red) relationship to drive torque (blue).

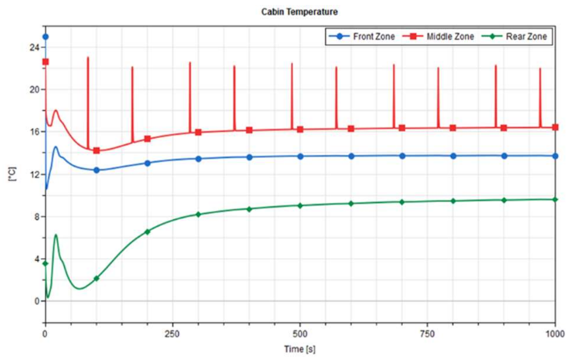


Figure 17 - Cabin temperature in each of the 3 zones; front (blue), middle (red) and rear (green).

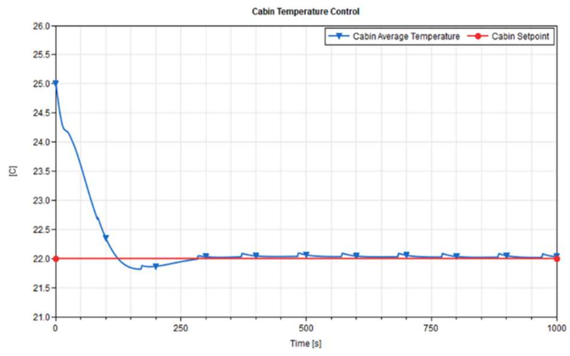


Figure 18 - Temperature control of the cabin. Cabin temperature is blue, the setpoint in red.

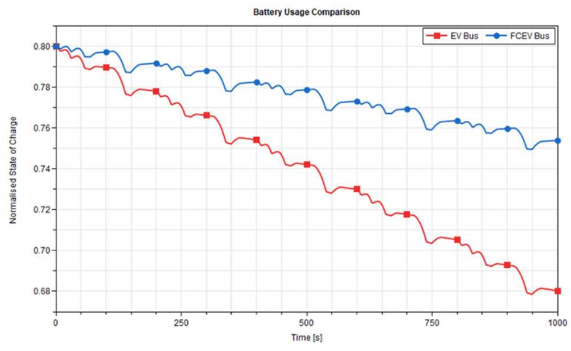


Figure 19 - Comparison between the SOC of the battery in comparable FCEV (blue) and BEV (red) bus models.

3.3 Discussion of results

As the SORT drive cycle was specifically designed for evaluating bus driveline performance, it gives some interesting insights into the FCEV Bus model. Immediately from Figure 8, the fuel cell is responding well to the current demanded from it. There is little response lag, suggesting the ancillary and control systems are working well and keeping the fuel cell in a desired operational window. Fast response to current reduction indicates that the choice to control the fuel cell via current demand was valid, as there is no delay in current reduction from the fuel cell. Detail around the peaks of the current response mirroring the demand indicates the fidelity of the model being used, able to react swiftly. Such detail would be important when trying the evaluate the total range/fuel consumption of the cell. It must be noted however, that the fuel cell is exceeding the current demand; this can likely be remedied with improved controller parameterisation. Voltage time history in Figure 9 supports the conclusions drawn from Figure 8, also indicating there is a slight overshoot in fuel cell voltage as the current demand is removed. Fidelity is further demonstrated by Figure 10, with fluctuations in current demand evident in the oxygen concentration in the cathode. Lambda spikes to a higher value than the steady state after current demand is reduced suggesting a degree of actuator hysteresis, likely in the compressor control.

Comparing the temperature of the stack itself and the gaseous mixtures in the inlet/exhaust of the stack anode and cathode in Figure 11 indicates a global level of settling of temperatures to a steady state, most likely because of unoptimized simulation start values. Once again, they fluctuate with demand.

Generally, we see heat flows from the cathode to the anode side, with the cathode exhaust losing temperature and the anode exhaust gaining it relative to their respective intakes.

Interestingly, the instantaneous temperature fluctuations found in the cathode exhaust relative to the inlet are lessened in the anode. One could deduce that this could be a result of the differing thermal inertias of the gaseous mixtures themselves, although a more likely explanation is that the cathode side features an exhaust valve to atmosphere; the anode system features a hydrogen recirculation system and a purge valve. As the cathode system is more sensitive to the volumetric change, comparatively the cathode side gas will be subject to a larger temperature increase due to compression as the valve closes.

Cross referencing with Figure 13 supports this hypothesis, as both the anode purge and the cathode exhaust valves are overshooting the target somewhat, but it is only the cathode gases experiencing a momentary temperature spike. Essentially, the cathode exhaust valve closes too much and then opens a little as the system dynamics settle. Tuning of the controllers will remedy this.

Owing to a greater thermal inertia of the stack itself, overall stack temperature fluctuations are more muted than the anode and cathode gases. This is entirely expected.

Figure 12 indicates that the 3 control valves, cathode exhaust, anode purge and hydrogen supply, are behaving broadly as they should to manage the pressure within the stack. Pressure fluctuations are minimal, but a cursory glance at Figure 13 does reveal some overshoot and further investigations reveals some interesting insights about the system dynamics. Hydrogen supply is increased due to demand; the purge valve also opens more during demand events, indicative of a greater oxygen concentration in the anode as more reaction takes place. The action of the cathode exhaust valve corroborates this, as it is closing slightly during demand events; this will be to maintain pressure across the cathode, indicating a pressure drop of some kind. Therefore, it appears that there is gaseous loss towards the anode. The temperature gradient identified previously from the cathode inlet to the anode exhaust supports a theory of gaseous loss towards the anode in this manner.

Figure 14 indicates an opportunity to improve the air compressor control. The constant flow of coolant into the intercooler and the stability of the cathode inlet temperature (during steady state low current demand) belies a compressor which is running at a constant speed. This explains why the Lambda value spikes when there is little current demand; instead of reducing speed when there is little load, it continues at a constant speed. Stack coolant flow positively correlates with greater heat generation due to demand.

Reviewing Figure 15 and 16, we can make some conclusions as to how the overall logic of the electrical controller, responsible for the current demand from the stack, is functioning. Current demand is rising and falling rapidly; this is due to the controller being parameterised to only demand current continuously when the battery SOC drops below a certain threshold. Therefore, as Figure 16 proves, the fuel cell is only engaging fully in heavy tractive events. This is an expected outcome, as the deployment strategy was inspired by Ogawa (2019), who proposed the use of a Fuel Cell demand override during strenuous tractive events, to reduce battery size. It should be noted that the maximum power produced by the stack does not reach the stated maximum rating. Improvements to the controller logic given the battery SOC are likely to remedy this, by improving the running conditions of the stack.

Figures 17 and 18 indicate that the cabin thermal and HVAC models are performing as intended. The air inlet compromising the cabin doors is connected to the middle zone of the cabin, as described in Figure 5. With an outside air temperature of 30°C, this means a volume of warm air is admitted to the cabin; the HVAC system, has a setpoint of 22°C. We see that the middle zone temperature spikes when the cabin doors are opened, letting warmer air into the volume. Whilst this causes the middle section to increase in temperature alone versus the front and rear, Figure 18 demonstrates how it increases the average temperature. Action of the HVAC system then returns the temperature to the setpoint.

Finally, Figure 19 demonstrates the positive effect of the fuel cell on the battery charge, versus a comparable EV bus with no fuel cell. Whilst it is true that the battery deployed in the FCEV is much larger in capacity relative to numbers disclosed in published literature by Toyota and Tata, it nevertheless serves as a reminder of the advantages of a FCEV versus a pure EV in such situations.

4 Conclusions

Overall, it can be concluded that the FCEV bus model presented in this paper is functioning in a valid way as a proof of concept. Useful observations regarding the interplay of the control system, which has 3 elements (hydrogen supply to the anode, air supply to the cathode and current demand) to manage can be made. We can see the model is sensitive to small control changes, in a logical way. Such sensitivity is important, as it encompasses all the systemic nonlinearities that a controller or physical system must manage. This gives confidence to using such a model in place of a real-life prototype. As discussed, there is scope to further improve the simulation with optimization of the control parameters. Nevertheless, it has been useful to exercise the system and observe the trends and dynamics.

4.1 Further Work

The most immediate improvement that could be made to the model would be to deploy the *stack_MembraneDetailedHumidity* stack model, to include the effect of humidity directly on the current produced by the fuel cell. One advantage of a detailed model such as the one presented in the paper is the ability to study the coupled thermodynamic effects; all the transport components such as pipes in this model have the capability to include thermal heat transfer to the boundary. Coupled phenomena regarding the design of the bus could be studied. Such a case would be the effect of the HVAC system running during hot weather, with the addendum potential for a greater current demand from fuel cell. Preconditioning studies could also be conducted, a feature which literature shows potential for improving the range of conditions a fuel cell can operate. Cold conditions specifically are of interest, as they traditionally impact BEV performance greatly. Further studies into fuel cell management and lifecycle could be conducted, especially the case of dosing water management, if physical water capture, use and recycling is added to the model. Further insight into the specific infrastructural requirements needed to support Hydrogen buses could be understood by doing this.

Acknowledgements

The authors would like to thank Imke Lisa Krüger and Markus Andres of Dassault Systèmes for their assistance in using the Hydrogen library components in this model.

References

L. Andaloro, S. Micari, G. Napoli, A. Polimeni and V. Antonucci (2016). A hybrid electric fuel cell minibus: drive test. *EVS29 International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium.* World Electric Vehicle Journal. 2016, 8(1), 131-138; https://doi.org/10.3390/wevj8010131

L. Eudy and M. Post. 2021. Fuel Cell Buses in U.S.Transit Fleets: Current Status 2020. Golden, CO: National Renewable Energy Laboratory. NREL/TP-5400- 75583. https://www.nrel.gov/docs/fy21osti/75583.pdf.

O. Gröger, H. A. Gasteiger, and JP. Suchsland. Review – Electromobility: Batteries or Fuel Cells? *Journal of the Electochemical Society.* 162 (14) A2605-A2622 2015. DOI: 10.1149/2.0211514jes

H. Hammond-Scott and M. Dempsey (2018). Vehicle Systems Modelling and Analysis (VeSyMA) Platform. *Proceedings of the 2nd Japanese Modelica Conference, 2018*. DOI: 10.3384/ecp18148

M. Kormann and I. L. Krüger (2019). Application of a Real Gas Model by van der Waals for a Hydrogen Tank Filling Process. *Proceedings of the 13th International Modelica Conference, 2019.* DOI: 10.3384/ecp19157665

Luxfer Gas Cylinders. 2022. *Hydrogen Buses.* Online. Accessed 1st August 2022. Available at: https://www.luxfercylinders.com/news/hydrogen-buses#:~:text=On%20average%2C%20a%20hydrogen%20bus,ride%20is%20smoother%20and%20quieter.

T. Ogawa, K. Umayahara and Y. Ikogi (2019). Development of Fuel Cell (FC) System for New Generation FC Bus. *SAE International Journal Advances & Current Practice in Mobility* 1(2):782-786, 2019, doi:10.4271/2019-01-0372.

T. Sugiura, A. Tanida, and K. Tamura (2016). *Efficiency Improvement of Boost Converter for Fuel Cell Bus by Silicon Carbide Diodes. SAE International Journal Alternative Powertrains.* 5(2):2016, doi:10.4271/2016-01-1234.

United Nations. 2022. *UN Highlights urgent need to tackle impact of likely electric car battery production boom*. Accessed 1st August 2022. Available at: https://news.un.org/en/story/2020/06/1067272

D. C. Vock. 2019. *Batteries or Hydrogen? Cities Weigh Making Buses Electric.* Online. Accessed 1st August 2022. Available at: https://www.govtech.com/products/batteries-or-hydrogen-cities-weigh-the-best-way-for-buses-to-go-electric.html

S.A. Yogesha, S. Brahmbhatt, M. Raja, S. Arikapudi, B. Bhut and V. Jalkumar (2019). Development of Hydrogen Fuel Cell Bus Technology for Urban Transport in India. *SAE Technical Paper 2019-26-0092.* 2019, doi:10.4271/2019-26-0092.

slPCMlib: A Modelica Library for the Prediction of Effective Thermal Material Properties of Solid/Liquid Phase Change Materials (PCM)

Tilman Barz   Aurelien Bres   Johann Emhofer

Center for Energy, AIT Austrian Institute of Technology GmbH, Giefingasse 2, 1210 Vienna, Austria
tilman.barz@ait.ac.at

Abstract

slPCMlib predicts the effective thermal properties of solid/liquid phase change materials (PCM) showing a non-isothermal phase transition behavior. The effective properties are valid over the PCM functional temperature range where latent heat is absorbed and released. Different phenomenological phase transition models are implemented to account for temperature shifts in latent transition changes, e.g. due to multi-step transitions and thermal hysteresis. The library currently contains generic PCM and specific commercial paraffin-based and hydrated salt-based PCM (media). Its purpose is the analysis of partial and complete melting and solidification processes relevant for engineering applications, such as the design of PCM-enhanced building components.

Keywords: solid/liquid phase transition, thermal hysteresis, phase change material (PCM)

1 Introduction

Solid/liquid phase change materials (PCM), such as salt hydrates, paraffin waxes, fatty acids and eutectics of organic and non-organic compounds are used for storing thermal energy (heat or cold) and/or to regulate temperatures, in a small temperature range with high efficiency. Numerous applications are reported where PCM is incorporated in building envelopes (Al-Yasiri and Szabó, 2021; Kuznik et al., 2011).

***Ideal* versus *real* phase change behavior:** While *ideal* PCM show an isothermal phase change behavior, many *real* (commercial) PCM show a non-isothermal phase change behavior: they melt and solidify over an extended temperature range. Moreover, a large part of the PCM available for building applications shows thermal hysteresis (including supercooling), which can be measured by a (temperature) shift in the enthalpy curves for heating and cooling. This phenomenon additionally extends the temperature range where the latent heat is absorbed and released. Accordingly Kośny (2015) introduce the “PCM functional temperature range”, which starts at the lowest temperature limit of the solidification process and ends at the highest temperature of the melting process. Figure 1 exemplifies different phase transition behavior of *ideal*

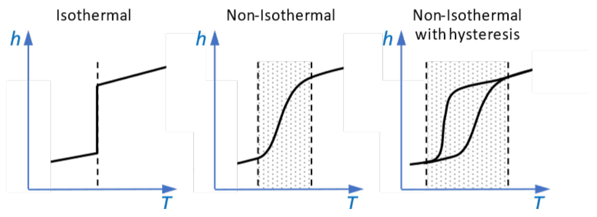


Figure 1. Enthalpy as a function of temperature for the case of an isothermal transition, a non-isothermal transition, and a non-isothermal transition with hysteresis.

and *real* PCM.

The thermo-physical and rheological properties of PCM are usually characterized not only by the calorific properties, but also by the thermal conductivity in the solid and liquid phases, viscosity of liquid PCM and density as a function of temperature (Kośny, 2015). Considering *real* PCM these properties also change over the phase transition temperature range.

Phenomenological phase transition models: Although it has been recognized by many research groups that complex phase transition phenomena in *real* materials can have a significant impact on PCM performance, only few numerical models have been developed which are able to represent specific effects such as hysteresis and supercooling (Kośny, 2015).

Kośny (2015) review PCM modeling algorithms commonly used in building energy and hygrothermal software. The phase transition behavior is mostly characterized by a single enthalpy-temperature, or apparent heat capacity-temperature curve which can be obtained e.g. from caloric measurements. Corresponding models are purely data-driven, phenomenological models, which can be easily applied for the analysis of PCM showing non-isothermal and rate-independent phase transition phenomena (Barz et al., 2019). However, in most software the parametrization of the curve’s shape is usually restricted. Only few software offer separate curves for melting and freezing (to account for thermal hysteresis).

Recently, different phenomenological thermal hysteresis models have been proposed: The so-called “curve track” model uses different curves for complete melting or solidification processes, e.g. (Michel et al., 2017; Biswas et al., 2018; Moreles et al., 2018; Filonenko et al., 2020). The so-called "curve switch" model is an extension allowing minor loops relevant for incomplete (or interrupted) phase transitions, e.g. (Bony and Citherlet, 2007; Rose et al., 2009; Buonomano and Guarino, 2020; Goia et al., 2018; Hu and Heiselberg, 2018). Another extension is the so-called "curve scale" model, e.g. (Barz and Sommer, 2018; Barz et al., 2019; Barz, 2021; Lizana et al., 2021).

Modelling heat transfer in PCM: There exist different numerical modeling approaches to deal with the moving boundary between phases during melting and solidification. Considering *real* PCM, it seems most reasonable to adopt the so-called weak formulation, specifically the enthalpy method and apparent heat capacity method. Here, the explicit treatment of a moving interface is avoided, and instead, a mushy transition zone between the two phases is considered where effective enthalpy- or apparent heat capacity-temperature curves are applied, see Voller et al. (1990) for details.

For a recent literature review on Modelica implementations of numerical models for heat transfer in *ideal* and *real* PCM we refer to Helmns et al. (2021). As an example, Helmns et al. (2021) uses the enthalpy method as implemented in the Modelica Buildings Library (Wetter et al., 2014) for the development of a component model of a thermal energy storage with PCM. The heat conduction equation is formulated with the enthalpy (or internal energy) as dependent variable. The temperature is modeled as a piecewise linear function of enthalpy (inverse enthalpy-temperature relation), which is represented (approximately) by a cubic hermite spline interpolation. The enthalpy method allows for the solution of heat conduction problems in *real* and *ideal* PCM. In the Modelica Buildings Library generic PCM with an (almost) isothermal behavior use small phase transition temperature ranges of 0.02 K.

Leonhardt and Müller (2009); Halimov et al. (2019) use the apparent heat capacity method and extend AixLib, a Modelica model library for building performance simulations, by heat capacity-temperature relations for *real* PCM. Different curve shapes were experimentally validated for a commercial paraffin-based PCM using alternative temperature-dependent continuous ansatz functions, such as arctangent function (Halimov et al., 2019).

This contribution: A new library slPCMlib is presented which predicts effective properties of *real* PCM. It contains the above mentioned phenomenological phase transition (hysteresis) models as well as generic and specific PCM (media) for which the phase transition behavior was identified from caloric measurement data. Examples for conduction dominated heat transfer in PCM are presented adopting the apparent heat capacity method.

2 Effective material properties

The following assumptions are taken for modeling effective PCM properties:

- There are only two phases (two-phase model): a solid and a liquid phase.
- Phase transitions are induced by temperature and are independent of pressure.
- Phase transitions extend over a temperature range (non-isothermal phase transitions) and are continuous.
- Within the phase transition temperature range the solid and liquid phases coexist as a homogenous mixture (macroscopic view). The material is then in a semi-solid or semi-liquid state which produces a mushy zone in the PCM domain.
- Properties of the mushy state are local effective (also apparent) mixture properties, which are defined by a weighting of contributions from solid and liquid phases. The weighting is based on the phase change progress, i.e. the mass (or volume) phase fraction.

The effective enthalpy $h(T)$, density $\rho(T)$ and thermal conductivity $\lambda(T)$ are calculated as[1]:

$$h(T) = (1-\xi(T))\, h^s(T) + \xi(T)\, h^l(T) \tag{1a}$$

$$\rho(T) = (1-\phi(T))\, \rho^s(T) + \phi(T)\, \rho^l(T) \tag{1b}$$

$$\lambda(T) = (1-\phi(T))\, \lambda^s(T) + \phi(T)\, \lambda^l(T) \tag{1c}$$

where $\xi(T)$ and $\phi(T)$ are the liquid mass and liquid volume phase fraction, respectively[2]. Their relation is:

$$\phi(T) = \frac{\xi(T)}{\xi(T) + (1-\xi(T))\frac{\rho^l(T)}{\rho^s(T)}} \tag{2}$$

The apparent specific heat capacity $\tilde{c}(T) = dh(T)/dT$ reads:

$$\tilde{c}(T) = \underbrace{(1-\xi(T))\, c_p^s(T) + \xi(T)\, c_p^l(T)}_{\text{baseline, } c_{\mathrm{BL}}(T)} + \underbrace{\frac{d\xi}{dT}\left(h^l(T) - h^s(T)\right)}_{\text{peak function}} \tag{3}$$

It is assumed that the properties of the single phases ρ^l, ρ^s, λ^l, λ^s, and c_p^l, c_p^s are available. The difference in solid and liquid enthalpies h^l, h^s defines the phase transition enthalpy. For non-isothermal transitions there exist different approaches for the calculation. They are linked with the method for determining the phase transition function $\xi(T)$. The determination of $\xi(T)$ and $h^l(T)$, $h^s(T)$ are discussed in the following.

[1]While viscosity is also an important property it is not considered as effective variable here. The reason is that in numerical modeling viscosity in the solid is usually either neglected or artificially increased to ensure zero velocity fields in the solid phase. However, liquid viscosity might be considered by extending basic PCM properties in `slPCMlib.Media` discussed in Section 4.1.

[2]For better readability the symbols ξ and ϕ have no superscript (l) to indicate liquid phase fraction. Obviously, considering two components the solid phase fractions read $1-\xi(T)$ and $1-\phi(T)$.

2.1 Phase transition functions for heating and cooling

Phase transition functions describe the phase change progress for complete transitions during heating (complete melting with $dT/dt > 0$), and for complete transitions during cooling (complete solidification with $dT/dt < 0$), respectively. Note that, because the behavior might be different for melting and solidification (thermal hysteresis), two different transition functions are considered:

$$\begin{aligned} \xi^H &= \xi(T) \quad \text{for heating} \\ \xi^C &= \xi(T) \quad \text{for cooling} \end{aligned} \tag{4}$$

It is assumed that: The transition functions:

- depend on temperature, are differentiable and monotonically increase with rising temperature.
- realize a transition from $\xi = 0$ (solid) to $\xi = 1$ (liquid).
- are shifted in temperature (thermal hysteresis) and do not intersect: $\xi^H(T) \leq \xi^C(T)\ \forall\, T$.

The limits of the phase transition temperature are defined as:

$$\begin{aligned} T_{\min} &= \max\{T \,|\, \xi(T) = 0\} \\ T_{\max} &= \min\{T \,|\, \xi(T) = 1\} \end{aligned} \tag{5}$$

Because of the assumption above, in case of thermal hysteresis $T_{\min}$ corresponds to ξ^C, and $T_{\max}$ to ξ^H. This means that the phase transition temperature range (also PCM functional temperature range) starts at $T_{\min}$ of the solidification process, and ends at $T_{\max}$ of the melting process. Examples for the transition functions are shown in Figure 3 (bottom).

2.2 Determination of phase transition functions and single phase enthalpies

The heat storage capacity of PCM is usually tabulated as scalar values for the phase change enthalpy and melting temperature. For some PCM also apparent heat capacity curves are available, e.g. Figure 2. As pointed out e.g. by Kośny (2015), a *real* PCM with a non-isothermal phase change behavior should be represented by a temperature dependent function, e.g. $h(T)$.

In Differential Scanning Calorimetry (DSC) analysis, which is a standard technique for caloric measurements of PCM, the (*scalar*) phase transition enthalpy Δh_t is usually determined as the area between two curves defined in Equation (3): the apparent (effective) heat capacity $\tilde{c}(T)$, and the baseline heat capacity $c_{\mathrm{BL}}(T)$ (Hemminger and Sarge, 1991).

$$\Delta h_t = \int_{T_{\min}}^{T_{\max}} \left(\tilde{c}(\tau) - c_{\mathrm{BL}}(\tau)\right) d\tau \tag{6}$$

The baseline $c_{\mathrm{BL}}(T)$ connects solid and liquid heat capacities in the phase transition temperature range and is determined by a suitable baseline construction method. After

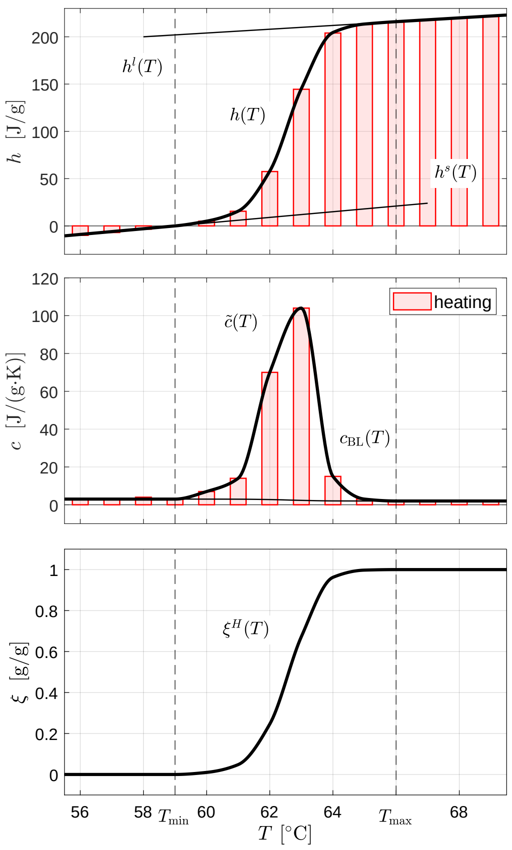


Figure 2. Determination of the phase transition function for heating and the single phase enthalpies from heat capacity data of RT62HC. Middle: The bars depict the data as provided by the PCM manufacturer (Rubitherm Technologies GmbH). The data (partial enthalpies for one Kelvin intervals) was collected using a three-layer-calorimeter. The lines depict the fitted effective heat capacity, see Barz et al. (2020) for details. Top: Derived enthalpy data. Bottom: Derived phase transition function for heating.

substraction of the baseline, the phase transition function is obtained from the cumulative integral over the normalized peak, taking $T_{\min}$ and $T_{\max}$ as integration limits.

For *ideal* PCM with an isothermal phase change behavior, the *scalar* Δh_t in Equation (6) is the transition enthalpy at a reference temperature, i.e. the melting temperature. For *real* PCM (considered in this contribution), Δh_t is *temperature dependent*. The dependence is given by the so-called Kirchhoff equation (or Kirchhoff's Law), which relates the isobaric temperature variation of the phase transition enthalpy to the difference in specific heat capacities at constant pressure (McDonald, 1953):

$$\left.\frac{\partial \Delta h_t}{\partial T}\right|_p = c_p^l - c_p^s \tag{7}$$

The Kirchhoff equation yields a *temperature-dependent* phase transition enthalpy, which is used in Equation (3) in the peak function:

$$\Delta h_t(T) = h^l(T) - h^s(T) \tag{8}$$

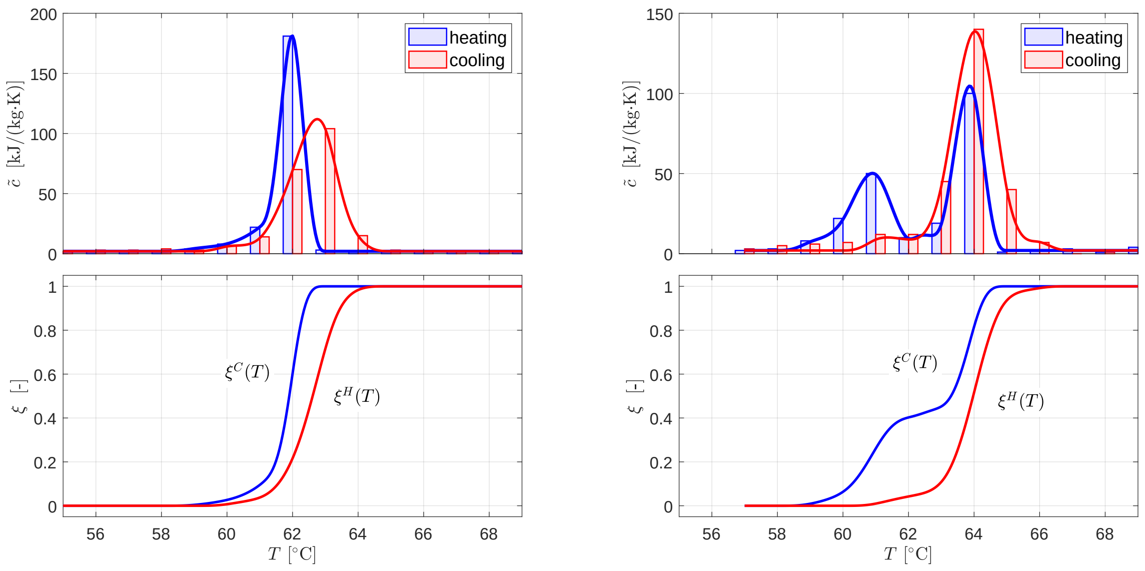


Figure 3. Determination of phase transition functions for heating ξ^H and cooling ξ^C from heat capacity data of Rubitherm RT62HC (left) and RT64HC (right), see Barz et al. (2020) for details. Top: The bars depict heat capacity data provided by the PCM manufacturer (Rubitherm Technologies GmbH). The data (partial enthalpies) were recorded using a three-layer-calorimeter. The lines depict the fitted effective heat capacity. Bottom: Derived phase transition functions.

In slPCMlib the *single phase* solid and liquid enthalpies are defined as:

$$h^s(T) = h^s(T_{\text{ref}}) + \int_{T_{\text{ref}}}^{T} c_p^s(\tau)d\tau \tag{9a}$$

$$h^l(T) = h^l(T_{\max}) + \int_{T_{\max}}^{T} c_p^l(\tau)d\tau \tag{9b}$$

with $T_{\text{ref}} \leq T_{\min}$ and

$$h^s(T_{\text{ref}}) = h_{\text{ref}} \tag{10a}$$

$$h^l(T_{\max}) = h_{\text{ref}} + \int_{T_{\text{ref}}}^{T_{\max}} c_{\text{BL}}(\tau)d\tau + \Delta h_t \tag{10b}$$

To briefly sum up, the following definitions are used:

- Δh_t is the *scalar* phase transition enthalpy for melting of a *real* PCM with a non-isothermal phase change behavior. It is defined for the *melting temperature range* (not the melting temperature), and it is determined from heat capacity data for heating using Equation (6) .
- $\Delta h_t(T) = h^l(T) - h^s(T)$ is the *temperature dependent* phase transition enthalpy. The single phase enthalpies h^l and h^s are computed via Equation (10) for a given *scalar* Δh_t.

Enthalpy $h(T)$ and apparent heat capacity $\tilde{c}(T)$ are calculated considering the temperature dependent transition enthalpy in Equation (8) and Equations (1a) and (3).

Note that, if heat capacity data for melting and solidification is different (thermal hysteresis), then the phase transition functions for melting ξ^H and solidification ξ^C are determined for each data set independently, see Figure 3 for examples. However, the *scalar* phase transition enthalpy Δh_t is obtained from the data for melting. In some cases it might by necessary to adapt the value of Δh_t in order to generate a "best fit" for both data sets.

3 Phase transition models

In addition to the assumptions in Section 2, the following assumption is used for modeling the phase transition behavior for melting and solidification processes:

- Phase transitions are rate-independent (equilibrium model).

It follows that, increased heating or cooling rates lead to faster melting and solidification. However, the rate has no effect on the systems behavior itself. The graphs in the (ξ, T)-plane and the (h, T)-plane are unchanged. For the hysteresis models discussed in the following, this means that also the magnitude of the hysteresis is rate-independent.

3.1 The *melting curve* model

The simplest model (here referred to as *melting curve* model) predicts the evolution of the phase fraction as response to arbitrary changes in temperature using the phase transition function for heating:

$$\xi = \xi^H(T) \tag{11}$$

The model does not account for hysteresis phenomena, see Figure 4 for an example.

3.2 The *curve track* hysteresis model

The *curve track* hysteresis model predicts the evolution of the phase fraction as response to positive or negative changes in temperature T, starting from T_0 to the final value T_f, using one of the following submodels:

$$\xi(T) = \xi^H(T) \qquad \text{if} \quad T_0 = T_{\min} \tag{12a}$$

$$\xi(T) = \xi^C(T) \qquad \text{if} \quad T_0 = T_{\max} \tag{12b}$$

where T_0 is the discrete (piecewise constant) temperature which changes only when T crosses the limits of the phase transition temperature range. At this point the final variable T_f is reached, the process is restarted setting $T_0 = T_f$ and choosing the next submodel. This means that T_0 holds the information which transition (melting or solidification) was completed last, e.g. $T_0 = T_{\min}$ means that the PCM was in the complete solid state and the heating curve $\xi^H(T)$ is currently used.

The *curve track* model is completely defined by $\xi^H(T)$ and $\xi^C(T)$, and the limits $T_{\min}$ and $T_{\max}$. The model is useful for the prediction of complete melting or complete solidification processes. The model is not useful for incomplete phase transition processes. This is because switches between heating and cooling, while the material is still within the phase transition range, do not result in a change (e.g. switch) of the phase transition function, see (Barz et al., 2019). An example for complete and incomplete melting and solidification processes is shown in Figure 4.

3.3 The *curve switch* hysteresis model

The *curve switch* model, first proposed by Bony and Citherlet (2007), extends the *curve track* model for an improved prediction of interrupted phase transitions, i.e. incomplete transitions with switches between heating and cooling. Incomplete transitions are modeled by a straight line between the phase fraction-temperature curves (and enthalpy–temperature curves) for heating and cooling. Following this connecting line realizes the so-called *curve switch*.

The evolution of the phase fraction as response to positive or negative changes in T, starting at T_0 and ending at T_f is described by three submodels for melting, soldification and the curve switch:

$$\xi(T) = \xi^H(T) \qquad \text{if} \quad T_0 = T_{\min} \tag{13a}$$

$$\xi(T) = \xi^C(T) \qquad \text{if} \quad T_0 = T_{\max} \tag{13b}$$

$$\xi(T) = \text{constant} \qquad \text{if} \quad T_{\min} < T_0 < T_{\max} \tag{13c}$$

where, in the same way as in Equation (12), T_0 is the discrete (piecewise constant) temperature indicating which submodel is used. The following conditions can trigger an event (T_f is reached): When T crosses the limits of the phase transition temperature range, then either $T_f = T_{\min}$ or $T_f = T_{\max}$; During melting or solidification with $T_{\min} < T < T_{\max}$, when the temperature rate dT/dt changes the sign, then $T_f = T$ (initiation of curve switch); During the curve switch, when $\xi(T)$ reaches either the curve for heating $\xi(T) = \xi^H(T)$ or cooling $\xi(T) = \xi^C(T)$, then either $T_f = T_{\min}$ or $T_f = T_{\max}$ (finalization of curve switch).

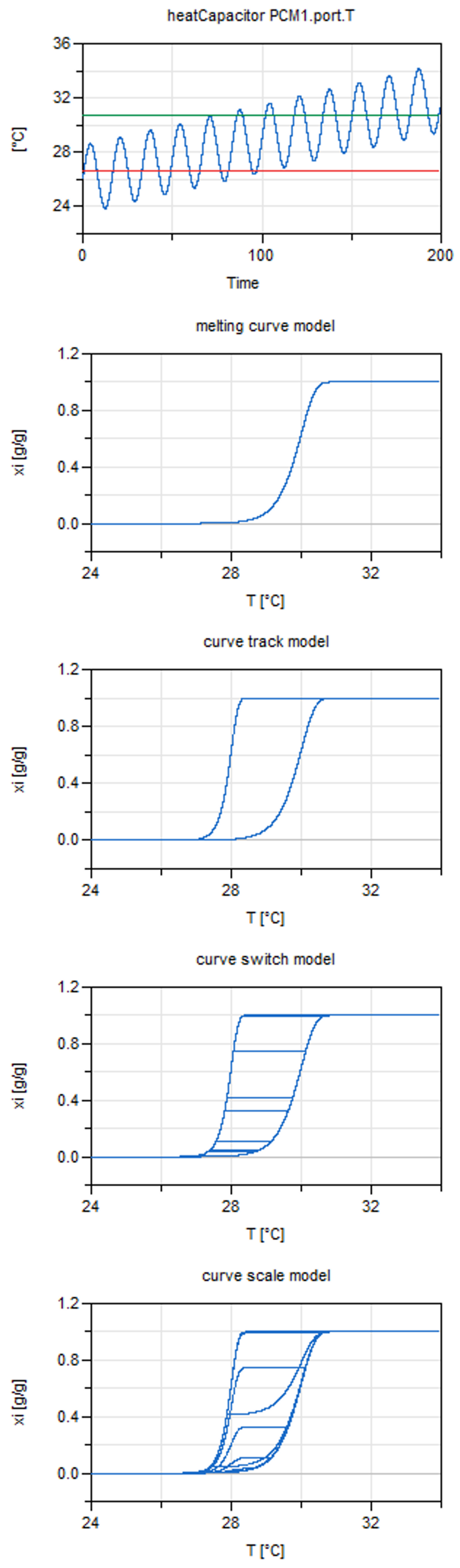


Figure 4. Evolution of the phase fraction as response to sinusoidal temperature variations considering different rate-independent phase transition models. The first subfigure shows the temperature input and the limits of the phase transition temperature range. The following subfigures show the corresponding responses in the (T, ξ)-plane. Note that, since the hysteresis models are static models, time is given in arbitrary unit.

When T_f is reached the process is restarted setting $T_0 = T_f$ and choosing the next submodel.

The *curve switch* model is completely defined by $\xi^H(T)$ and $\xi^C(T)$, and the limits $T_{\min}$ and $T_{\max}$. For complete melting and solidification the model produces the same results as the *curve track* model. An example is shown in Figure 4.

3.4 The *curve scale* hysteresis model

The *curve scale* model, originally introduced by Ivshin and Pence (1994) for the modeling of temperature induced phase transitions and first applied by Barz and Sommer (2018) in the context of solid/liquid PCM, is another extension of the *curve track* model. The model accounts for different hysteresis magnitudes for cycles within the phase transition temperature range and makes use of the temperature history. Major and minor hysteresis loops are constructed by scaling the functions for complete transitions $\xi^H(T)$ and $\xi^C(T)$.

The evolution of the phase fraction as response to a monotonous change in T from a starting value T_0 to the final value T_f is described by two submodels:

$$\xi(T) = 1 - s^{\text{pos}} \cdot \left(1 - \xi^H(T)\right) \quad \text{if} \quad dT/dt \geq 0 \tag{14a}$$

$$\xi(T) = s^{\text{neg}} \cdot \xi^C(T) \quad \text{if} \quad dT/dt < 0 \tag{14b}$$

with the scaling factors:

$$s^{\text{pos}} = \frac{1-\xi_0}{1-\xi^H(T_0)}\,, \qquad s^{\text{neg}} = \frac{\xi_0}{\xi^C(T_0)} \tag{15}$$

where T_0 and $\xi_0 := \xi(T_0)$ denote the initial temperature and phase fraction for time intervals with either increasing or decreasing temperatures. Thus, during monotonically increasing (or decreasing) temperatures all variables in Equation (15) are constant. Their values are updated only at time instants (events) when the sign of dT/dt changes.

An equivalent differential form of the curve scale hysteresis model is obtained by differentiation of Equations (14), (15) with respect to time using the chain rule (Ivshin and Pence, 1994):

$$\frac{d\xi}{dt} = \frac{1-\xi(T)}{1-\xi^H(T)} \cdot \frac{d\xi^H(T)}{dT} \cdot \frac{dT}{dt} \quad \text{if} \quad \frac{dT}{dt} \geq 0 \tag{16a}$$

$$\frac{d\xi}{dt} = \frac{\xi(T)}{\xi^C(T)} \cdot \frac{d\xi^C(T)}{dT} \cdot \frac{dT}{dt} \quad \text{if} \quad \frac{dT}{dt} < 0 \tag{16b}$$

With this modification the discrete (piecewise constant) variables in Equation (15) are replaced by continuous variables. It turns out that the model can be implemented as one differential equation with a discontinuous right hand side.

The *curve scale* model is completely defined by $\xi^H(T)$ and $\xi^C(T)$. For complete melting and solidification it produces the same results as the curve track model. Results for incomplete melting and solidification are shown in Figure 4.

4 Implementation in Modelica

An overview of the packages contained in slPCMlib is given in Figure 5.

4.1 Definition of media

The package `slPCMlib.Media` (see Figure 5) contains *specific* PCM data of commercial organic and inorganic PCM for which solid and liquid properties are tabulated and heat capacity curves are available in the technical data sheets provided by manufacturers. In addition, the package also contains four examples with *generic* PCM data, which can be adapted by the user to match a certain peak shape.

Each medium (PCM) extends the partial package `slPCMlib.interfaces.partialPCM` which contains the basic definition of a medium. Functions for *single phase* solid and liquid densities $\rho^s(T)$, $\rho^l(T)$ and thermal conductivities $\lambda^s(T)$, $\lambda^l(T)$) can be arbitrary functions of temperature. They are defined by a `replaceable partial function`. In contrast, functions for *single phase* heat capacities c_p^l, c_p^s are assumed to be linear functions of temperature: $a + b\cdot(T - T_{\text{ref}})$. Corresponding coefficients a, b, reference temperature T_{ref}, as well as reference enthalpy $h_{\text{ref}} = h(T_{\text{ref}})$, *scalar* phase transition enthalpy Δh_t, and the limits of the phase transition temperature range are defined in a replaceable record called `propData`.

4.1.1 Functions for heating and cooling

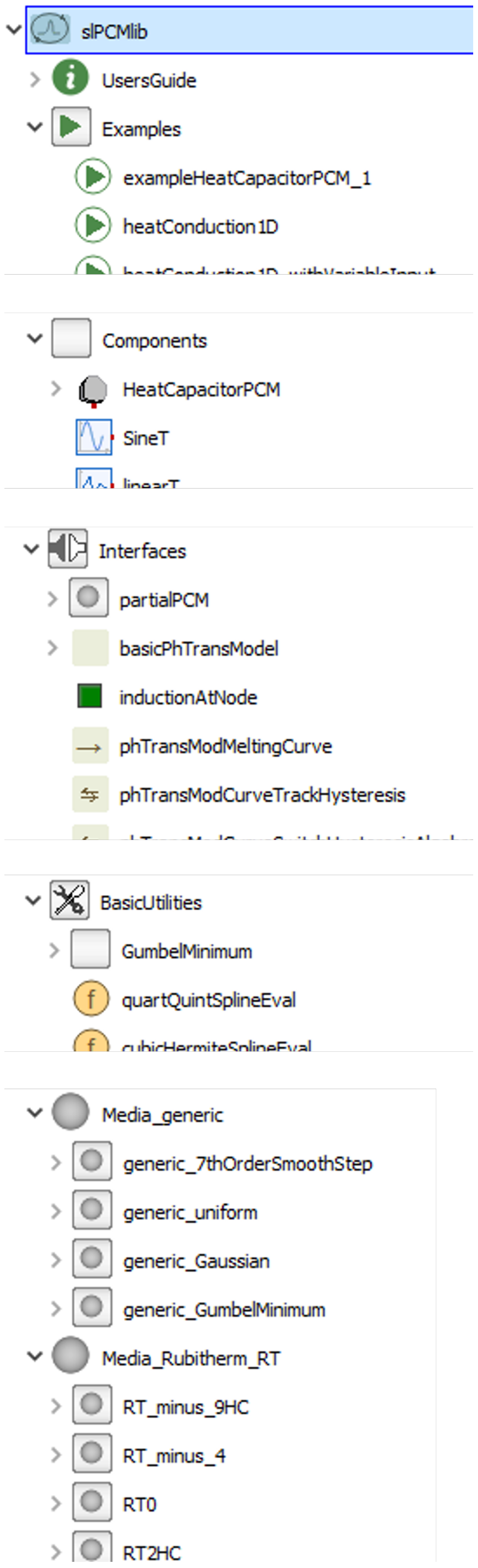
In the package `slPCMlib.Media` (see Figure 5) each PCM extends (replaceable partial) phase transition functions for heating and cooling, see Equation (4), contained in `slPCMlib.interfaces.partialPCM`.

```
replaceable partial function
↪phaseFrac_complMelting "Returns liquid
   mass phase fraction for complete
   melting processes"
  extends Modelica.Icons.Function;
  input Modelica.Units.SI.Temperature T;
  output Modelica.Units.SI.MassFraction xi;
  output Real dxi(unit="1/K");
end phaseFrac_complMelting;
```

The functions of the *specific* PCM are piecewise interpolation splines, see Barz et al. (2020) for details and Figures 3 (bottom) for examples. They are evaluated using e.g. `slPCMlib.BasicUtilities.-quartQuintSplineEval`.

The four *generic* PCM use different ansatz functions. These are the uniform cumulative distribution function (CDF), the Gumbel Minimum (also Extreme value type I) CDF, the Gaussian (also Normal) CDF, and the 7th-order smoothstep function, shown in Figure 6 (top).

The uniform and Gaussian distribution are contained in `Modelica.Math.Distributions`, the Gumbel distribution extends this package and is implemented together with the smoothstep in `slPCMlib.BasicUtilities`.



- slPCMlib
  - UsersGuide
  - Examples
    - exampleHeatCapacitorPCM_1
    - heatConduction1D
    - heatConduction1D_withVariableInput
  - Components
    - HeatCapacitorPCM
    - SineT
    - linearT
  - Interfaces
    - partialPCM
    - basicPhTransModel
    - inductionAtNode
    - phTransModMeltingCurve
    - phTransModCurveTrackHysteresis
  - BasicUtilities
    - GumbelMinimum
    - quartQuintSplineEval
    - cubicHermiteSplineEval
  - Media_generic
    - generic_7thOrderSmoothStep
    - generic_uniform
    - generic_Gaussian
    - generic_GumbelMinimum
  - Media_Rubitherm_RT
    - RT_minus_9HC
    - RT_minus_4
    - RT0
    - RT2HC

Figure 5. Packages in slPCMlib.

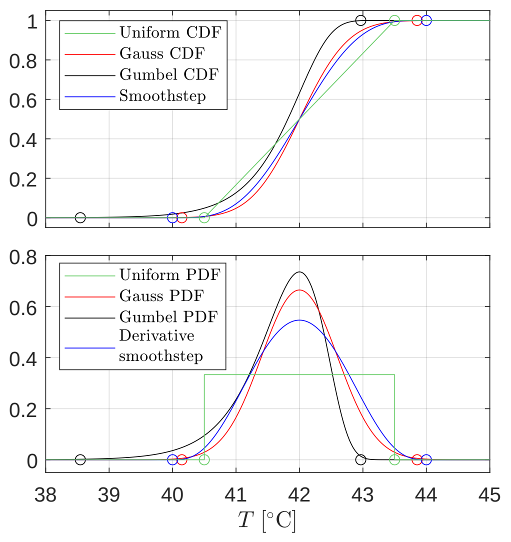


Figure 6. Ansatz functions used for the parametrization of *generic* PCM. Top: Examples for phase transition functions (either for heating or cooling). The following parameters have been used: $T_{\min} = 40.5$, $T_{\max} = 43.5$ (uniform); $\mu = 42$, $\beta = 0.5$ (Gumbel); $\mu = 42$, $\sigma = 0.6$ (Gauss); $T_{\min} = 40$, $T_{\max} = 44$ (Smoothstep). The circle indicates the (approximate in case of Gumbel and Gauss) start and end of the transition range. Bottom: Derivative w.r.t. temperature.

The derivatives of these ansatz functions represent the peak in the effective heat capacity, see also Equation (3) and Figure 6 (bottom). The asymmetric Gumbel distribution can be especially useful for the fitting of heat capacity data for cooling. The symmetric smoothstep function can be parametrized intuitively as it's so-called edge parameters coincide with $T_{\min}$ and $T_{\max}$. Because the CDF of Gumbel and Gauss distributions only asymptotically reach 0 and 1, the limits of the transition range are approximately set for the probabilities $P(T_{\min}) = 0.001$ and $P(T_{\max}) = 0.999$.

4.2 Computation of effective properties

Equations (1) - (3) are contained in the partial model `slPCMlib.Interfaces.basicPhTransModel`. The integral in Equation (10b) is computed by numerical integration using the function `Modelica.Math.Nonlinear.quadratureLobatto` within the `initial equation` section of `basicPhTransModel`. Because the *single phase* heat capacities are modeled as linear functions of temperature, the integrals in the formulas for the corresponding enthalpies in Equation (9b) can be solved analytically. The functions for the heat capacities, baseline and enthalpies are contained in `slPCMlib.BasicUtilities`.

4.3 Phase transition models

The phase transition models described in Section 3 are contained in `slPCMlib.Interfaces`, see Figure 5. They extend the `slPCMlib.Interfaces.basicPhTransModel`. The hysteresis models in Section 3.2 - 3.4 are implemented using when-statements and discrete-time variables. As an example, the *curve track* model in Equation (12) uses the `discrete Boolean heatingOn` to hold the information which submodel is used. Events are triggered and the value of `heatingOn` is updated when T enters the phase transition temperature range, either crossing $T_{\min}$ with rising T, or when crossing $T_{\max}$ with falling T.

```
algorithm
  when (indVar.T <
      PCM.propData.rangeTmelting[2]) then
      ↪heatingOn := false;
  end when;
  when (indVar.T >
      PCM.propData.rangeTsolidification[1])
      ↪then heatingOn := true;
  end when;
```

In the same way, the *curve switch* model in Equation (13) uses the `discrete Integer modelInd` which can take the values +1 (Equation (13a)), -1 (Equation (13b)), and 0 (Equation (13c)).

The algebraic version of the *curve scale* model in Equations (14), (15) use `discrete Real` variables which are initialized and updated using the operator `pre()`. The following code fragment illustrates the updating of the scaling factor for heating:

```
algorithm
  when (indVar.der_T > 0) then
    T0  := pre(indVar.T);
    Xi0 := pre(xi_m);
    (Xi_at_T0,) :=
    ↪PCM.phaseFrac_complMelting(T0);
    scaler := (1.0-Xi0)
    ↪/max((1.0-Xi_at_T0),eps);
    heatingOn := true;
  end when;
```

In addition, several measures were taken to improve the robustness, performance and accuracy of the models:

- The *curve switch* and *curve scale* model are differentiated to avoid nonlinear algebraic equations. Note that for the *curve scale* model there are two versions available, one uses the algebraic formulation (Equation (14)) and the other uses the differential formulation (Equation (16a)).
- The states of the differentiated models are reinitialized at certain points where the exact solution is known. E.g. when the temperature passes the limits of the phase transition temperature range, the phase fraction is reinitialized either with zero or one.
- Additional conditions are considered in when-statements to reduce the number of events. E.g. in the *curve scale* model events are not triggered upon switches between heating and cooling outside the phase transition temperature range.

The following code fragment gives an example for the second point:

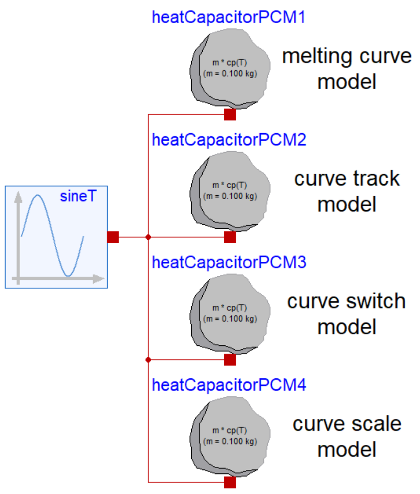


Figure 7. Diagram layer with the modified heat capacitor for the generation of the simulated data in Figure 4.

```
when (indVar.T <= PCM.propData.
↪rangeTsolidification[1]) then
  reinit(xi, 0.0);
elsewhen (indVar.T >= PCM.propData.
↪rangeTmelting[2]) then
  reinit(xi, 1.0);
end when;
```

4.4 Linking transition models and Media

In the following, the selection and use of a transition model and a PCM (medium) is discussed for a simple component model assuming homogenous temperatures inside the PCM. The Modelica Standard Library `heatCapacitor` contained in `Modelica.Thermal.HeatTransfer` has been modified to account for a temperature-dependent specific heat capacity. The modified capacitor `slPCMlib.Components.HeatCapacitorPCM` is shown in Figure 7. A PCM and a phase transition model are selected as:

```
replaceable package PCM=
  slPCMlib.Media.generic_GumbelMinimum;
replaceable slPCMlib.Interfaces.
↪phTransModCurveScaleHysteresis
    phTrModel(PCM=PCM);
```

In the equation section the temperature of the heat port of the capacitor `T` and it's derivative `der_T` are connected with the *inducing* port of the phase transition model `phTrModel.indVar`. The heat flow rate over the port of the capacitor `port.Q_flow` is calculated considering the temperature-dependent heat capacity:

```
equation
  T = port.T;
  der_T = der(port.T);
  phTrModel.indVar.T = T;
  phTrModel.indVar.der_T = der_T;
  phTrModel.cp*m*der(port.T) = port.Q_flow;
```

5 Application examples

5.1 Partial phase transitions

Partial (or interrupted) phase transitions are common in PCM applications. They are characterized by switches between heating and cooling within the phase transition temperature range. Relevant scenarios are interrupted melting and subsequent cooling, or interrupted solidification and subsequent heating.

Figure 8 shows periodic temperature variations and responses in the enthalpy-temperature diagram for two different PCM. Results have been computed with the modified capacitor `HeatCapacitorPCM` choosing the "curve scale" model. The temperature first sweeps the full phase transition range, and subsequently the amplitude is reduced and the temperature oscillates within the transition range. It can be seen that the PCM undergo first a complete melting and solidification cycle forming a major loop. Subsequently, the enthalpy approaches the limit cycle (minor loop with partial transitions) where the absorbed and released heat is reduced. Figure 9 shows the corresponding density changes for both PCM.

5.2 Heat transfer in PCM

This example studies 1D heat conduction in a commercial PCM impregnated gypsum board, namely the SmartBoard®26 (SB26) manufactured by Knauf Gips KG, Germany. The interior plasterboard product with around 30 % mass fraction of microencapsulated paraffinic PCM is available for drywall construction applications in buildings (Kośny, 2015).

A package with the effective thermal properties of SB26 is contained in `Buildings.HeatTransfer.-Data.SolidsPCM`. The phase transition function is modeled as piecewise linear function assuming a nearly isothermal phase change behavior. The transition range of this *ideal* SB26 is [25.99 °C, 26.01 °C].

Another package with effective properties of SB26 is contained in `slPCMlib.Media_Knauf_SmartBoard`. It uses the same single phase properties and phase transition enthalpy. However, two different phase transition functions for heating and cooling are considered (hysteresis). The transition functions were determined from heat capacity data published in Lerche et al. (2010) and are modeled by piecewise interpolation splines. The extended transition range of this *real* SB26 is [20 °C, 30 °C].

The SB26 wall element is modeled using the `Buildings.HeatTransfer.Conduction.-SingleLayer` component of the Modelica Buildings Library. A modified version (`SingleLayerSlPCMlib`) is used with the transition models and media contained in slPCMlib.

As an application example, the test case 600FF of the Building Energy Simulation Test (BESTEST) validation suite (Judkoff and Neymark, 1995) is considered, as implemented in the Buildings library (Wetter et al., 2014). Case 600FF is a light-weight building with a single room

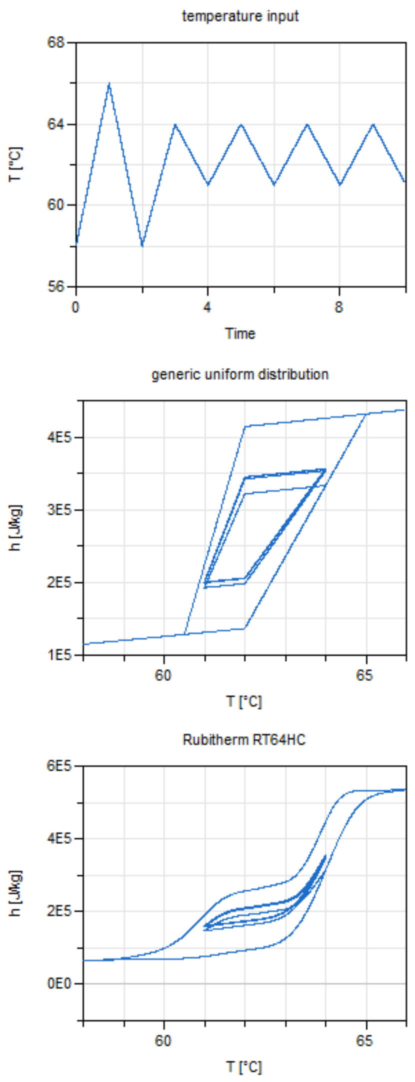


Figure 8. Periodic temperature variations (first subfigure), and the response in the enthalpy-temperature diagram computed with the "curve scale" model: for a generic PCM (second subfigure), and a specific PCM (third subfigure).

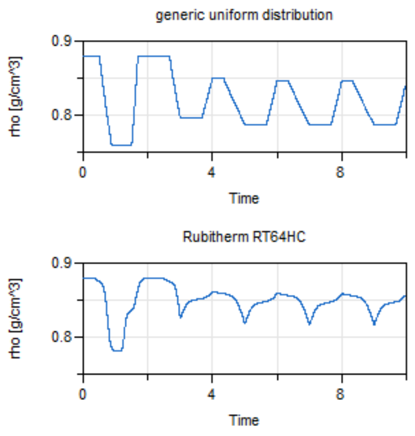


Figure 9. Density changes as response to the temperature variations in Figure 8 (first subfigure). The response is modeled with the "curve scale" model for the two PCM shown in the second and third subfigure in Figure 8.

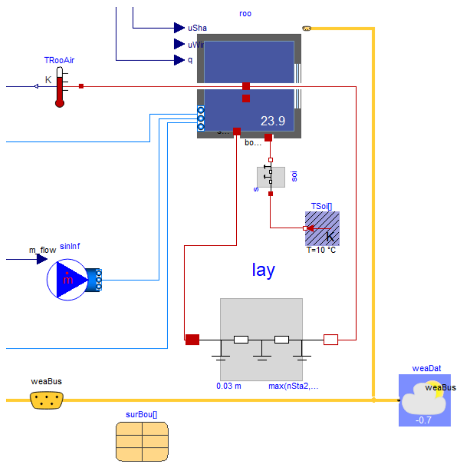


Figure 10. Test case 600FF taken from Modelica Buildings Library (Wetter et al., 2014) with a modified model for a single layer PCM wall element (lay).

of 8 m by 6 m and 2.7 m height. The room temperature is free floating. Double SB26 wall elements of $2 \cdot 15\,\mathrm{mm}$ thickness are attached to the interior wall of the room. We take simplifying assumptions, define the total surface area as $2 \cdot (8\,\mathrm{m} + 6\,\mathrm{m}) \cdot 2.7\,\mathrm{m}$, and connect one heat port of the `SingleLayer` as a heat port of surface connected to the room air and connect the other port to the heat port to air volume, see Figure 10. SB26 temperatures are approximated by two states inside the wall element. This means that two differential heat balance equations need to be solved, each using effective properties ρ, λ and $\tilde{c}$. Because the PCM is microencapsulated, temperature-dependence of ρ which can lead to volume changes, internal material velocities and convection is neglected.

Figure 11 shows results of yearly simulations for three scenarios: without wall element, and with a SB26 wall element considering either an *ideal* or a real phase change behavior. The simulation results indicate differences between all three scenarios. While the PCM wall element leads to an increase in the number of hours with temperatures between 20 and 30 °C, the ideal and real phase change behaviors result in different temperature distributions.

The CPU-times for integration are given in Table 1. As expected, the CPU-times of the test case including SB26 are increased compared with the original case (w/o SB26). The case with the *real* SB26 needs less time compared with the case with ideal SB26. This can be explained by the differences in the phase transition temperature range, i.e. narrow range of the *ideal* SB26 and wide range for the *real* SB26, with corresponding sharp and smooth changes in the apparent heat capacity (and enthalpy) curve, see the third subfigure in Figure 11. Sharp changes in thermal properties make the problem more nonlinear and thus, more difficult to solve. The temperature and climate variability over a year triggers many events when solving the hysteresis models. Interestingly enough, corresponding stops and restarts of DASSL do not heavily affect the over-

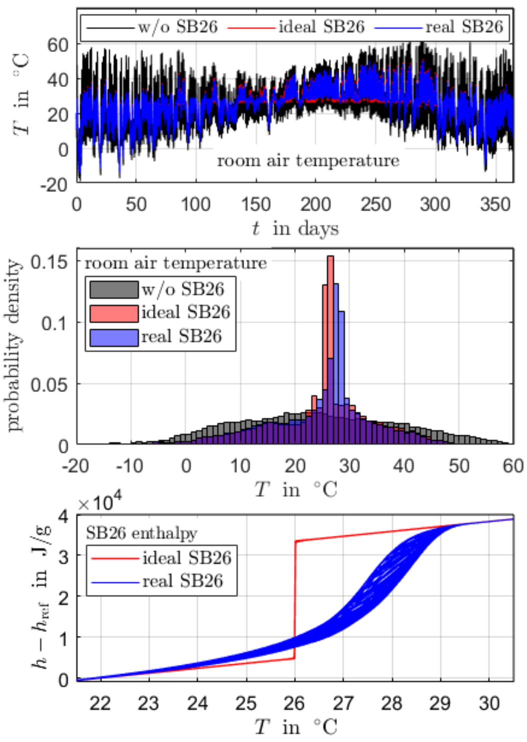


Figure 11. Test case 600FF considering the installation of a commercial PCM impregnated gypsum board SmartBoard®26 (SB26). The first and second subfigure show the temporal evolution and a histogram of the room temperature for three scenarios: without SB26 wall element; with SB26 wall element considering an ideal phase change behavior (as implemented in the Buildings Library); and with SB26 wall element considering a realistic phase change behavior identified from caloric measurements (as implemented in slPCMlib). The third subfigure shows the evolution of the SB26 specific enthalpy in the enthalpy-temperature diagram.

Table 1. CPU-time for integration of the test case 600FF. All computations were carried out on a laptop with Intel(R) Core(TM) i5-8350U CPU @ 1.70 GHz 1.90 GHz and 16 GB RAM. Parallelization of code was not used. The default solver DASSL was selected with a tolerance of 1E-6 and interval length of 60 s.

Scenario	*Model*	*Computation time*	
		absolute	*relative*
w/o SB26	-	74 s	100 %
ideal SB26	melting curve	121 s	164 %
real SB26	melting curve	88 s	119 %
real SB26	curve track	101 s	136 %
real SB26	curve switch	129 s	174 %
real SB26	curve scale	106 s	143 %

all computation times. It should however be noted that the above results are not necessarily generalizable.

6 Discussion and conclusions

slPCMlib allows for a detailed analysis of the thermal performance of PCM enhanced materials and components. Compared with property models considering an isothermal phase change behavior the phase transition models in slPCMlib may be more realistic and accurate. The second application example indicates that this comes at the cost of a quite acceptable increase in model complexity and solution times.

The presented phenomenological hysteresis models are rate-independent. From a practical perspective this is an advantage: They can be easily parametrized by only two curves, e.g. considering heat capacity data from melting and solidification experiments. A comparable approach is used for parametrization of the so-called Tellinen hysteresis model (Tellinen, 1998) which predicts properties of ferromagnetic materials (Ziske and Bödrich, 2012). An interesting extension for rate-dependent modeling could be based on a model-free kinetic analysis of PCM heat capacity data, as recently proposed by Lizana et al. (2021).

In the current version of slPCMlib it is not possible to use the hysteresis models with the enthalpy method for modeling heat transfer in PCM. To do so, an inverse relation for the phase fraction-temperature (and/or enthalpy-temperature) relation would be needed. Examples can be found in literature, however, they are restricted to specific phase transition ansatz functions (and curve shapes): Huang et al. (2022) derives an analytic form of the inverse *curve scale* model parametrized with piecewise linear phase transition functions (the *generic* uniform distribution ansatz function in slPCMlib). Takacs et al. (2008) derives an inverse magnetic hysteresis model considering hyperbolic ansatz functions.

The performance (initialization and CPU-times) of the *curve switch* and *curve scale* models is significantly improved by differentiation with respect to time. However, relaxed solver tolerances may lead to incorrect results. With the current implementation the *melting curve* and *curve track* models, as well as the *curve scale* model (with reasonably strict tolerances) can be all recommended for use. Results from the *curve switch* model might need a critical validation. In any case, a plot in the enthalpy-temperature diagram is recommendable to quickly identify possible errors.

slPCMlib and the examples discussed above can be found at https://github.com/AIT-TES/slPCMlib.

7 Acknowledgement

T.B. acknowledges funding by the Energieforschung (e!MISSION) project fronTector (phase front detector, number 881143). The authors acknowledge the helpful discussion with Atiyah Elsheikh and his suggestions on how to structure the library.

References

Q. Al-Yasiri and M. Szabó. Incorporation of phase change materials into building envelope for thermal comfort and energy saving: A comprehensive analysis. *Journal of Building Engineering*, 36:102122, 2021.

T. Barz. Paraffins as phase change material in a compact plate-fin heat exchanger-Part II: Validation of the "curve scale" hysteresis model for incomplete phase transitions. *Journal of Energy Storage*, 34:102164, 2021.

T. Barz and A. Sommer. Modeling hysteresis in the phase transition of industrial-grade solid/liquid PCM for thermal energy storages. *International Journal of Heat and Mass Transfer*, 127:701–713, 2018.

T. Barz, J. Emhofer, K. Marx, G. Zsembinszki, and L.F. Cabeza. Phenomenological modelling of phase transitions with hysteresis in solid/liquid PCM. *Journal of Building Performance Simulation*, 12(6):770–788, 2019.

T. Barz, J. Krämer, and J. Emhofer. Identification of phase fraction–temperature curves from heat capacity data for numerical modeling of heat transfer in commercial paraffin waxes. *Energies*, 13(19):5149, 2020.

K. Biswas, Y. Shukla, A. Desjarlais, and R. Rawal. Thermal characterization of full-scale PCM products and numerical simulations, including hysteresis, to evaluate energy impacts in an envelope application. *Applied Thermal Engineering*, 138:501–512, 2018.

J. Bony and S. Citherlet. Numerical model and experimental validation of heat storage with phase change materials. *Energy and Buildings*, 39(10):1065–1072, 2007.

A. Buonomano and F. Guarino. The impact of thermophysical properties and hysteresis effects on the energy performance simulation of PCM wallboards: Experimental studies, modelling, and validation. *Renewable and Sustainable Energy Reviews*, 126:109807, 2020.

K. Filonenko, V.B. Ljungdahl, T. Yang, and C. Veje. Modelica implementation of phase change material ventilation unit. In *2020 6th IEEE International Energy Conference (ENERGYCon)*. IEEE, 2020.

F. Goia, G. Chaudhary, and S. Fantucci. Modeling and experimental validation of an algorithm for simulation of hysteresis effects in phase change materials for building components. *Energy and Buildings*, 174:54–67, 2018.

A. Halimov, M. Lauster, and D. Müller. Validation and integration of a latent heat storage model into building envelopes of a high-order building model for Modelica library AixLib. *Energy and Buildings*, 202:109336, 2019.

D. Helmns, D.H. Blum, S.M. Dutton, and P. van Carey. Development and validation of a latent thermal energy storage model using Modelica. *Energies*, 14(1):194, 2021.

W. Hemminger and S. Sarge. The baseline construction and its influence on the measurement of heat with differential scanning calorimeters. *Journal of Thermal Analysis and Calorimetry*, 37(7):1455–1477, 1991.

Y. Hu and P.K. Heiselberg. A new ventilated window with PCM heat exchanger—performance analysis and design optimization. *Energy and Buildings*, 169:185–194, 2018.

H. Huang, J. Lin, Q. Zhao, Z. Xie, and Y. Xiao. Numerical model of pcm heat exchange based on dynamic boundary. *Energy Reports*, 8:579–587, 2022.

Y. Ivshin and T.J. Pence. A constitutive model for hysteretic phase transition behavior. *International Journal of Engineering Science*, 32(4):681–704, 1994.

R. Judkoff and J. Neymark. International Energy Agency building energy simulation test (BESTEST) and diagnostic method. Technical Report NREL/TP-472-6231, National Renewable Energy Lab. (NREL), Golden, CO, United States, Feb 1995. URL https://www.osti.gov/biblio/90674.

J. Kośny. *PCM-Enhanced Building Components: An Application of Phase Change Materials in Building Envelopes and Internal Structures*. Engineering Materials and Processes. Springer, 2015.

F. Kuznik, D. David, K. Johannes, and J.-J. Roux. A review on phase change materials integrated in building walls. *Renewable and Sustainable Energy Reviews*, 15(1):379–391, 2011.

C. Leonhardt and D. Müller. Modelling of residential heating systems using a phase change material storage system. In *Proceedings of the 7th International Modelica Conference*, pages 507–512. Linköping University Electronic Press, 2009.

C. Lerche, H. Siegmund, K. Brands, and L. Kossack. Entwicklung eines innovativen Aktiv- Kühl- und Heizboden unter Ausnutzung des Latentspeicherprinzips. Technical Report Az: 23836 - 24/2, Deutsche Bundesstiftung Umwelt, IBB MODUL AIR KG, Germany, Bad Honnef, Germany, Oct 2010. URL https://www.dbu.de/2468.html.

J. Lizana, A. Perejón, P.E. Sanchez-Jimenez, and L.A. Perez-Maqueda. Advanced parametrisation of phase change materials through kinetic approach. *Journal of Energy Storage*, 44:103441, 2021.

J.E. McDonald. Homogeneous nucleation of supercooled water drops. *Journal of Atmospheric Sciences*, 10(6):416–433, 1953.

B. Michel, P. Glouannec, A. Fuentes, and P. Chauvelon. Experimental and numerical study of insulation walls containing a composite layer of PU-PCM and dedicated to refrigerated vehicle. *Applied Thermal Engineering*, 116:382–391, 2017.

E. Moreles, G. Huelsz, and G. Barrios. Hysteresis effects on the thermal performance of building envelope PCM-walls. *Building Simulation*, 11(3):519–531, 2018.

J. Rose, A. Lahme, N. U. Christensen, P. Heiselberg, M. Hansen, and K. Grau. Numerical method for calculating latent heat storage in constructions containing phase change material. In *Proceedings of Building Simulation 2009: 11th conference of the international building performance simulation association*, pages 400–407, 2009.

Rubitherm Technologies GmbH. PCM RT-line: Technical data sheets. https://www.rubitherm.eu/en/productCategories.html. [Online; accessed 01-December-2019].

J. Takacs, G. Kovacs, and L.K. Varga. Hysteresis reversal. *Physica B: Condensed Matter*, 403(13-16):2293–2297, 2008.

J. Tellinen. A simple scalar model for magnetic hysteresis. *IEEE Transactions on Magnetics*, 34(4):2200–2206, 1998.

V. R. Voller, C. R. Swaminathan, and B. G. Thomas. Fixed grid techniques for phase change problems: A review. *International Journal for Numerical Methods in Engineering*, 30(4): 875–898, 1990.

M. Wetter, W. Zuo, T.S. Nouidui, and X. Pang. Modelica Buildings library. *Journal of Building Performance Simulation*, 7 (4):253–270, 2014.

J. Ziske and T. Bödrich. Magnetic hysteresis models for Modelica. In *Proceedings of the 9th International MODELICA Conference*, pages 151–158. Linköping University Electronic Press, 2012.

[Industrial paper] Digital Twin Applications Using a Cloud Native Modelica Platform

Abhilash Kumar[1] Arunkumar Narasimhan[1] Tharrini Rajendran[1] Stéphane Velut[2]

[1]Modelon Engineering Pvt. Ltd., India, arunkumar.narasimhan@modelon.com
[2]Modelon AB, Sweden, stephane.velut@modelon.com

Abstract

This paper showcases how Modelica technology can be leveraged for real-time applications using a cloud native simulation platform, Modelon Impact™. The platform allows for real-time, two-way communication of data, from the IoT connected plant to a physical model and, from the physical model to a dashboard for plant monitoring and control. The communication relies on open standards and REST-API, which makes it possible to implement digital twins for various applications, such as plant monitoring, predictive maintenance, fault isolation or controls. The paper describes a state estimation workflow where data is transmitted back and forth to the simulation platform via Message Queuing Telemetry Transport (MQTT) and where Node-Red is used for the end-user interface.

Keywords: Digital twin, State estimation, Cloud native Modelica platform, MQTT, Node-RED, REST-API.

1 Introduction

A digital twin is a virtual representation of a real-world physical system or process (a physical twin) that serves for practical purposes, such as system simulation, integration, testing, monitoring, and maintenance. Modelica provides a clear separation between model and analysis definition which has proven successful in various applications over the years, also touching digital twins as illustrated in the following examples.

In motorsports, digital twins of the car, the track, and sometimes even the human driver are used in software-in-the-loop or hardware-in-the-loop configurations. The Modelon Vehicle Dynamics Library® (VDL, 2022) for instance has been used to define digital twins since well over a decade. Driven both by cost and regulations, a successful team in any of the higher leagues such as F1 or NASCAR have virtual representations of each of the cars they put on the racetrack. There are various applications with the common purpose to predict or estimate vehicle behavior beyond what is feasible or even allowed to investigate while the race car is driven on the racetrack.

Figure 1 shows a setup where a digital twin of the car is used to investigate the vehicle behavior on a certain part of the track in more detail. This model is used in offline as well as real-time applications. In the picture, the boundary conditions of the car are given from track and race data and includes for example track curvature, lateral acceleration, and throttle position. Since the model contains detailed representation of the race car's mechanics the workload of critical components such as tires, springs, and dampers can be estimated.

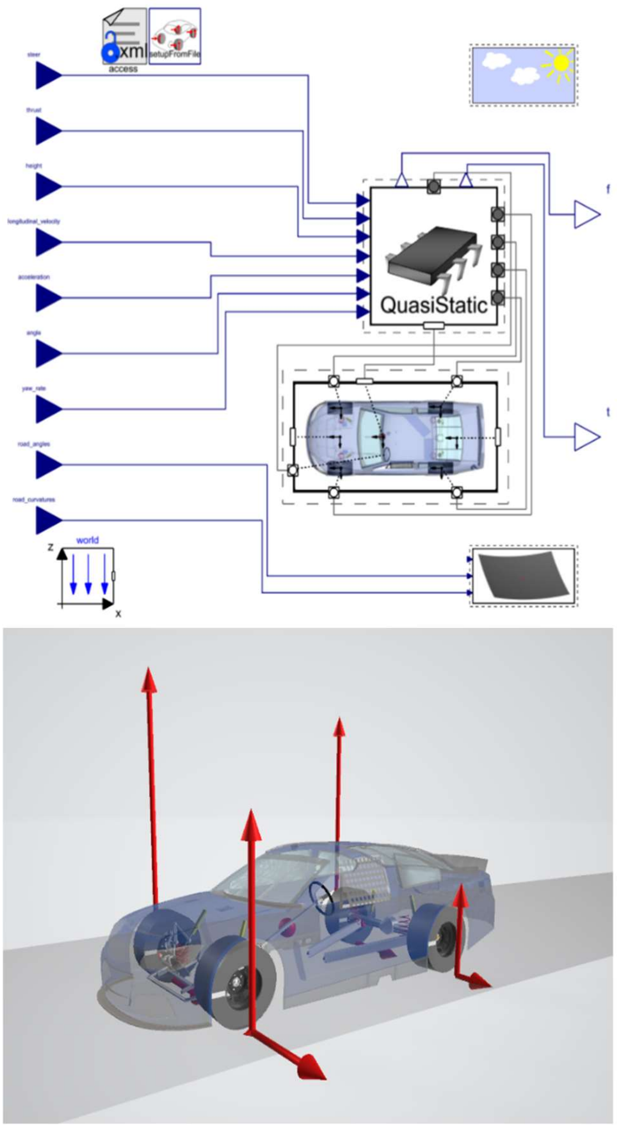


Figure 1. Digital twin of race car from the NASCAR series. Diagram view with race car and boundary conditions (top) and 3D visualization (bottom). The arrows show the estimated individual tire forces.

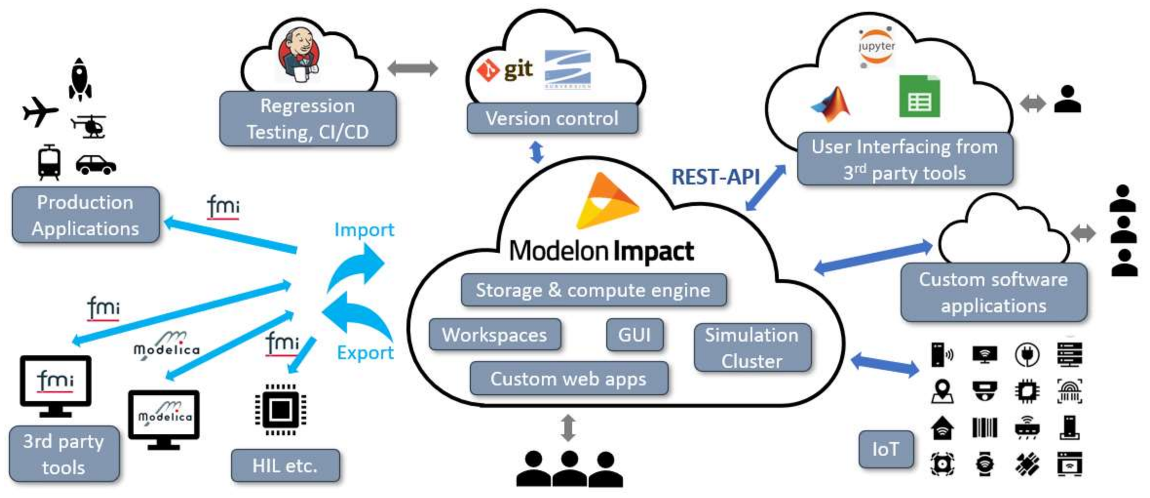


Figure 2. Modelon Impact is prepared to work in an eco-system with well-defined communication through public APIs.

In other applications, such as predictive maintenance or fault detection, the physical model needs to be used in combination with an algorithm to extract valuable information from the process in operation. This can be achieved by exporting the physical model to a scripting environment such as Matlab or Python using the Functional Mock-up Interface (FMI) standard. (Ruggaber and Brembeck, 2021; Gonzalez, et al., 2017; Andrén, et al., 2015) demonstrate how various variants of Kalman filters can be implemented for state and parameter estimation, also exploiting the directional derivatives defined by the FMI 2.0 standard. The combined usage of Modelica, FMI and the scripting environment has been proven to be successful for optimal start-up of power plants in offline mode (Dietl et al., 2014). For rapid testing and deployment of the state estimators without any need for scripting environment, the estimation algorithm can also be embedded according to the FMI standard as a Functional Mockup Unit (FMU), as shown in (Brembeck et al., 2011), (Bonvini et al, 2014), (Laughman and Bortoff, 2020). This however requires manual adaptation work for every considered plant model.

Data exchange is also an essential component that sets requirements on the digital twin implementation in terms of connectivity and openness. This can be achieved by integrating a plant model as FMU into a connected data management or control system as it was done in (ENGIE, 2022). Modelon Impact™ was used there to derive a digital twin of a solar photovoltaic power plant in Chile and to train fault detection algorithms. The model was run online, and its predictive nature permitted to detect and isolate component failures. In (Dietl and Link, 2018) the communication between the control system and the simulation platform relied on OPC-UA. The authors implemented and deployed a Moving Horizon state model predictive controller based on Modelon's optimization toolchain.

The mentioned examples have in common that they showcase the industrial value and the feasibility of digital twin type analyses based on Modelica and FMI technologies. They also illustrate the need for a framework that allow for a more systematic way of implementing and deploying models for real-time applications. The objective is to achieve a modular and flexible implementation to keep the model and the analysis separated and thereby facilitate code reuse for multiple applications.

This paper showcases how this can be achieved with Modelon Impact, a cloud-native Modelica-environment with public APIs. The paper is structured to first outline the relevant properties of Modelon Impact in Section 2, prediction, and correction in Section 3, followed by a set of select applications in Sections 4-5.

2 Enabling cloud infrastructure

Modelon Impact (Modelon Impact, 2022) is a cloud-native systems modelling and simulation environment that enables connectivity through public APIs. Modelon Impact generates and runs FMUs on the cloud. It is also possible to upload third party Modelica packages and use them either as dependencies or editable models. Modelon supports connectivity to version control software like Git and SVN. Modelon Impact also features installation on private clouds to ensure data security. Figure 2 shows an overview of the connectivity options that are offered. In this paper we will focus on the APIs that allow for 3rd party tools to communicate with the compute engine in Modelon Impact.

Modelon Impact communicates through a Representational State Transfer (REST) API, that enables remote controlling from other applications such as Microsoft Excel, Jupyter (Kluyver, et al., 2016), and custom web apps. The Modelon Help Center (Modelon Help Center, 2022) contains detailed information of the available REST API calls.

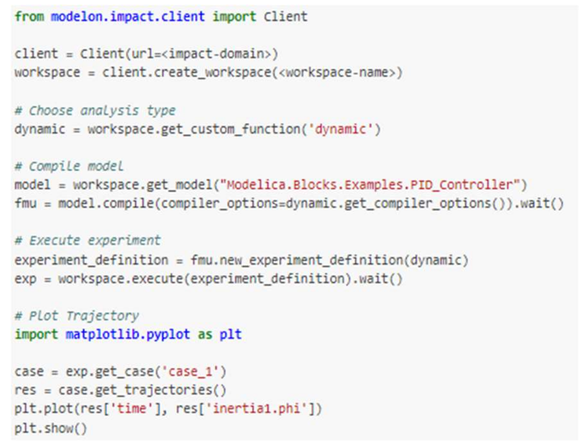
Modelon Impact has client libraries in Python and JavaScript wrapping around the low-level web-interface (REST API) which makes it easy to programmatically connect and interact with a Modelon Impact server.

The client libraries help with:

- Defining and executing simulations on the Impact server.
- Compiling models on the server and downloading them as FMUs.
- Fetching results and do post-processing.
- Authenticate users against Modelon Impact.
- Creating and automating custom workflows in your favorite programming language

The client libraries enable the execution of workflows orchestrated on a client and executed on a Modelon Impact server, which may be running remotely. With sufficient login credentials and an API Key, Modelica models may be uploaded, compiled, and executed on a server. The results can be either processed on the server with a custom function or downloaded to the client for further analysis.

An analysis could be set up and executed and relevant trajectories plotted using the Python client library in a few lines of code as shown in Figure 3. Further information about the usage of the API is given for each application below.



```python
from modelon.impact.client import Client

client = Client(url=<impact-domain>)
workspace = client.create_workspace(<workspace-name>)

# Choose analysis type
dynamic = workspace.get_custom_function('dynamic')

# Compile model
model = workspace.get_model("Modelica.Blocks.Examples.PID_Controller")
fmu = model.compile(compiler_options=dynamic.get_compiler_options()).wait()

# Execute experiment
experiment_definition = fmu.new_experiment_definition(dynamic)
exp = workspace.execute(experiment_definition).wait()

# Plot Trajectory
import matplotlib.pyplot as plt

case = exp.get_case('case_1')
res = case.get_trajectories()
plt.plot(res['time'], res['inertia1.phi'])
plt.show()
```

Figure 3. Sample code to remotely operate Modelon Impact using a Python client library.

3 State Estimator implementation

As mentioned in the introduction, state estimation is an important component in digital twin applications. It can be used to filter noisy measurements, estimate key variables that cannot be reliably measured or estimate unknown parameters in the plant model. All state estimators, from standard to advanced Kalman Filters or Moving Horizon Estimators (MHE), share a similar structure as shown in Figure 4. They are driven by the plant inputs and measurements and generate estimates of key performance indicators. The physical plant model is often extended by a disturbance model to cope with modelling errors or unknown parameters.

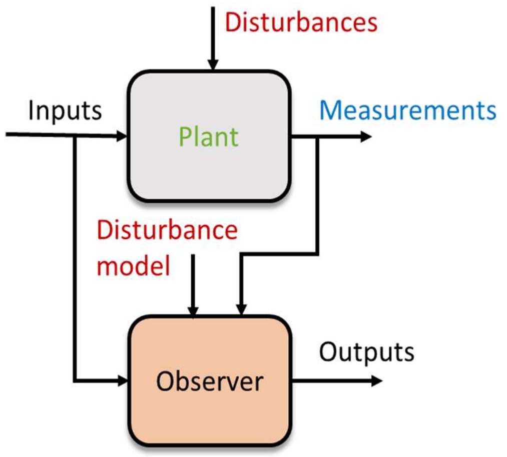


Figure 4. Schematics of an observer model that generates estimates of key variables from plant inputs, measurements, a plant model and eventually a disturbance model.

Such estimation workflows can be conveniently implemented and deployed using Modelon Impact and standard technologies and communication protocols. A dashboard for plant monitoring is built on Node-RED (NR) (Node-RED, 2022) where the data flow between the nodes of plant, digital twin/observer, and the NR dashboard visualized as shown in Figure 5.

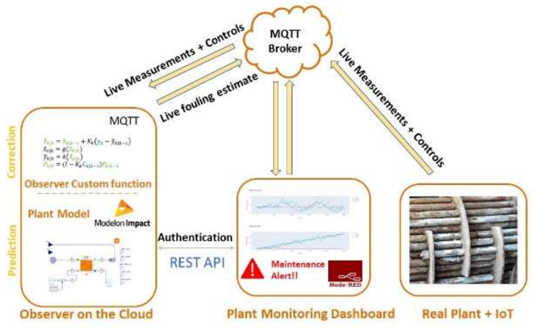


Figure 5. Modelon Impact and Node-RED based data flow for plant monitoring.

A combination of the Modelon Impact JavaScript client library and MQTT (MQTT Protocol, 2022), a publish/subscribe messaging protocol in the backend facilitates the two-way data exchange, where the plant measurement data are published on a specific topic to a central MQTT message broker and Modelon Impact acts as subscriber and listens to this topic. The measurement values received are fed to a custom function implementing an Extended Kalman filter as state estimator. The Kalman filter consists of two parts: a combined plant and disturbance model in Modelica for the prediction step and a Python script that implements the correction step. The plant estimates are then further broadcasted to the plants NR dashboard though the

MQTT broker. The authenticated user would call Modelon Impact to simulate the digital twin using the JavaScript APIs to Modelon Impact. Key performance indicators along with state variables and estimates would be published in the plant dashboard as shown in Figure 6 for fault prevention and predictive maintenance. NR dashboard compares model predicted state variables and corrected estimates from Extended Kalman Filter (EKF) along with live measurements from the real plant.

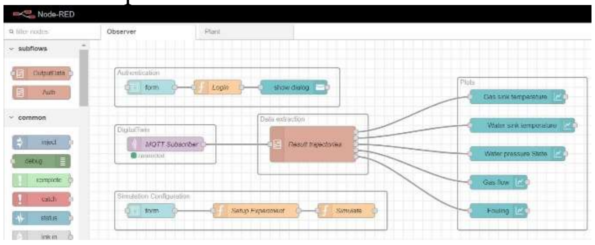


Figure 6. NR flow for plant dashboard.

The workflow is independent of the estimator type. If needed, one could implement a customer function for each estimation approach: Kalman filter, EKF, Unscented Kalman Filter (UKF) or MHE. The whole digital twin implementation is modular and flexible: plant model, algorithm for estimation, integration algorithms, monitoring dashboard are all separate, can be maintained and developed independently.

4 Digital Twin application 1: Heat exchanger fouling estimator

In this section, the state estimator workflow presented earlier is tested on a specific example: fouling estimation in a heat exchanger. In real time applications, fouling is always a major concern with the use of heat exchangers, which gradually degrades system performance and component life, while increasing the operational costs over time. But fouling cannot be measured and often cannot be identified early without leveraging live plant data. Fouling will here be estimated using a generic custom function implementing an Extended Kalman Filter, a heat exchanger model from Modelon Thermal Power Library and the framework described in the previous section. Plant data is here emulated using another heat exchanger model that runs on Modelon Impact, exchanging data using MQTT protocol.

Figure 7 shows a heat exchanger (HX) model from Modelon Thermal Power Library with open boundary conditions. The application in mind here is to estimate fouling on the gas side based on five noisy measurements, encircled in blue in Figure 7: all inlet and outlet temperatures as well as the liquid mass flow rate. It is assumed that the gas flow rate is not measurable, and it will also be estimated. The heat exchanger model is discretized in the flow direction according to a finite volume implementation. Each section has then three dynamic states, for liquid pressure, liquid temperature, and wall temperature. The plant model has been extended by a disturbance model to describe the unknown gas flowrate and the fouling factor (encircled in red in Figure 7). The disturbances are implemented in Modelica and assumed to be constant in time although they will be varied in the experiment.

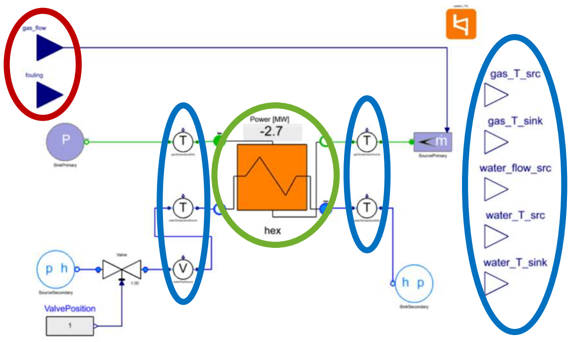


Figure 7. Physical plant model using the fouling estimator.

The measurements from the emulated plant and predictions of those measurements from the observer model are compared to validate the estimations. In Figure 8 showing the estimates, the fouling estimate in grey and the gas flow rate estimates are able to follow the true value. The offsets are due to the fast dynamics of the emulated fouling and gas flow changes, but they could be reduced with a more detailed disturbance model.

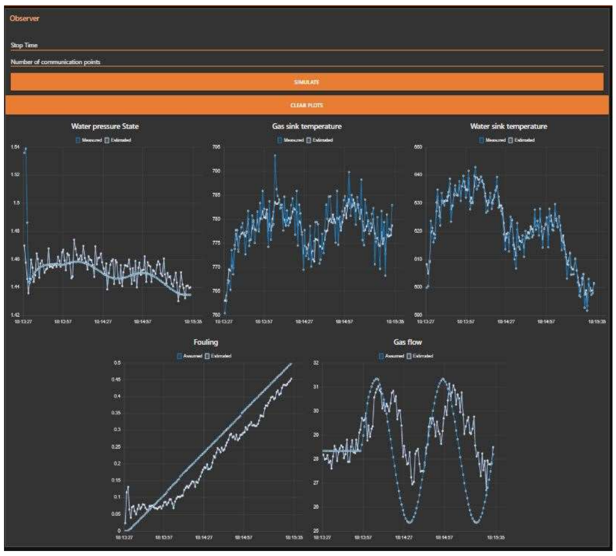


Figure 8. NR plant monitoring dashboard displaying measurements as well as estimates for fouling and gas flowrate.

5 Digital Twin application 2: Windmill Fleet

The previous section illustrates the application of Modelon Impact for performance monitoring of a single plant. There are a variety of cases where it is necessary to monitor a fleet of assets. For example, the growing number of wind farms demand highly reliable integrated systems to curtail Operation & Maintenance cost. The workflow described in section 4 can be extended to implement Digital Twins for online monitoring of a fleet of assets that are geographically spread as shown in Figure 9.

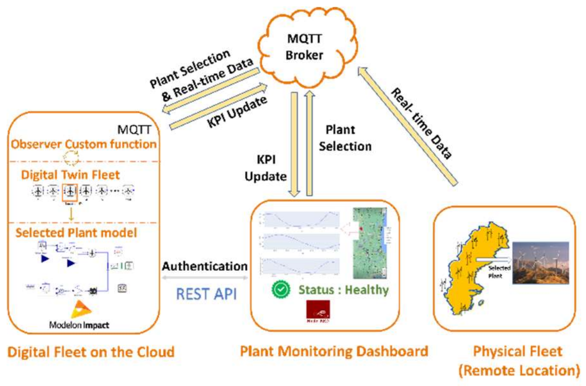


Figure 9. The overall digital twin workflow in the case of an asset fleet, here a Windmill Fleet.

The Digital Twin Fleet was implemented similarly as the single asset example, using three modules (i) Digital Twin Fleet based on a set of Modelica models built in Modelon Impact (ii) Real Time wind and plant data at various locations (iii) Interactive plant monitoring dashboard.

The plant monitoring dashboard needs to include an interactive map component that allows operators to select the windmill of interest in the windmill fleet. A proof-of-concept has been implemented using JavaScript. The map component in the dashboard is interactive and allows the user to select the location of the windmill of interest. It is a reaction based interactive map component containing map data through Google API. To the map, several different interface elements like overlays for point of interest, zoom in/out, highlight selection is added for best user interaction experience. Real time wind data for various windfarms in different locations were embedded into the map element through JavaScript. Real time data for the wind at any selected location was fetched from (Trafikeverket's open API, 2022). Additionally, the monitoring dashboard has been implemented to display the data of the windmill selected from the fleet. In this case study, no state estimation problem was solved. The goal of the demonstrator was instead to show the ability of the technology to deal with a fleet of digital twins with respect to plant selection, data visualization and bi-directional data exchange between the model and the plant data.

6 Conclusion

Digital twin applications based on Modelica models and FMI standard are not new. Different solutions have been suggested in literature and some tested in industrial applications. This clearly shows the feasibility and the potential of the approach. With Modelon Impact on the cloud and its public APIs, the path from systems modeling and simulation to digital twin in operation is significantly shortened. It also allows for a modular and flexible digital twin implementation where models and algorithms can be kept separate and be re-used for different applications. The NR dashboards powered by Modelon Impact Digital Twin are easy to setup and can present complex scenarios in easily understandable manner. They can also meet the connectivity requirements of simulation platform in digital twin applications by enabling bi-directional data exchange using standard communication protocols.

Acknowledgements

Special thanks to Johan Andreasson, Peter Sundström and John Griffin for their input on motorsports digital twin applications, and to Emil Fredriksson as well as Ola Flisback for the implementation of the windmill fleet example.

References

M. T. Andrén, and C. Wedding, (2015): Development of a Solution for Start-up Optimization of a Thermal Power Plant, M.Sc. thesis, Department of Automatic Control, Lund University, Sweden.

M. Bonvini, M. Wetter, and M. Sohn,(2014), An FMI-based Framework for State and Parameter Estimation. 10.3384/ecp14096647. *In Proceedings of the 10th International Modelica Conference Lund March 10-12.*

J. Brembeck, M. Ottera, and D. Zimmer,(2011): Nonlinear Observers based on the Functional Mockup Interface with Application to Electric Vehicles, *In Proceedings of the 10th International Modelica Conference Dresden March 20-22,2011.*

K. Dietl and K. Link, (2018): Startup optimization of Combined Cycle Power Plants: Controller development and real plant test results. *5th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 599-604, doi: 10.1109/CoDIT.2018.8394850.

K. Dietl, S.G. Yances,A. Anna, J. Akesson, K. Link, and S. Velut, (2014): Industrial application of optimization with Modelica and Optimica using intelligent Python scripting. *In Proceedings of the 10th International Modelica Conference*, *Lund,March 10-12.*

ENGIE: URL https://modelon.com/support/engie-solar-power-plant-predicitive-maintenance-digital-twin/,2022.

M.Gonzalez, O. Salgado, J. Croes, B. Pluymers, and W. Desmet, (2017): Model-based virtual sensors by means of Modelica and FMI. *In Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017,* 337-344. 10.3384/ecp17132337.

T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and Jupyter development team, (2016): Jupyter Notebooks – a publishing format for reproducible computational workflows. *Loizides, Fernando and Scmidt, Birgit (eds.) In Positioning and Power in Academic*

Publishing: Players, Agents and Agendas. IOS Press. pp. 87-90. (doi:10.3233/978-1-61499-649-1-87)

C. Laughman, and S. Bortoff, (2020): Nonlinear State Estimation with FMI: Tutorial and Applications. *In Proceedings of the American Modelica Conference 2020 Boulder, Co, USA March 23-25* 186-195. 10.3384/ecp20169186

Modelon Impact: https://modelon.com/modelon-impact/, 2022.

Modelon Help Center: https://help.modelon.com, for API https://help.modelon.com/latest/guides/apidocumentation.html, 2022.

Modelon Vehicle Dynamics Library (VDL): https://modelon.com/library/vehicle-dynamics-library/, 2022.

MQTT Protocol, http://mqtt.org/, 2022

Node-RED, https://nodered.org/, 2022.

J. Ruggaber and J. Brembeck, (2021): A Novel Kalman Filter Design and Analysis Method Considering Observability and Dominance Properties of Measurands Applied to Vehicle State Estimation. *Sensors* 21, no. 14: 4750. https://doi.org/10.3390/s21144750

Trafikeverket's open API for traffic information with weather data: https://api.trafikinfo.trafikverket.se/, 2022.

Advanced open source data formats for geometrically and physically coupled systems[*]

Andreas Naumann[1] Jens Saak[1,3] Stefan Sauerzapf[2] Julia Vettermann[1] Michael Beitelschmidt[2] Roland Herzog[1]

[1]Technische Universität Chemnitz, Faculty of Mathematics, 09107 Chemnitz, Germany (andreas.naumann,julia.vettermann,roland.herzog@mathematik.tu-chemnitz.de).
[2]TU Dresden, Faculty of Mechanical Science and Engineering, Institute of Solid Mechanics, Marschnerstraße 30, 01307 Dresden, Germany (stefan.sauerzapf,michael.beitelschmidt@tu-dresden.de).
[3]Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany (saak@mpi-magdeburg.mpg.de).

Abstract

Numerical calculations based on models are nowadays standard tools in all engineering disciplines. The tools, which faciliate the modeling, generally include all tasks in an engineering workflow. These tasks range from simple model descriptions to advanced visualization of results.

While the incorporation of all tasks in one tool fits neatly into a single-person scheme, it makes teamwork with shared tasks very hard. In particular, every member of the team has to use the exact same software, and every sub-task has to be available in the tool. This requirement, in turn, makes a joint development of advanced methods unnecessarily complicated. Especially the numerical analysis of problem tailored methods requires a detailed knowledge of all model ingredients.

The basis for joint workflows and teamwork are interfaces and common data formats. In this publication, we present a data format for geometrically and physically coupled systems. The data formats structure bases on the standardized format *JSON*, whereas the content is derived from a mathematical model. Finally, for presentational purposes, we present an instance of a simplified model.

Keywords: model language, software interfaces, partial differential equations

1 Introduction

Over the recent decades, numerical simulations have proven to be an indispensable tool for understanding scientific and industrial processes.

Part of the success story of numerical methods are the thorough numerical analysis and the increasing trust of the users. The users of numerical software are a very heterogeneous group. On the one end of the spectrum we find users who only concentrate on the results and runtime of the method. The quality of a solution is then assessed by comparison to experience, expectation and measurements. Deviations from the expectations are usually attributed to the model. On the opposite end we have those who have a deeper understanding of the method, and select numerical methods based on the properties of the model. This subgroup also considers numerical errors in addition to the model errors. In contrast to both, the numerical analysts consider the method itself as a research subject. They know the details and properties of the methods, and relate the efficiency of the method to the properties of the model. To them, the model error is not of interest, but serves as tolerance.

A development team for a new technical device, together with its digital twin, has to incorporate all the above user types. We aim to provide a workflow that suites all their needs and allows for maximum flexibility in the distribution of tasks among them, i. e. benefits teamwork in the best possible way. As an example we investigate the development of a thermo-elastic model of a machine.

The workflow was applied to a thermal model in Sauerzapf et al., 2020 and Vettermann et al., 2021 and a thermo-mechanical model in Naumann, Herzog, 2021 using the proposed data formats. Here, we concentrate on the interface and the key ideas. In particular, we consider the data format as an approach to extend Modelica® with functionality for coupled partial differential equations (PDEs).

The development of new machines requires, often prior to construction, a deep understanding of the resulting machine's behavior under load and other influences from the environment. Particularly the interaction between the machine components, and their influence on quantities of interests (QOI), require a thorough analysis. Nowadays, this analysis is generally carried out numerically with the help of a digital twin of the machine.

Computer aided design (CAD) representations of the machine form the basis for the numerical analysis. These CAD models represent the geometry of the machine. Depending on the specific question, the geometry can be cho-

[*]This work was supported by the DFG grants 174223256 – TRR 96 and 460135501 - NFDI 29/1 "MaRDI – Mathematische Forschungsdateninitiative".

sen more or less detailed.

The physical model extends the geometric model with the physical effects of interest. A physical model contains a variety of sub-models, where each sub-model describes a particular effect up to the required accuracy. For example, (thermal) loads inside a machine describe the heat input per volume, or the description of the interaction with the environment are sub-models. These examples belong to one physics, but different geometric parts. Nowadays multi physics models, which link several physics, are standard. Our use case, the thermo-elastic model of a machine tool, comprises the evolution of the heat distribution and the corresponding mechanical expansion. Therefore, a data format for the description of the physical system must be able to distinguish sub-models of the different physics.

We describe the overall physics using coupled systems of PDEs. The physical model, thus, consists of equations of different types. Among these types, there are systems of PDEs to describe the overall physics, like the evolution of the temperature field, algebraic equations for simple sources and ordinary differential equations (ODEs) to describe more complex sources. For the heat equation, these sources correspond to thermal loads inside and at the boundary. In the mechanical model, the sources correspond to forces acting inside a volume or at a boundary. In the thermo-elastic model, each physical type can exist.

To the authors knowledge, there are not many data formats to describe a system of geometrically and physically coupled PDEs. The authors are aware of

- extensions to support PDEs in Modelica®, see Li, Zhang, Zheng, 2008; Saldamli et al., 2005. So far, these extension to Modelica® only support very simple geometries, and, as a consequence, very simple (spatial) discretization methods.
- toolboxes in symbolic software Portela, Charafi, 2002. These toolboxes are tightly bound to the software, which we want to avoid.
- two similar approaches to describe finite element (FE) analysis using XML Michopoulos et al., 2001; Pinheiro, Moita, 2004. Their goal is to exchange a full set of FE analysis data, including results and solver data. Our aim is to separate the solver and tool specific configurations from the problem description. Sadly, the website `femml.sourceforge.net` had the last update in May 2002.
- a masters thesis Hvalstad-Nilsen, 2019, in which a YAML-based file format was developed. Similar to Michopoulos et al., 2001, the student describes a full FE analysis.

Due to this lack of tool-agnostic data formats, or libraries, for PDEs we developed a new format.

The numerical solution of the PDEs requires discretization in space and time. We follow the method-of-lines approach, which first performs a spatial-semidiscretization, leading to a system of ODEs.

For simplicity, we will present the structures and explain their relationships using the heat equation, and note the extensions for the mechanical behavior, where appropriate.

The thermal loads induce the majority of the thermal behaviour of the machine. Therefore, they must be considered properly in the thermal PDE model. At the same time, these loads determine the overall time scales. Prominent examples are rapidly varying loads and the coupling heat fluxes for relatively moving machine components. In particular the treatment of the temperature dependency of the coupling fluxes are of special interest when discretizing the PDE.

We propose the separation into a problem description and algorithm description. That way, only the algorithm description requires tool specific entries whereas the problem part concentrates on the common entries. Thus, we propose to use external software packages for the finite element (FE) discretization. Proprietary software solutions often provide a seamless workflow starting from the CAD model and proceeding all the way to the solution of the discretized ODEs. Consequently, the mathematical details and methods are completely hidden in the software.

This abstraction has the advantage that the construction of large and complicated models becomes feasible. At the same time, the specific numerical methods are harder, and more frequently not at all, extendable or even known. In contrast, the open-source packages are usually more specialized in the single tasks. For example, the mesh generation and the assembly (or, nowadays, the application) of the FE operators are usually implemented in different libraries. Thus, the implementation of a model requires a profound understanding of the underlying numerical methods. From the scientific point of view, access to the numerical methods and their implementation is key for further improvements and for understanding the efficiency of the simulation. At the same time, the open-source software in many cases has free licenses, thus the costs for the development can be reduced, when the license model allows proprietary downstream use, and modelers have received the appropriate training. See Sauerzapf et al., 2020 for an example where we show how to use open-source tools for discretization and model order reduction from within the proprietary FE package ANSYS®.

To facilitate the cooperation between scientific disciplines we propose an interface between

(i) industry standard modeling tools, like ANSYS® or COMSOL®,

(ii) open-source FE software packages, like DUNE or FEniCS,

(iii) ODE software packages.

In addition to the aforementioned tasks, the same interface connects to model order reduction (MOR), sensor placement Naumann, Herzog, 2021 or even parameter estimation. While the incorporation of the MOR toolbox relies mainly on the structure of the mathematical model, higher-level tasks such as sensor placement and parameter estimation require further operations. We concentrate on the extension of the FE discretization in space with the geometric coupling approaches. The interface to MOR and ODE packages will be discussed together.

For these purposes, the pointwise evaluation, as provided by a functional mockup unit (FMU), is not sufficient. In particular the MOR methods require the separation into inputs and corresponding coefficients. Thus, the FMU export and import between ANSYS® and Modelica® does not meet all our requirements.

We propose a tool-agnostic data format, which provides all required information about the model in an object-oriented layout. As a result, our proposed interface contains only the relationships between physical entities, but does not denote variables in the sense of a particular programming language, or environment.

From an abstract point of view, our format has similar goals as the SSP-standard Association, n.d. Both aim to separate common properties and relationships from tool specific ones. While the SSP-standard aims to bundle (technical) systems, we consider a mathematical model. In addition we made different technical decisions, like the basic file format or notation of URIs.

The structures of the (mathematical) PDE model and the ODE model are the basis for the interface. We briefly explain these models in Section 2. The two interfaces, one between the industry modeling tools and the open-source software, and one between the generated ODE and the simulation software packages, will be described in Section 3. Finally, we demonstrate the interface format with a simple model in Section 4 and give an outlook in Section 5.

2 Mathematical background

This section provides the mathematical foundations of the PDE and the ODE model at hand. Because the second stems from the discretization of the first, they have some parts in common.

The PDE model will use the heat equation for purpose of presentation. As it is a scalar equation for a scalar field, we can concentrate on the general structure. Other physical PDE, like Eulers equation, Navier-Stokes equations, equations of linear elasticity follow the same scheme. They comprise differential operators in space and time, which operate on a solution field. The tensor shape of the solution field is determined by the physical quantity. The boundary conditions close the system. Boundary conditions for vector valued problems can also restrict the vector to a sub-space and in general depend on the local basis of the vector field. For example, in linear elasticity, one fixes one coordinate direction, but leaves the others free to prevent penetration of a wall, but allow frictionless sliding along it.

In analogy, we present the ODE in Section 2.2 as first order ODE. The order is determined by the order (of the time derivative) of the PDE. Although one can transform every higher order ODE into a system of first-order ODEs, in a data format it is favorable to keep the second-order structure.

2.1 The PDE model

Real world applications often feature complicated geometries, where the FE discretization is the standard procedure. For the mathematical theory of the discretization we refer the reader to Grossmann, Roos, 1994; Zienkiewicz, Zhu, 1987.

We consider the conservation of heat in a system of solids with heat exchange between them. In the ith solid with domain $\Omega^{(i)}$ we describe the evolution of the temperature $T^{(i)}$ using the heat equation, i. e.

$$\partial_t\left(\rho^{(i)}C_p^{(i)}T^{(i)}\right) - \nabla\cdot\left(\lambda^{(i)}\nabla T^{(i)}\right) = Q^{(i)} \qquad \text{in } \Omega^{(i)} \tag{2.1a}$$

$$\partial_n\lambda^{(i)}T^{(i)} = g_{N,j}^{(i)} \qquad \text{on } \Gamma_{N,j}^{(i)} \tag{2.1b}$$

$$T^{(i)} = g_{D,j}^{(i)} \qquad \text{on } \Gamma_{D,j}^{(i)}. \tag{2.1c}$$

Inside the domain $\Omega^{(i)}$, the PDE (2.1a) holds, which depends on the material parameters density $\rho^{(i)}$, capacity at constant pressure $C_p^{(i)}$ and heat conduction $\lambda^{(i)}$, as well as on the volumetric thermal sources $Q^{(i)}$. Since the machine parts are often assembly groups consisting of several bodies, in addition, we assume to have a partition of the domain $\Omega^{(i)}$ into subdomains $\Omega_j^{(i)}$. Each domain $\Omega^{(i)}$, then, is a part and the subdomains $\Omega_j^{(i)}$ are the single bodies inside that part.

When the assembly groups move relative to each other, the domain $\Omega^{(i)}$ is time-dependent. Thus, the domain $\Omega^{(i)}$ at time t can be defined as

$$\Omega^{(i)}(t) := \left\{x^{(i)} \,\middle|\, x^{(i)}(t) = g_M^{(i)}(t,\hat{x}^{(i)})\ \forall \hat{x}^{(i)} \in \widetilde{\Omega}^{(i)}\right\}, \tag{2.2}$$

where $\tilde{\Omega}^{(i)}$ is a fixed reference domain and $g_M^{(i)}$ represents the movement. For simplicity, we restrict the movements to expressions of the form $g_M^{(i)}(t,\hat{x}^{(i)}) = O^{(i)}(t)\hat{x}^{(i)} + b^{(i)}(t)$, where $O^{(i)}$ are orthogonal matrices. In case of time-dependent non-orthogonal coordinate transformations, the space-discrete system gets additional time-dependent terms.

Each boundary $\Gamma^{(i)}$ is also separated into sub-boundaries $\Gamma_{N,j}^{(i)}$ and $\Gamma_{D,j}^{(i)}$. At each sub-boundary we apply either condition (2.1b), or (2.1c), whichever corresponds to the sub-boundary type. Please note that the subset of the boundaries is independent of the selection of the bodies, although they might intersect. The subscripts N and D represent Neumann and Dirichlet conditions, respectively.

In general, the material parameters and right-hand side of the boundary conditions can be arbitrary expressions. In particular, all expressions are allowed to depend on time, space and temperature, or a mixture thereof. The dependencies of the material parameters, the sources and the boundary conditions on the temperature determine the classification of the PDE.

Despite the simplicity, temperature-independent material parameters and at most linearly temperature-dependent sources and boundary conditions, often describe the reality to sufficient accuracy. This special case renders the PDE model linear, which carries over to the ODE, after spatial semi-discretization. The solution of PDEs with constant coefficients is well understood and a wide range of numerical methods exists. The discretization requires the dependency information for all expressions.

We equip the system of heat equations with QOIs

$$y = \mathscr{C}(T)\,, \tag{2.3}$$

where the operator $\mathscr{C}$ maps all temperature fields to one vector. The following output operators are of special interest:

(i) A linear, block-structured operator $\mathscr{C}$, such that the QOI y_i depends on the temperature field $T^{(i)}$ in the i-th body. This case includes the temperature in a point, the average temperature in the whole domain, in parts of the domain, or averages on surfaces. This structure was used in Vettermann et al., 2021 to describe thermally coupled, relatively moving parts.

(ii) A linear, block structured operator $\mathscr{C}$, such that a QOI depends on the temperature fields in two parts. These QOIs can describe the heat flux between neighboring parts, in form of a scaled multiple of the temperature difference.

(iii) An unstructured operator, which correspond to displacements in certain important points. These displacements describe the translation and rotation of a tool. The operator is the application of the inverse of the discrete linear elasticity on a temperature field. Due to the mechanical coupling, the operator acts on all temperature fields.

Thus, the complete PDE model describes a scalar PDE in a moving domain with mixed boundary conditions and user defined output operators. Please note, that the coupling of different physics can be hidden in the output operators.

2.2 The ODE model

The discretization in space transforms the system of PDEs into a system of ODEs. This ODE is block structured, where every block row represents the contribution of the PDE for one solid. In the following, we will omit the superscripts and describe the construction of a single block.

The basis for the FE discretization in space is the weak formulation of Eq. (2.1) on the reference domain $\widetilde{\Omega}$, i. e.

$$\int_{\widetilde{\Omega}} \partial_t(\rho C_p T)\psi dx + \int_{\widetilde{\Omega}} (\lambda \cdot \nabla T)\cdot \nabla \psi dx = \int_{\widetilde{\Omega}} Q\psi dx + \int_{\widetilde{\Gamma}_N} g_N \psi dS, \tag{2.4}$$

with suitable test functions ψ. Finally, we represent the solution $T(t,x)$ in the form $T(t,x) = \sum_l \mathbf{T}_l(t)\varphi_l(x)$. Thus, the solution is decomposed into a time-dependent vector and a set of space-dependent, but time-independent, functions φ_l. The functions φ_l form a basis for the function space, which includes the Dirichlet conditions. The test functions ψ vanish on the Dirichlet boundary Γ_D.

Table 2.1. Expressions corresponding to the differential operators of Eq. (2.1a).

dependency	$\partial_t(\rho C_p T)$	$-\nabla\cdot(\lambda\cdot\nabla T)$
constant	$\rho C_p M_V \dot{\mathbf{T}}$	$\lambda L_V \mathbf{T}$
time	$\partial_t(\rho C_p) M_V \mathbf{T} + (\rho C_p)(t) M_V \dot{\mathbf{T}}$	$\lambda(t) L_V \mathbf{T}$

The dependencies of the material parameters, the sources, and the boundary conditions on the temperature, space or time can lead to more terms, or a higher complexity. Note that a space parameter also has a representation in the same form as the solution. Therefore, we can neglect the space dependency in the following and concentrate on the time- and solution-dependence.

The Tables 2.1 and 2.2 represent the FE discretization for the volume and surface expressions in Eq. (2.1). Please note the linearity or independence of the solution T. Therefore, the PDE is linear too, and the discretized system will remain linear in the states.

The expressions M_V, L_V, b_V, $b_{S,j}$ represent the space-dependent part of the FE discretization. We use the subscripts V, S and C to refer to the integrals over a volume, surface and surface intersections, respectively. The symbols M, A and b refer to the FE mass matrix, the discrete Laplacian matrix and the integral vectors of the test functions, i.e.

$$\begin{aligned}
[M_V]_{kl} &= \int_{\widetilde{\Omega}} \varphi_l \psi_k & [b_V]_k &= \int_{\widetilde{\Omega}} \psi_k \\
[M_{S,j}]_{kl} &= \int_{\widetilde{\Gamma}_j} \varphi_l \psi_k & [b_S]_k &= \int_{\widetilde{\Gamma}} \psi_k \\
[L_V]_{kl} &= \int_{\widetilde{\Omega}} \nabla\varphi_l \cdot \nabla\psi_k & [M_{C,m}^{(i,j)}]_{kl} &= \int_{\Gamma^{(i)}_{N,m}\cap\Gamma^{(j)}_{N,m}} \varphi_l^{(i)} \psi_k^{(j)}\,.
\end{aligned}$$

All matrices depend only the discretization, but not on material parameters. Thus for example the mass matrix is neither the usual capacity matrix, nor the mass matrix from mechanics.

The volume integrals on the left-hand side can be further decomposed into bodies to account for piecewise material parameters. Thus, for piecewise material, the expressions $\rho C_p M_V$ and λL_V in Table 2.1 can be decomposed into sums over all bodies.

The Dirichlet boundary conditions (2.1c) describe the value in all nodes on the surface Γ_D. Thus, the test functions ψ vanish on Γ_D. In turn the entries $[b_V]_k$ and $[b_S]_k$

Table 2.2. Expressions corresponding to the sources and the boundary conditions Eqs. (2.1a) and (2.1b).

	Q	$g_{N,j}$	$g_{D,j}$
constant	Qb_V	$g_{N,j}b_{S,j}$	$g_{D,j}b_{D,j}$
time	$Q(t)b_V$	$g_{N,j}(t)b_{S,j}$	$g_{D,j}(t)b_{D,j}$
$\alpha(t)T+\beta(t)$	$\alpha(t)M_D\mathbf{T}+\beta(t)b_V$	$\alpha(t)M_{S,j}\mathbf{T}+\beta(t)b_{S,j}$	—
$\alpha(t)\left(T^{(j)}-T^{(i)}\right)$	—	$\alpha(t)\left(M^{(i,j)}_{C,m}\mathbf{T}^{(j)}-M^{(i,i)}_{C,m}\mathbf{T}^{(i)}\right)$	—
nonlinear in T	$Q(y)b_V$	$g_{N,j}(y)b_{S,j}$	—

and kth rows of M_S and M_V vanish for $x_k \in \Gamma_D$. Instead we insert the condition $T_l = g_{D,j}(x_l)$ into L_V and a corresponding vector b_D, where the subscript D refers to the Dirichlet conditions. We replace the row k of L_V by the unit row and set b_D to the kth column of L_V afterwards.We refer the reader to Logg, Mardal, Wells, 2012, Section 6.3 for a local element approach. Please note that the Dirichlet conditions cause algebraic equations and therefore lead to a differential algebraic equation (DAE) system instead an ODE system. For the sake of presentation, we consider only systems of ODEs.

The Table 2.2 lists the expressions, which arise from the discretization of the sources and the boundary conditions. The rows are sorted with increasing dependency, where the time-dependence has lower complexity than temperature-dependence and a linear expression has also a lower complexity than a nonlinear one.

The fourth row in Table 2.2 represents the coupling between the thermal fields through a heat flux. The heat flux into part i is assumed to be linear in the temperature. Thus, the surface integrals for $M_{C,m}$ are restricted to some common surface between the components. Due to the relative movement, this surface is in general time-dependent, thus the matrices $M^{(i,j)}_{C_m}$ and $M^{(i,i)}_{C_m}$ are in general time-dependent too, even if the heat transfer coefficient α is constant. In an efficient implementation, this part is interpreted as an operator instead of a matrix.

The fifth row in Table 2.2 provides an additional approach to approximate the heat flux between different parts and accounts for a simplified approximation of nonlinear sources and boundary conditions. The convergence of an FE discretization requires an approximation of the integrals in Eq. (2.4) up to the required order, with a suitable quadrature formula. Thus, the nonlinear boundary condition g_N has to be evaluated in the quadrature points at the element level. Using piecewise constant temperatures, where the pieces are subdivisions of the domain (or surface), leads to a cheaper approximation at the costs of the order of the approximation.

The piecewise constant temperature on each piece is the average temperature, which can also serve as a QOI of the model. The same idea can be used to approximate the heat exchange. The coupling matrices $M^{(i,j)}_{C_m}$ depend on the meshes of both parts and the exchange between the meshes can be quite demanding. Thus, approximation uses the subdivision of the surface and replaces the pointwise temperatures $T^{(i)}$ and $T^{(j)}$ by their averages over the pieces on their meshes. This makes the meshes independent and the computations far cheaper. At the same time we can also represent the heat exchange between moving geometries without time dependent coefficients. The disadvantages of the piecewise constant approximation is the lower accuracy and the dependency of inputs in one block on outputs of another block.

Block ODE The full block system consists of all blocks from the previous paragraph combined with a set of external systems without any particular structure. Thus, we collect the terms by coefficients of $\dot{x}$, x, dependence on time t and for nonlinear T and outputs y in this order to obtain the first block in the ODE

$$M\dot{x} = (A_c + A_t(t))x + B_c u_c + B_t u_t(t) + B_y u_y(t,Y) \tag{2.5a}$$

$$y = Cx \tag{2.5b}$$

$$\dot{x}_{\text{ext}} = f_{\text{ext}}(t, x_{\text{ext}}, Y) \tag{2.5c}$$

$$y_{\text{ext}} = h_{\text{ext}}(x_{\text{ext}}). \tag{2.5d}$$

The second block consists of all right-hand sides, which are provided by external functions. A particular example for the external functions are the FMUs in model exchange mode. By contrast, an FMU in Co-Simulation mode contributes to the inputs u_y, where the subscript y denotes the dependency on outputs. Thus, the ODE consists of all terms of the form in Tables 2.1 and 2.2, but they are decomposed into the coefficients and the inputs. The matrix M is composed of all coefficients of the time derivatives, and is block diagonal. The matrices A_c and A_t are the block matrices, composed of the coefficients of $\mathbf{T}$, which includes the (negative) discrete Laplacians and the mass matrices M_S and M_C from the flux boundary conditions g_N. Please note the different origins of the time-dependence of the coefficients. These can originate from time-dependent heat transfer coefficients (HTCs), or from relatively moving parts. While the time-dependent HTCs lead to a time-dependent scalar coefficient with a constant surface mass matrix, the coupled surfaces lead to time-dependent operators, involving the integrals on the common surface.

The matrices B originate from the vectors b and commonly correspond to the inputs u of the ODE. This includes all sources and boundary conditions, irrespective of the type. The only differences, denoted by the subscripts c, t and y, are the dependencies on time and outputs. Note that the output dependencies u_y introduce an indirect de-

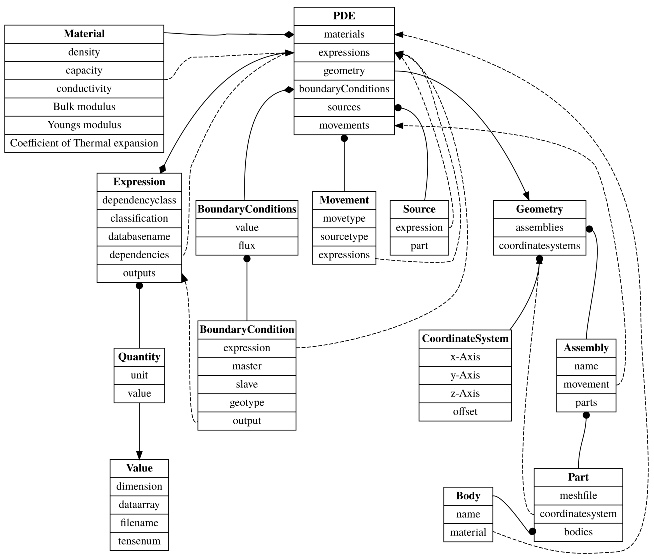


Figure 3.1. Relation between entities in the PDE model. Solid edges denote an aggregation, where the heads of the edges determine the aggregation types. A diamond head represents a dictionary, a circle head an array and an arrow corresponds to a single value of the type. Heads without marker correspond to single values of the referenced structur. Dashed lines represent indirect relations using keys of dictionaries or indices into an array.

pendence on the states and can render the ODE a nonlinear system, as well.

The QOIs $\mathscr{C}$ are represented by the operator C. We will consider only linear output operators, therefore C can be represented by a matrix.

3 Interfaces

The preceding section described the model equations for the PDE and the DAE. Each equation provides a structure and consists of several terms. Usually, the expressions in the terms correspond to physical properties. Thus, a hierarchical model description denotes the physical properties as objects, whereas the equations link them together. This section is devoted to the structure and content of the new model descriptions.

3.1 PDE model

The Eqs. (2.1) and (2.2) show all components of the PDE model:

- the reference domains $\widetilde{\Omega}^{(i)}$, which are approximated by mesh files,
- the movements, which are vector valued expressions, in contrast to the field valued sources and boundary conditions,
- the material parameters ρ, C_p and λ, which are scalar expressions,
- each boundary condition $g_{N,j}$ and $g_{D,j}$, which is a pair of a scalar expression and the reference to the corresponding sub-boundary.

The components of the PDE model are depicted in Fig. 3.1. The central node with the title "PDE" combines all other entries. The entry *materials* is separated from the remaining model, and only relates to the expressions.

The model joins all geometric properties and relations using the class *Geometry* and the entry *geometry* of the class PDE. As a consequence, the subgraph of the geometry contains the assemblies, which consist of parts and bodies. Each part is attached to a coordinate system, which is also part of the geometry subgraph.

The entry *boundaryConditions*, which is an instance of the class *BoundaryConditions*, contains value and flux conditions. Each condition references the corresponding expression and is attached to the *master*. The rank of these conditions must be the same, as the rank of the field. Thus, for the heat equation, the expression must be scalar, whereas for the equations of elasticity the expressions are vector valued.

We distinguish coupling from non-coupling boundary condition with the enum *geotype*. A value of *Face* represents loads, which are active on one surface, whereas *FaceFace* represents the coupling of two neighboring surfaces. The corresponding facing / opposite surface is determined by the entry *slave*. The boundary conditions and their associated expressions must match to their physics, respectively. Therefore, the description of mixed systems of PDEs modeling different physical quantities require an appropriate association. Please note the missing association with the physics. As our models are the equations of thermo-elasticity, we had no need to specify that explicitly.

All the assembly's movements are part of the list *movements*, where every entry is an instance of the class *Movement*. Each assembly references the local movement, which is relative to the kinematic parent. Thus, the global movement of an assembly is given by the successive composition of the local movements of all predecessors.

The expressions are the common structure between the PDE model and the ODE model. Due to their central nature, we explain them in detail in Section 3.3.

3.2 ODE model

We depict the ODE model in Fig. 3.2. The model consists of a *blockIO*, the *expressions* and *associations*. The mathematical basis is a block structured ODE as given by Eq. (2.5). Each matrix entry of the *blockIO* represents a block matrix. While the output matrices C and the mass matrices E are block diagonal, the coefficient matrices A_c and A_t might be dense. Thus, the former are arrays of submatrices, whereas the latter are arrays of arrays of submatrices. The matrices E, A_c and A_t have coefficients, which are stored in the arrays m, α and β, respectively. In analogy the input matrices B_c, B_t and B_y correspond to the inputs u_c, u_t and u_y. The sub-scripts refer to the dependencies of the expressions, i. e. constant, time-dependent and output-dependent. The associations B_y link the outputs to the expression. Please note the missing entry of the size

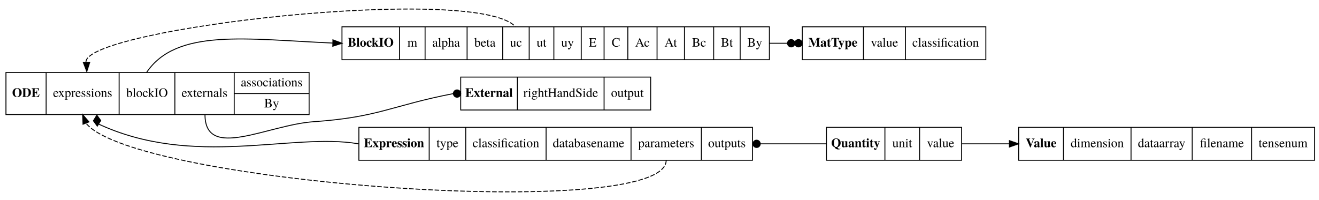


Figure 3.2. The relationship of the ODE model. The line and arrow styles are the same as in Fig. 3.1.

of the blocks. This must be inferred from the size of the sub-matrices.

This format proves to be very flexible and easy to extend, in a backward compatible way. Thus, it allows for the incorporation of advanced mathematical methods for the considered models. We highlight some aspects which were included in the data format to meet the method's special requirements using the example of MOR for linear time-invariant as well as linear parameter-varying systems.

MOR is used to compute low-dimensional surrogate models, which allow for accelerated simulations, see e.g. Antoulas, 2005; Benner, Grivet-Talocia, et al., 2021. System-theory based MOR methods require a model in input-output form as given in Eq. (2.5). Especially, in the case where a physical parameter is supposed to be preserved in the reduced-order model, i. e. in the parametric MOR (PMOR), the splitting into time-dependent coefficients β and constant matrices A_t coincides with the parameter-affine format proposed, e. g., in Benner, Gugercin, Willcox, 2015,. Then, the corresponding MOR methods, based on interpolation in parameter direction, come into consideration. Therefore, a possible parameter p and corresponding coefficients $\beta(p)$ were added to the model description. Moreover, these methods require a suitable parameter interval $[a,b]$. On this interval, local reduced-order models (ROMs) are computed in some parameter sample points. For a given parameter value $p \in [a,b]$ these sample ROMS are then interpolated in one way or another. These PMOR methods can, thus, be directly applied to the models at hand.

Furthermore, the expected magnitude of the inputs u is important for reliable ROMs, as the MOR error scales with, both, the system approximation error, and u, see Vettermann et al., 2021 and the references therein for further information. Thus, this magnitude is computed using the provided expressions u_c, u_t and u_y and is then used to transfer the scaling to the columns of the input matrices B_c, B_t and B_y prior to the MOR step, such that it can be taken into account in the system approximation and u itself is normalized.

The addition of the geometric coupling approaches, as stated in Section 2, allows for tailored MOR strategies utilizing the special structure of these models, see Vettermann et al., 2021. By including all necessary information in the ODE model-description the MOR methods can be applied in a user-friendly semi-automatic process.

3.3 Expressions

The expressions are the common structures, which are shared between the PDE and ODE model.

We separate the expressions with respect to two classifications

(i) The dependency on other expressions or solution fields with the entry *dependencyclass*. These classes are

- **Constant** are constant values, which do not depend on anything else.
- **Equationset** represents compound expressions, which involve further expressions. The dependencies between expressions are denoted by the map *dependencies*. These also include characteristic maps, which are represented by file names. FMUs are treated as a particular (complicated) expression. There, interaction with the solution is determined by the classification. The interaction between different FMUs is described by the dependency. Thus the expressions here are comparable to the transformations in the SSP-standard Association, n.d., p. 23.
- **Sensor** represents a linear functional of a solution field on one part. The specific geometry entity is described in the dependency map in analogy to the dependency on other expressions. Examples for these are the temperature average on a surface and the temperature value at a point.

(ii) The function classification, given by *classification* separates the expressions into time, time and one field value and time and two field values, or a vector of outputs. Thus, this classification concretises an expression of dependency class *Equationset*.

The first classification separates the expressions into simple, compound field-independent expressions and field-dependent expressions. Please note that an expression with dependency **Sensor** can exist only in a PDE model and is the only expression which can reference a geometric object. Therefore, during the ODE generation, the generator must keep track of all Sensor-type expressions and replace their occurrences by a reference to the corresponding output.

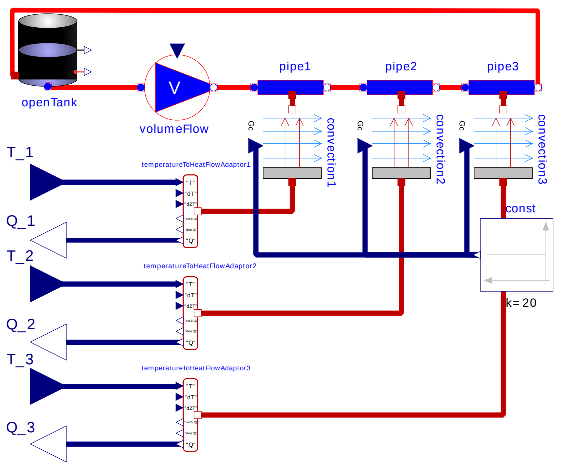


Figure 4.1. The cooling model with three flux outputs and three temperature inputs.

The second classification provides the interface for the function implementation, including the dependencies, to other expressions. Please note that this classification corresponds to the inputs u in the ODE system (2.5). At the same time, this classification implicitly serves as a notation for the dependency between the expressions and the solutions. Thus, when generating an ODE, an expression which depends on a solution value in the PDE model must be either decomposed according to Table 2.2 or related to an output.

Finally, an expression can have outcomes of different sizes or even different units. Every outcome is an entry in the list *outputs*, which is an instance of the class *Quantity*. A *Quantity* denotes the physical unit, and the description of the value. Each value consists of a vector *dimension*, which encodes the shape. Constant expressions also have a data-array, which contains the value, or a filename, if the values are stored elsewhere.

A numerical solution of an ODE requires the evaluation of the expressions. These implementations are encapsulated in an accompanying library, which represents a database of functors. This database maps a string to a functor with the signature according to the function classification (Item (ii)). The key of the corresponding functor is the entry *databasename* in the expression. It is up to the implementation of the database to provide further properties and operations, like linearity and derivatives with respect to constant arguments.

4 Two body model

To illustrate the usage of the interface formats described in Section 3, a simple model along with excerpts from json files. These files are exchanged between the interdisciplinary groups in the CRC/TR96. In detail, the json file Fig. 4.4 was generated using ANSYS®by one author, and used the code from a second author to generate Fig. 4.5. This ODE model will be fed to the MOR toolbox, written by the third author. Please note, that the ODE model could also be generated using ANSYS® directly.

The Fig. 4.2 shows the geometry of the example model. It comprises two assemblies, where each one consists of one part. Each part comprises two bodies, which are highlighted by the colors. Both assemblies exchange heat through a neighbouring surface with a linear temperature-dependent flux of the form $q = \alpha(T^{(1)} - T^{(2)})$.

The boundary conditions we applied are shown in Fig. 4.3. In this particular case, the cooling system shown in Fig. 4.1 is integrated as an FMU.

Collaborators in the authors research project constructed a complex cooling system in Shabi, Weber, Weber, 2017, which is part of the thermal model of a complex machine tool. For the sake of presentation, we simplified this model to a tank, an ideal pump and three pipes. The heat exchange between fluid and body walls is realized using convection subsystems in the Modelica® model.

The connection to the machine is two-fold:. The FMU inputs are the (averaged) temperatures at the corresponding surfaces. Internally, the FMU computes the heat fluxes for every pipe. These are the outputs of the FMU, and serve as the heat fluxes at the aforementioned surfaces.

Figure 4.4 depicts the PDE model. This relates to Figs. 4.2 and 4.3 using the same colors. We added the FMU using a single expression and reference the corresponding outputs in the boundary condition. As a consequence, practically, the FMU is evaluated only once.

The ODE model in Fig. 4.5 shows the relation of the coefficient matrix to the piecewise materials. In addition, we also highlight the corresponding expressions for the boundary conditions. As can be seen from the coefficient *alpha*, we decompose the coupling heat flux and use only the coefficient. Also note the change of the FMU expression. Instead of multiple outputs, we use a single vector valued output and classify the expression to depend on time and outputs.

5 Summary and outlook

This contribution describes a PDE model and an ODE model and introduces a data format to exchange each of them. Each model type comprises all entities arising in the mathematical model except the equations. Their relation is therefore only part of the documentation.

The simple two body model showed the basic thermal properties and a thermal load given as a simplified cooling system, which is provided as an FMU. Thus, the whole complexity, and possible intellectual properties, remain hidden in the FMU. This very simple example shows that both descriptions are general enough to describe geometrically coupled thermal problems with very complex loads, developed by collaborators in the same research project. Despite the simplicity, the example model also highlights the successful simplification of the model exchange between mathematicians and engineers. The engineers got access to state-of-the-art open source implementations of FE discretizations and MOR techniques. In the other di-

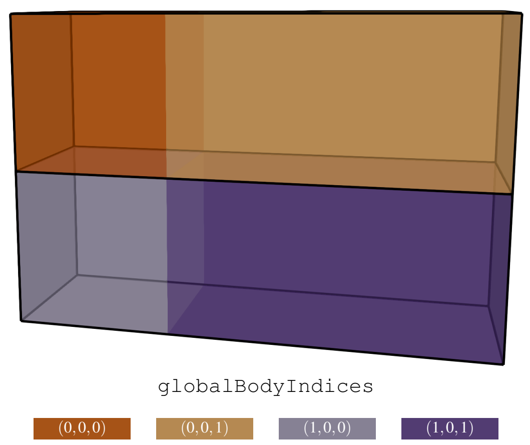


Figure 4.2. The geometry of the example model. Each color represents one body, whereas the legend indices denote the logical position as (assembly, part, body) in the description.

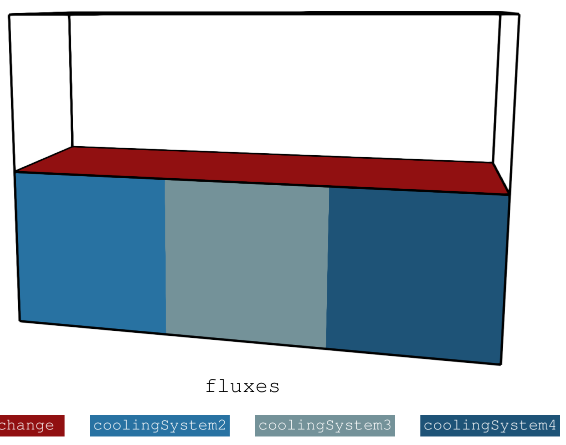


Figure 4.3. The boundary conditions in the example model. The colors red and light to dark blue represent the heat exchange and the cooling boundary conditions, respectively.

```
{
 "geometry": {
  "assemblies": [
   {
    "parts": [
     {
       "meshfile": "coupledflux_top.json",
       "name": "top",
       "bodies": [
        {"name": "left", "material": "steel" },
        {"name": "right", "material": "alu" }
       ],
       "coordinatesystem": 0
     }
    ],
    "name": "assembly"
   },
   {...}
   }
  ],
  "coordinatesystems": [...]
  },
 },
 "materials": {...},

                        :
```

```
                    :
"boundaryConditions": {
  "flux" :[
   {
    "master": [0, 0, 0],
    "expression": "alphaEx",
    "name": "alphaEx_0_1",
    "slave": [1, 0, 0],
    "geotype": "FaceFace"
   },
   {
    "master": [1, 0, 1],
    "expression": "coolingSystem",
    "name": "coolingSystem2",
    "output" : 0
    "geotype": "Face"
   },
   {
    "master": [1, 0, 2],
    "expression": "coolingSystem",
    "name": "coolingSystem4",
    "output" : 1
    "geotype": "Face"
   },
   {...}
  ]
},           :
```

```
                          :
"expressions": {
  "steel_capacity": {...},
  "alphaEx": {
    "dependencyclass": "Equationset",
    "classification": "timeTempTemp",
    "databasename": "chiAlphaToMT",
    "outputs": [{
       "value": {"tensenum": "Scalar"},
       "unit": {"unitstring": "kg s^-3"}
    }],
    "arguments": ["alphaIkb"]
  },
  "coolingSystem" : {
    "dependencyclass" : "Equationset",
    "classification" : "timeTemp",
    "databasename": "fmuWithTemp",
    "dependencies" : [...],
    "outputs": [{
       "value": {"tensenum": "Scalar"},
       "unit": {"unitstring": "kg s^-3"}
    }, {...}, {...}]
  },
    ...
 },
 "sources": []
 "movements": [] }
```

Figure 4.4. Excerpt from json-file used to exchange models in the PDE-Interface format. The colors mark the corresponding entities in Fig. 4.2 and Fig. 4.3.

```
{
 "blockIO": {
  "m": [
   ["BG_0_mass_factor_1", "BG_0_mass_factor_2"],
   ["BG_1_mass_factor_1", "BG_1_mass_factor_2"]
  ],
  "alpha": [
   [["BG_0_lapl_factor_1", "BG_0_lapl_factor_2", "alphaIkb"], ["alphaIkb"]]
   [["alphaIkb"], ["BG_1_lapl_factor_1", "BG_1_lapl_factor_2", "alphaIkb"]]
  ],
  "uy": [
   [],
   ["coolingSystem"]
  ],
  "Ac": [...],
  :
  },
  "externals": [],
  "associations": {
   "By": [
    [],
    [[[1, 0], [1, 1], [1, 2]]]
   ],
   :
  },
                       :
```

```
                 :
  "expressions": {
   "alphaIkb": {...},
   "alphaEx": {
    "dependencyclass": "Equationset",
    "classification": "timeTempTemp",
    "databasename": "chiAlphaToMT",
    "outputs": [{
      "value": {"tensenum": "Scalar"},
      "unit": {"unitstring": "kg s^-3"}
    }],
    "arguments": ["alphaIkb"]
   },
   "coolingSystem" : {
    "dependencyclass" : "Equationset",
    "classification" : "timeOutputs",
    "databasename": "fmuHandler",
    "outputs": [{
      "value": {"tensenum": "Vector", "dimension": [3]},
      "unit": {"unitstring": "m^2 kg s^-3"}
    }]
   },
   :
  }
 }
```

Figure 4.5. Excerpt from json-file used to exchange models in the ODE-Interface format. The colors mark the corresponding entities in Fig. 4.2 and Fig. 4.3

rection, the mathematicians optimized sensor placements for the same models.

The main advantage is the simplicity of the underlying data format. This simplicity renders the (ODE) description easily extendable with further entries for more sophisticated methods, like parametric MOR approaches.

Nevertheless models with systems of PDEs describing various physics require extensions of the PDE model.

We concentrated on the equations for thermo-elasticity, where the elasticity is assumed to be linear and stationary. In that particular case, boundary conditions apply either to the heat equation or the equations of linear elasticity. Thus, they can be distinguished by the dimension and unit of the associated expression. In a complicated model, an additional property would be less error prone and might be used for consistency checks.

In particular physically coupled models, like thermo-elastic models, require the addition of another kind of physics. These require the extension of the boundary conditions to assign them to the particular physics. The thermo-elastic models with stationary equations of elasticity can already be described using the output operators in the ODE model. At the same time the space-discrete dynamic equations of linear elasticity are second-order systems. Thus, the ODE model requires additional coefficients to represent these.

References

Antoulas, A. C. (2005). *Approximation of Large-Scale Dynamical Systems*. Vol. 6. Advances in Design and Control. With a foreword by Jan C. Willems. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM). DOI: 10.1137/1.9780898718713.

Association, M. (n.d.). *System Structure and Parameterization*. Version 1.0. URL: https://ssp-standard.org/publications/SSP10/SystemStructureAndParameterization10.pdf.

Benner, P.; S. Grivet-Talocia; A. Quarteroni; G. Rozza; W. Schilders; L. M. Silveira, eds. (Oct. 2021). *Model Order Reduction. Volume 1: System- and Data-Driven Methods and Algorithms*. De Gruyter. DOI: 10.1515/9783110498967.

Benner, P.; S. Gugercin; K. Willcox (Jan. 2015). "A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems". *SIAM Review* 57.4, pp. 483–531. DOI: 10.1137/130932715.

Grossmann, C.; H.-G. Roos (1994). *Numerik partieller Differentialgleichungen*. Stuttgart: Teubner. DOI: 10.1007/978-3-322-96752-7.

Hvalstad-Nilsen, O. (2019). "YAML Based Input File Format For Finite Element Analysis". MA thesis. NTNU Norwegian University of Science and Technology.

Li, Z.; H. Zhang; L. Zheng (2008). "Description of PDE models in Modelica". *2008 International Symposium on Computer Science and Computational Technology*. Vol. 1. IEEE, pp. 809–813.

Logg, A.; K.-A. Mardal; G. N. Wells, eds. (2012). *Automated Solution of Differential Equations by the Finite Element Method*. Springer. DOI: 10.1007/978-3-642-23099-8.

Michopoulos, J.; P. Mast; T. Chwastyk; L. Gause; R. Badaliance (2001). "FemML for data exchange between FEA codes". *ANSYS Users' Group Conference*.

Naumann, A.; R. Herzog (Jan. 2021). "Optimal sensor placement for thermo-elastic coupled machine models". *PAMM* 20.1. DOI: 10 . 1002/pamm . 202000255.

Pinheiro, M. C.; G. F. Moita (2004). "Usando XML para criação de um padrão para o intercâmbio de dados entre programas de elementos finitos". *Educação & Tecnologia* 9.1.

Portela, A.; A. Charafi (2002). *Finite Elements Using Maple*. Springer Berlin Heidelberg. DOI: 10 . 1007/978-3-642-55936-5.

Saldamli, L.; B. Bachmann; H. Wiesmann; P. Fritzson (2005). "A framework for describing and solving pde models in modelica". *Paper presented at the 4th International Modelica Conference*.

Sauerzapf, S.; J. Vettermann; A. Naumann; J. Saak; M. Beitelschmidt; P. Benner (2020). "Simulation of the thermal behavior of machine tools for efficient machine development and online correction of the tool center point (TCP)-displacement". euspen Special Interest Group Meeting: Thermal Issues. Aachen, Germany, pp. 135–138.

Shabi, L.; J. Weber; J. Weber (2017). "Model-based Analysis of Decentralized Fluidic Systems in Machine Tools". *Proceedings of 15: th Scandinavian International Conference on Fluid Power, June 7-9, 2017, Linköping, Sweden*. 144. Linköping University Electronic Press, pp. 116–123.

Vettermann, J.; S. Sauerzapf; A. Naumann; J. Saak; P. Benner; M. Beitelschmidt; R. Herzog (Apr. 20, 2021). "Model order reduction methods for coupled machine tool models". Special Issue ICTIMT 2021.

Zienkiewicz, O. C.; J. Z. Zhu (1987). "A simple error estimator and adaptive procedure for practical engineering analysis". *International Journal for Numerical Methods in Engineering* 24.2, pp. 337–357. DOI: 10.1002/nme.1620240206.

Optimal Design for Thermodynamic System with OpenModelica and MATLAB/Simulink

Kazuki Yoshida[1] Hitoi Ono[1] Hirotaka Okazaki[1]

[1] Mitsubishi Heavy Industries, Ltd., Japan, {kazuki.yoshida.ws, hitoi.ono.x3, hirotaka.okazaki.hy}@mhi.com

Abstract

The complicated thermodynamic system includes nonlinear characteristics and is expressed by high order differential algebraic equations. Therefore, it is difficult to carry out numerical analysis such as optimization. To deal with this problem, we verify an integrated design support environment incorporating OpenModelica for a centrifugal chiller as an example. In this paper, it is shown that the parameters of the model and the control logic can be adjusted by the coupled simulation of the chiller model written in the Modelica language and the control logic on MATLAB/Simulink. We also simplify the centrifugal chiller model by approximating the nonlinear characteristics with a smooth polynomial, and by reducing the order of the differential algebraic equation by the Pantelides method. Then, a case study of a startup profile optimization is shown.

Keywords: thermodynamic system, optimization, design tool, FMI

1 Introduction

There is an increasing demand for efficient operation of energy equipment for the transition to a carbon-free society. In order to develop and commercialize a device composed of a wide variety of equipment in a short time, it is important to use dynamic simulation in the early stage of development, and determine the feasibility including operation characteristics. In general, this dynamic simulation uses a precise model close to the actual machine. However, in performing a mathematical optimization in order to adjust model parameters based on actual machine data or to generate an operation profile, a smooth approximate model suitable for optimization is required. These approximation models were developed manually and took time.

In this study, an integrated design support environment incorporating OpenModelica is verified using a centrifugal chiller as an example. Figure 1 shows the environment's work flows and main 3 features, (1) Profile optimization, (2) Logic verification and (3) Software-In-the-Loop-Simulation (SILS) verification.

First, a detailed dynamic model of the chiller is developed in the Modelica language (Fritzson, 2004). This model and a control logic are coupled with MATLAB/Simulink. With this coupling calculation, (1) Validation of control logic, (2) Fitting model parameters by mathematical optimization, and (3) Optimization of control parameters are possible. Either the control logic defined by MATLAB/Simulink or the execution file of control logic for real machine (regardless of programming language) can be used. The latter is SILS verification. In the optimization of the control parameters, optimization may be performed to match the optimal profile (described later) and the state variables of coupling simulation. This paper shows a parameter optimization without using an optimal profile.

Next, in performing profile optimization, simplified models, in which the nonlinear characteristics are approximated by a smooth polynomial, are prepared. Further, since the combined whole system model becomes a high-order differential algebraic equation (DAE) having two or more differential exponents, a mathematical processing of reducing the order is applied. Then, simplified model is exported as XML format by OpenModelica (Shitahun et al., 2013). The profile optimization problem for chiller startup is formulated from this XML file by using collocation method (Sabbagh and Gómez, 2018). Next, this problem is converted into Python script, which can be treated by optimization modeling tool CasADi (Andersson et al., 2012). Finally, optimized profile is calculated by CasADi and nonlinear programming (NLP) solver IPOPT (Wächter and Biegler, 2006)

The remainder of this paper is structured as follows: The configuration and model of the turbo chiller, and the simulation by Modelica are shown in Section 2. In Section 3, we present the tuning of equipment parameters and control parameters by using the coupled calculations on MATLAB/Simulink. Section 4 describes profile optimization using an approximate model. Section 5 shows a summary and future perspective.

2 Centrifugal chiller dynamic model

2.1 Process

As shown in Figure 2, the centrifugal chiller has two heat exchangers (evaporator, condenser) and a gas-water separator (economizer), and a two-stage centrifugal compressor and three valves (high-stage expansion valve, low-stage expansion valve, and hot gas bypass valve) (Okazaki et al., 2022). The non-CFC (Chlorofluorocar-

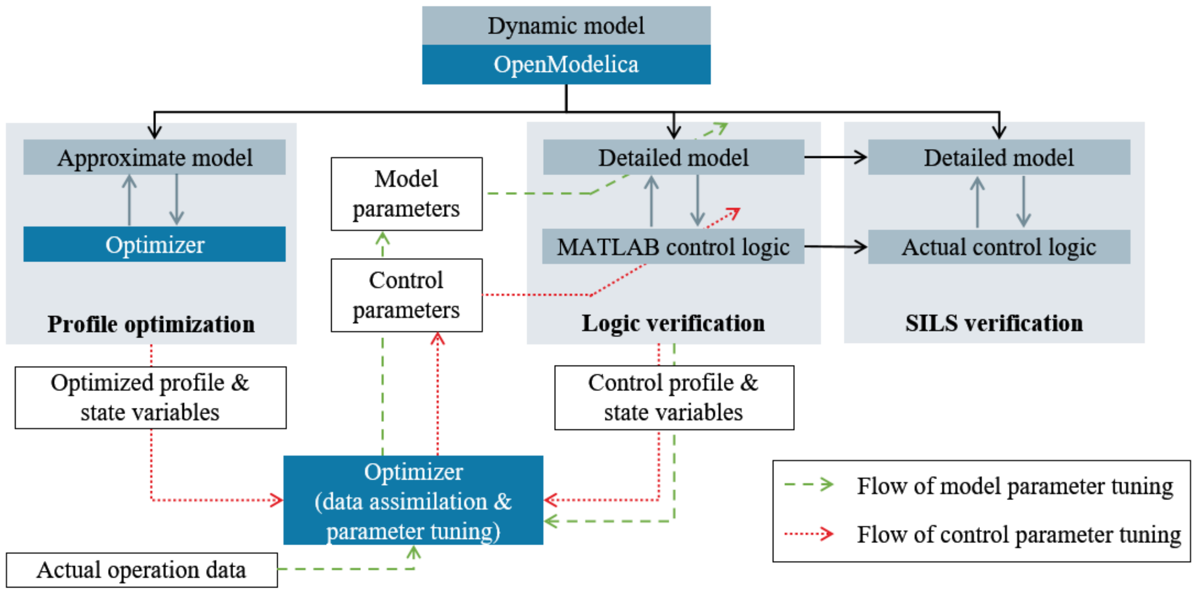


Figure 1. Work flows and features of the integrated design support environment

bon) type refrigerant circulating in the centrifugal chiller evaporates by heat exchange with chilled water in the evaporator and is sucked into the 1st stage compressor. The refrigerant adiabatically compressed by the two-stage compressor maintains the gas phase state and flows into the condenser. In the condenser, the refrigerant condenses by removing heat with cooling water. The refrigerant in the liquid phase state is depressurized by the high stage expansion valve, and a part of the refrigerant is vaporized. The vaporized refrigerant is separated from the liquid phase by the economizer and sucked into the middle stage of the compressor, and the liquid phase is further depressurized by the low stage expansion valve and flows into the evaporator. Chilled water that has been cooled to 7°C by being deprived of heat by the refrigerant in the evaporator is supplied to the HVAC (Heating Ventilation and Air Conditioning) system. By cooling the air conditioning air, the chilled water raises itself 12°C and returns to the evaporator. The cooling water rises to 37°C after it is used to cool the refrigerant in the condenser. In the cooling tower, the temperature is lowered to 32°C by a heat exchanger with air and returned to the condenser. The control of the centrifugal chiller is to maintain the chilled water outlet temperature at the set value temperature of 7°C against fluctuations in the cooling load, and the rotation speed of the compressor is manipulated. Furthermore, in order to operate the centrifugal chiller at a higher efficiency point, the vane opening of the two-stage compressor and the opening of the high stage expansion valve and the low stage expansion valve are manipulated to adjust the refrigerant circulation flow rate. The hot gas bypass valve is used to avoid a surge in the compressor by opening it at a low flow rate such as when starting up or when the load is low.

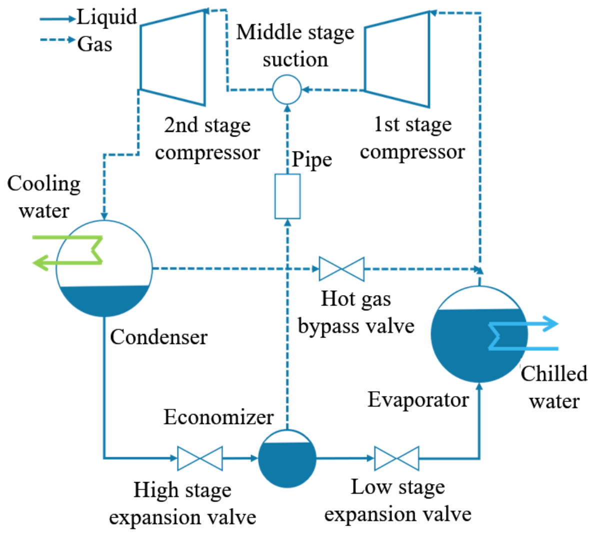


Figure 2. Configuration of centrifugal chiller

2.2 Physical models

Among the equipment constituting the centrifugal chiller, the evaporator, the condenser, and the economizer have a liquid level, and the refrigerant is in a two-phase state. In this study, it is assumed that the refrigerants in these devices are saturated, and the mass balance and energy balance are considered by the following Equation 1 and 2. Here, ρ (kg/m^3) is the average density of the liquid phase and the gas phase in the device, and E (kJ/m^3) is the average energy density. V (m^3) is a volume of equipment, G (kg/sec) is a mass flow rate, and h (kJ/kg) indicates specific enthalpy. Subscript letters in and out indicate inflow and outflow to the equipment. Q (kJ/sec) indicates the amount of heat for cold water, cooling water and refrigerant in the evaporator or condenser (Bendapudi et al., 2002).

$$V \cdot \frac{d\rho}{dt} = (G_{\mathrm{in}} - G_{\mathrm{out}}) \tag{1}$$

$$V \cdot \frac{dE}{dt} = (G_{\mathrm{in}} \cdot h_{\mathrm{in}} - G_{\mathrm{out}} \cdot h_{\mathrm{out}} + Q) \tag{2}$$

ρ and E are expressed according to Equation 3 and 4 using void fraction β, the density and the enthalpy of the liquid phase and the gas phase in the saturation state. Subscript letters l and g indicate liquid and gas phase.

$$\rho = \beta \cdot \rho_{\mathrm{g}} + (1-\beta) \cdot \rho_{\mathrm{l}} \tag{3}$$

$$E = \beta \cdot \rho_{\mathrm{g}} \cdot h_{\mathrm{g}} + (1-\beta) \cdot \rho_{\mathrm{l}} \cdot h_{\mathrm{l}} \tag{4}$$

When β is erased from these two equations, ρ and E are represented by the following Equation 5. Since ρ_{g}, ρ_{l}, h_{g}, and h_{l} can be approximated by a polynomial of pressure P (MPa), it is possible to obtain P and β from ρ and E, which are independent variables.

$$E = \frac{(\rho_{\mathrm{g}} \cdot h_{\mathrm{g}} - \rho_{\mathrm{l}} \cdot h_{\mathrm{l}})}{(\rho_{\mathrm{g}} - \rho_{\mathrm{l}})} \cdot \rho + \frac{(\rho_{\mathrm{g}} \cdot \rho_{\mathrm{l}} \cdot h_{\mathrm{l}} - \rho_{\mathrm{l}} \cdot \rho_{\mathrm{g}} \cdot h_{\mathrm{g}})}{(\rho_{\mathrm{g}} - \rho_{\mathrm{l}})} \tag{5}$$

The chilled water outlet temperature T_{weo} (°C) and the cooling water outlet temperature T_{wco} can be calculated from the following energy balance Equation 6 and 7 using the amount of heat exchange in the evaporator Q_{e} (kJ/sec) and condenser Q_{c} , respectively. Here, H_{e} and H_{c} indicate the heat capacity of the metal of the heat transfer tube and chilled and cooling water in the tube. G_{we} (kg/sec) and G_{wc} are the chilled water flow rate and the cooling water flow rate, and cp_{we} (kJ/kg/°C) and cp_{wc} are the specific heat respectively.

$$H_{\mathrm{e}} \cdot \frac{dT_{\mathrm{weo}}}{dt} = G_{\mathrm{we}} \cdot cp_{\mathrm{we}} \cdot (T_{\mathrm{wei}} - T_{\mathrm{weo}}) + Q_{\mathrm{e}} \tag{6}$$

$$H_{\mathrm{c}} \cdot \frac{dT_{\mathrm{wco}}}{dt} = G_{\mathrm{wc}} \cdot cp_{\mathrm{wc}} \cdot (T_{\mathrm{wci}} - T_{\mathrm{wco}}) - Q_{\mathrm{c}} \tag{7}$$

Q_{e} and Q_{c} are calculated by the following Equation 8 and 9 from the chilled water input T_{wei}, output temperature, the cooling water input T_{wci}, output temperature, evaporation temperature T_{e}, condensation temperature T_{c}, and heat transfer performance UA_{e} (kJ/sec/°C) and UA_{c}. f_{e} and f_{c} are functions for calculating the temperature difference between the refrigerant, chilled water, and cooling water.

$$Q_{\mathrm{e}} = UA_{\mathrm{e}} \cdot f_{\mathrm{e}}(T_{\mathrm{wei}}, T_{\mathrm{weo}}, T_{\mathrm{e}}) \tag{8}$$

$$Q_{\mathrm{c}} = UA_{\mathrm{c}} \cdot f_{\mathrm{c}}(T_{\mathrm{wci}}, T_{\mathrm{wco}}, T_{\mathrm{c}}) \tag{9}$$

The mass flow rate G_{cpj} passing through each stage of the compressor is calculated from the differential pressure ΔP_{cpj} (MPa) at the inlet and outlet, the specific volume υ_{cpj} (m^3/kg), the rotation speed N_{cp} (%), the vane opening degree V_{cpj} (%), and the flow rate characteristic map f_{cpfj} of the compressor by Equation 10.

$$G_{\mathrm{cpj}} = f_{\mathrm{cpfj}}(\Delta P_{\mathrm{cpj}}, \upsilon_{\mathrm{cpj}}, N_{\mathrm{cp}}, V_{\mathrm{cpj}}) \tag{10}$$

The mass flow rate G_{ev} through the expansion valve is calculated from the differential pressure ΔP_{ev} at the inlet and outlet, the elevation difference ΔL_{ev} (MPa), the specific volume υ_{ev}, the expansion valve opening V_{ev} , and the CV value determined from the CV characteristics f_{ev} by Equation 11.

$$G_{\mathrm{ev}} = \frac{f_{\mathrm{ev}}(V_{\mathrm{ev}})}{\upsilon_{\mathrm{ev}}} \sqrt{\Delta P_{\mathrm{ev}} + \Delta L_{\mathrm{ev}}} \tag{11}$$

The mass flow rate G_{by} through the bypass valve is calculated from the condensation pressure P_{c}, evaporation pressure P_{e}, the specific volume υ_{c}, the bypass valve opening V_{by} , and the CV characteristics f_{by} by Equation 12.

$$G_{\mathrm{by}} = \frac{f_{\mathrm{by}}(V_{\mathrm{by}})}{\upsilon_{\mathrm{c}}} \sqrt{P_{\mathrm{c}}^2 - P_{\mathrm{e}}^2} \tag{12}$$

The outlet enthalpy of the compressor h_{cpoj} is calculated by the following Equation 13 and 14 from the inlet enthalpy h_{cpij} , the value assuming the isentropic change h^{id}_{cpoj}, and the efficiency determined η_{cpj} from the compressor characteristics $f_{\mathrm{cp}\eta\mathrm{j}}$ (Bendapudi and Braun, 2002).

$$h_{\mathrm{cpoj}} = h_{\mathrm{cpoj}} + \frac{(h^{id}_{\mathrm{cpoj}} - h_{\mathrm{cpoj}})}{\eta_{\mathrm{cpj}}} \tag{13}$$

$$\eta_{\mathrm{cpj}} = f_{\mathrm{cp}\eta\mathrm{j}}(\Delta P_{\mathrm{cpj}}, N_{\mathrm{cp}}, V_{\mathrm{cpj}}) \tag{14}$$

2.3 Approximation of non-linear characteristics

In general, since the pressure loss is proportional to the square of the flow rate, the relationship between the differential pressure ΔP and the flow rate G is expressed as bellow (Okazaki et al., 2022).

$$G \propto \mathrm{sign}(\Delta P)\sqrt{|\Delta P|} \tag{15}$$

It is hard to optimize with Equation 15 since the slope of the right-hand side of the expression is ∞ when ΔP is 0. So, we represent Equation 15 to as bellow, firstly. The square root calculation is replaced by ReLU function. The ReLU function returns 0 when a negative value is an argument, or returns the input value if a positive value is an argument (Glorot et al., 2011).

$$\mathrm{sign}(\Delta P)\sqrt{|\Delta P|} = \sqrt{\mathrm{ReLU}(\Delta P)} - \sqrt{\mathrm{ReLU}(-\Delta P)} \quad (16)$$

The entire square root function is represented as a smooth curve, Equation 18, by replacing the ReLU function with the Softplus function of Equation 17.

$$Softplus(x) = \frac{\log\left(1+e^{kx}\right)}{k} \quad (17)$$

$$\begin{aligned}\mathrm{sign}(\Delta P)\sqrt{|\Delta P|} &\approx \sqrt{Softplus(\Delta P)} - \sqrt{Softplus(-\Delta P)} \\ &= \frac{\sqrt{\log\left(1+e^{k\Delta P}\right)} - \sqrt{\log\left(1+e^{-k\Delta P}\right)}}{\sqrt{k}}\end{aligned} \quad (18)$$

k is a value greater than 0 and is a parameter which has trade-off between the approximation accuracy of the function and the smoothness of the approximation function. Secondly, an approximation method for upper and lower limit processing of values such as state variables is described below. At the boundary where the upper and lower limit processing is performed, the gradient becomes discontinuous and the second order differentiation becomes impossible. The function that limits x to the upper and lower bounds in the $[L_x,U_x]$ interval can be written as follows.

$$\min(\max(x,L_x),U_x) = -\max(-\max(x,L_x),U_x) \quad (19)$$

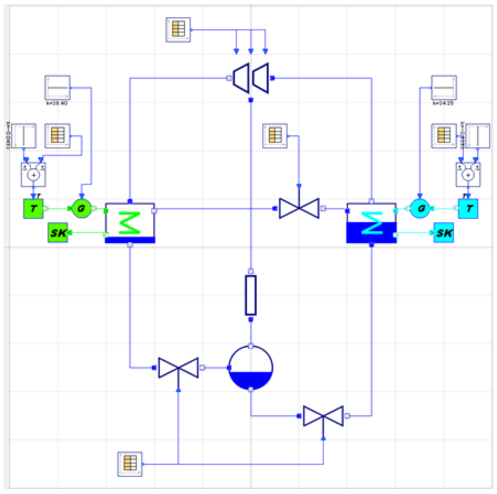
By replacing max in Equation 19 with LSE (Log-Sum-Exp) of Equation 20, it can be expressed in Equation 21 (Nielsen and Sun, 2016).

$$LSE_k(x_1,\cdots,x_n) = \frac{1}{k}\log\left(e^{kx_1}+\cdots+e^{kx_n}\right) \quad (20)$$

$$\begin{aligned}\min(\max(x,L_x),U_x) &= -\max(-\max(x,L_x),-U_x) \\ &= -LSE(-LSE(x,L_x),-U_x) \\ &= -\frac{1}{k}\log\left(\frac{1}{e^{kx}+e^{kL_x}} + e^{-kU_x}\right)\end{aligned} \quad (21)$$

Finally, we will describe that a model representing the backward enthalpy of this centrifugal chiller system can be expressed using a sigmoid function. This system may allow refrigerant to flow back, in which case the enthalpy flow must also be considered. It is necessary to switch the characteristics according to the direction in which the flow of the refrigerant. However, and the slope is discontinuous at the boundary of the switch. Thus, the representing characteristic switching model is expressed with Equation 22.

$$\mathrm{sigmoid}(x) = \frac{1}{1+e^{-kx}} \quad (22)$$



(a) Centrifugal chiller

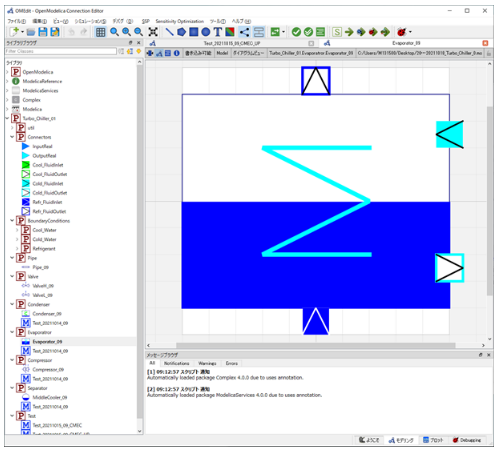
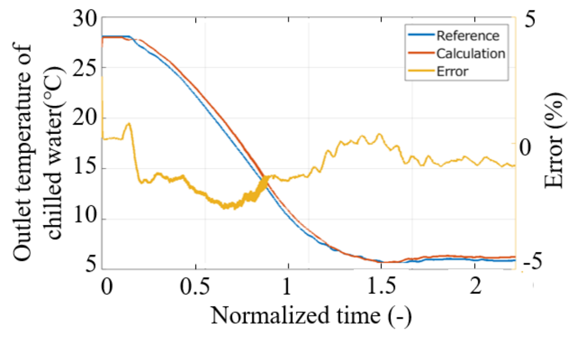

(b) Evaporator

Figure 3. Models on OpenModelica

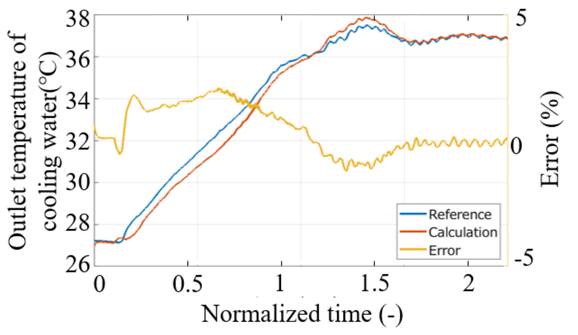
2.4 Validation

We implement the model of the centrifugal chiller in the previous section in Modelica language, and verify the model accuracy. The centrifugal chiller model on OpenModelica is shown in Figure 3(a). Considering deployment to chillers with different configurations, we implement each component. Figure 3(b) shows the evaporator component, which has the inlet and outlet to be connected to other components. Next, the components were combined in OpenModelica's GUI editor (OMEdit) to build a model of the entire chiller. The entire model takes as input the number of revolutions of the compressor, the opening of the bypass valve, the coolant inlet temperature, etc. The numbers of differential equations and algebraic equations are 9 and 100, respectively.

Next, we execute the startup simulation, giving the same conditions of the actual machine, such as the inlet temperature and flow rate of chilled water and cooling water, and the command value from the control logic, as boundary conditions. The simulation results and the measurement results of the actual machine are shown in Figure 4. The normalized time on the horizontal axis of



(a) Temperature of chilled water



(b) Temperature of cooling water

Figure 4. Validation results of centrifugal chiller model

the graphs represent the operation start point as 1. (In all the of following figures, the scaling factor is equal to that of Figure 4.) Both errors in the outlet temperature of chilled water and cooling water are within ± 2.5%. When the simulation period is 1800 seconds, the calculation time is 13.6 seconds (compilation: 12.0 seconds, and simulation: 1.6 seconds) with Intel Core i7-8700 CPU with 16 GB RAM machine.

3 Parameter tuning by coupled calculation

In this section, we describe how to perform coupled calculation of the model in Modelica language and the control logic on MATLAB/Simulink. Next, we present an example in which the parameters of the model are tuned to match the behavior of model to that of the actual machine. Finally, an example of optimizing the parameters of the control logic is shown.

3.1 Coupling of chiller dynamic model and control logic

The construction of coupled calculation using MATLAB/Simulink is shown. The model of centrifugal chiller is exported in FMU format by OpenModelica. We use a executable file of the control logic. Next, the FMU file of the centrifugal chiller is imported in the FMU block of Simulink of the MATLAB R2019b. The control logic communicates with S-function block of Simulink. The simulation is performed while the centrifugal chiller model and the control logic synchronize the time.

3.2 Tuning of equipment parameters

The procedure for adjusting the equipment parameters is as follows: (1) Startup simulation is performed, while the same inputs from control logic as the actual machine test is given to the centrifugal chiller model. (2) The equipment parameters are optimized to minimize the error between the simulation results and the results of real machine tests.

We tried the adjustment of each heat transmission coefficient of condenser and evaporator. The evaluation function is as follows:

$$\text{minimize} \int (|P_c - \hat{P}_c| + |P_e - \hat{P}_e|)dt \tag{23}$$

Here, P_c and P_e are the pressure of condenser and the pressure of evaporator in the real machine test, respectively. Also, $\hat{P}_c$ and $\hat{P}_e$ are the pressure of condenser and the pressure of evaporator in the simulation, respectively. That is, heat transmission coefficients are optimized to minimize the pressure errors. The Bayesian optimization algorithm is used.

We use the Bayesian optimization algorithm (Shahriari et al., 2016) because it is difficult to calculate the gradient of the evaluation function. In this algorithm, a response surface is generated by Gaussian process regression from the sampled data. The response surface represents the predictive uncertainty, and then the next sampling point is determined while considering the balance between the exploration of the high uncertainty region and the exploitation of already obtained optimal solution.

After 800 optimization iterations, we obtain the heat transmission coefficients that reduce the evaluation function from 4.16 to 0.64. Figure 5(a) compares condenser pressures of actual machine and these of before and after tuning. Figure 5(b) also compares these of evaporator pressure. The pressure errors of these 2 equipments are reduced after optimization. The optimization time is 1280 seconds.

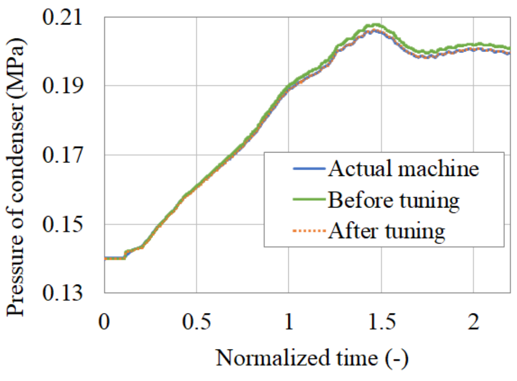
3.3 Tuning of control parameters

We optimize the parameters of the control logic by using the coupled simulation to reduce the startup time of the centrifugal chiller. We choose 12 parameters, which have a large effect on the startup time. The evaluation function is the difference between the chilled water outlet temperature T_{wco} and the target temperature $T_{wco,ref}$ as follows:

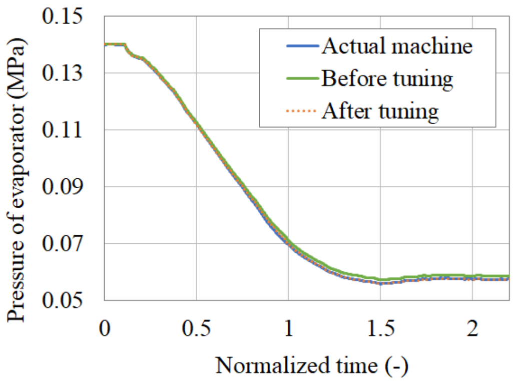
$$\text{minimize} \int |T_{wco} - T_{wco,ref}|dt \tag{24}$$

Here, $T_{wco,ref}$ was set at 7 °C. The Bayesian optimization algorithm is used.

After 40 optimization iterations, the parameters to reduce startup time were obtained. The simulation results

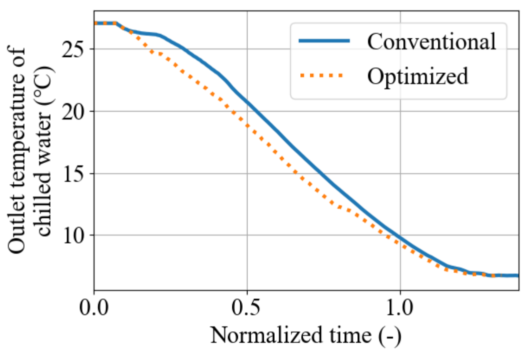


(a) Pressure of condenser

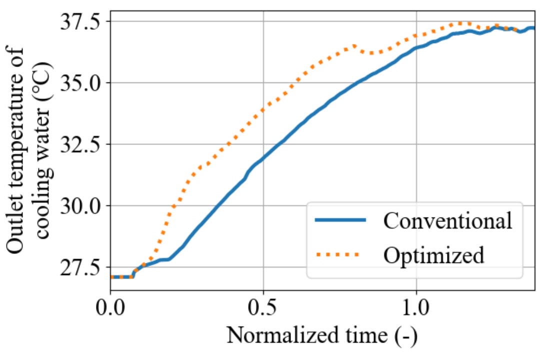


(b) Pressure of evaporator

Figure 5. Simulation results by tuned equipment parameters



(a) Temperature of chilled water



(b) Temperature of cooling water

Figure 6. Simulation results by tuned controller parameters

using the parameters before and after the optimization are shown in Figure 6. After the optimization, the outlet temperature of chilled water decreases faster to target value.

4 Dynamic optimization by approximation model

4.1 Approximation of non-linear characteristics

To improve convergence and reduce calculation time of complex dynamic system optimization, it is effective to approximate nonlinear characteristics with smooth polynomials. Therefore, we create approximate models, for example, property tables, pressure losses in valve and pipe (calculation of square root), operation ranges in valve and compressor (upper and lower limit), and evaluation of contraflow (if statement).

A model composed of multiple dynamic equipment generally has high-order DAE, whose order is 2 or more. On such complex models is difficult to perform numerical analysis such as simulation and optimization. In this study, we use the Pantelides method implemented in OpenModelica to reduce order of the model.

4.2 Formulation

We formulated the dynamic optimization as follows, where $\mathbf{u}^i \in \mathbb{R}^{n_u}$ are the manipulated variables at time phase i: The rotation speed of the compressor, the vane opening of the two-stage compressors, the opening of the high stage expansion valve, and the low stage expansion valve at the boundary of each phase. n_u is the number of operation variables ($n_u = 5$). $\mathbf{x}^{i,j} \in \mathbb{R}^{n_x}$ indicate the state variables: The outlet temperature of the chilled water (T_{weo}) or the cooling water (T_{wco}), the density of the liquid phase or the gas phase of the evaporator (ρ_{e}), the condenser (ρ_{c}) or economizer (ρ_{m}). The enthalpy of the liquid phase or the gas phase of the evaporator (E_{e}) , the condenser (E_{c}) or economizer (E_{m}) and pressure of the middle stage of the suction are also state variables. n_x is the number of state variables ($n_x = 9$). $\mathbf{z}^{i,j} \in \mathbb{R}^{n_z}$ are the algebraic variables: The pressure (P_{c}, P_{e}, P_{m}), temperature (T_{c}, T_{e}, T_{m}), liquid levels, and the mass flow rates of the condenser, evaporator and economizer. The flow rates of gas in the two-stage centrifugal compressor (G_{cp1}, G_{cp2}) and the flow rate of liquid in the valves (high-stage expansion valve (G_{h}), low-stage expansion valve (G_{l}), and hot gas bypass valve (G_{m})), the flow rates of and the hot gas bypass valve (G_{by}) are also algebraic variables. n_z is the number of algebraic variables ($n_z = 18$). $\mathbf{r}^0$ and $\mathbf{r}^n$ indicate the constraints in the initial condition and the termination condition. **F** is the functions related to differential equations, **G** is the functions related to algebraic equations, **H** is the inequality constraint. n is the number of divisions in the time direction about manipulated variables, and m is the number of state variables and algebraic variables. $i = 0$ means initial phase and $i = n$ indicates terminal phase. In this report with direct collocation method, the number of optimization variables are 137,508 (u:2500, x:45008, z:90000), equality constraints are 135,508, and inequality constraints are 2,500.

$$\begin{aligned}
\underset{\mathbf{U},\mathbf{X},\mathbf{Z}}{\text{minimize}} \quad & J(\mathbf{X},\mathbf{U}) \\
\text{subject to} \quad & \mathbf{r}^{0,0}(\mathbf{x}^{0,0},\mathbf{u}^{0},\mathbf{z}^{0,0}) = \mathbf{0} \\
& \mathbf{r}^{n,m}(\mathbf{x}^{n,m},\mathbf{u}^{n},\mathbf{z}^{n,m}) = \mathbf{0} \\
& \mathbf{x}^{i,j+1} = \mathbf{F}(\mathbf{x}^{i,j},\mathbf{u}^{i},\mathbf{z}^{i,j}) \cdot \Delta t \\
& \mathbf{z}^{i,j} = \mathbf{G}(\mathbf{x}^{i,j},\mathbf{u}^{i}) \\
& \mathbf{H}(\mathbf{x}^{i,j},\mathbf{u}^{i},\mathbf{z}^{i,j}) \geq \mathbf{0} \\
& \mathbf{x}^{i,j}_{min} \leq \mathbf{x}^{i,j} \leq \mathbf{x}^{i,j}_{max} \qquad (25) \\
& \mathbf{u}^{i}_{min} \leq \mathbf{u}^{i} \leq \mathbf{u}^{i}_{max} \\
& \mathbf{U} = [\mathbf{u}^{0^T},\cdots,\mathbf{u}^{n^T}]^T \\
& \mathbf{X} = [\mathbf{x}^{0,0^T},\cdots,\mathbf{x}^{n,m^T}]^T \\
& \mathbf{Z} = [\mathbf{z}^{0,0^T},\cdots,\mathbf{z}^{n,m^T}]^T \\
& \forall i = 0,\cdots,n \\
& \forall j = 0,\cdots,m
\end{aligned}$$

The objective function J consists of state variables and manipulated variables: The integration of the temperature difference between chilled water temperature (state variable) and the target temperature of 7°C in this centrifugal chiller. The refrigerant circulation flow rate, liquid levels are also considered. In addition, the penalty for the control variables affects the objective function.

4.3 Optimized profile

We demonstrate the profile optimization of startup of centrifugal chiller by using OpenModelica and CasADi OpenModelica can export a Modelica model as XML format. CasADi is an optimization modeling tool, which has Python, C++ and Octave/MATLAB interfaces.

Profile optimization was performed as follows: (1) Reduce the order of centrifugal chiller model by OpenModelica and export as XML file. (2) Convert the XML file into a Python script executable by CasADi. (3) Perform profile optimization with CasADi and NLP solver IPOPT.

The optimization algorithm is a quasi-Newton method implemented in IPOPT. CasADi automatically generate the derivatives, which is required for the quasi-Newton method. The Bayesian optimization used in the previous section is not suitable for the profile optimization because the calculation cost increases in the large optimization problem.

The chilled water temperatures of optimization results are shown in Figure 7. The black line is inlet temperature, which is boundary condition. The 3 orange lines are outlet temperature. The solid and dotted orange lines are the simulation results where the conventional and optimized

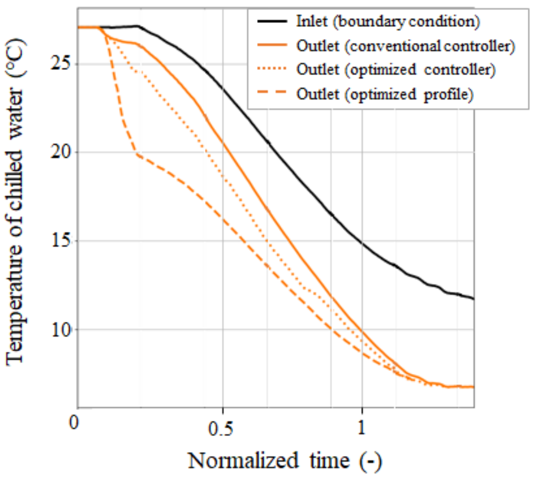


Figure 7. Temperature of chilled water by profile optimization

parameters of controller is used respectively. The dotted line, the optimized profile, shows the fastest decrease of outlet temperature, i.e., the fastest startup of the centrifugal chiller.

5 Conclusion

This paper presents a design support environment for thermodynamic systems. We conduct the coupled simulation on MATLAB/Simulink by using the chiller model in Modelica language and the control logic, and see that the parameters of the model and the control logic could be tuned. In addition, the startup profile optimization was carried out by the chiller model, which is simplified by smooth polynomial approximation and order reduction of DAE. Future work will focus on applying this design support environment to other cold products and chemical plants.

References

Joel Andersson, Johan Åkesson, and Moritz Diehl. Casadi: A symbolic package for automatic differentiation and optimal control. In *Lecture Notes in Computational Science and Engineering*, volume 87, 2012.

Satyam Bendapudi and James E. Braun. A review of literature on dynamic models of vapor compression equipment. Report, ASHRAE Research project 1043-RP, 2002.

Satyam Bendapudi, James E. Braun, and Eckhard A. Groll. A dynamic model of a vapor compression liquid chiller. In *International Refrigeration and Air Conditioning Conference. Paper 568*, 2002.

Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323. PMLR, 2011.

Frank Nielsen and Ke Sun. Guaranteed bounds on information-theoretic measures of univariate mixtures using piecewise log-sum-exp inequalities. *Entropy*, 18(12), 2016.

Hirotaka Okazaki, Hitoi Ono, and Noritaka Yanai. Optimization of start-up operation for centrifugal chiller. In *18th IFAC Workshop "Control Applications of Optimization"*, 2022.

Alejandro A. Sabbagh and Jorge M. Gómez. Optimal control of single stage libr/water absorption chiller. *International Journal of Refrigeration*, 92:1–9, 2018.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

Alachew Shitahun, Vitalij Ruge, Mahder Gebremedhin, Bernhard Bachmann, Lars Eriksson, Joel Andersson, Moritz Diehl, and Peter Fritzson. Model-based dynamic optimization with openmodelica and casadi. In *7th IFAC Symposium on Advances in Automotive Control*, pages 446–451, 2013.

Andreas Wächter and Lorenz Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106: 25–57, 2006.

Study on BEV concept design based on data driven approach

Juhyeong Park[1] Jinkyu Moon[2] Junbeom Lee[3] Daeoh Kang[1]

[1]Institute of Vehicle Engineering, Korea, jhp@ivh.co.kr
[2]Institute of Vehicle Engineering, Korea, jkmoon @ivh.co.kr
[2]Institute of Vehicle Engineering, Korea, jblee@ivh.co.kr
[3]Institute of Vehicle Engineering, Korea, bigfive@ivh.co.kr

Abstract

This paper researched the Battery EV concept design based on the data driven model. To determine the performance of BEV in the concept stage, a database was established through market research, and a data driven model was created to derive the target performance and specifications based on the database. To verify the results of the data driven model, the BEV model was generated, and the derived specifications were set. After that, the target performance was confirmed through simulation and detailed specifications were derived.

Keywords : BEV, e-Powertrain

1. Introduction

In the EV development process, there are difficulties in setting and designing system-level performance targets due to the multi-physics characteristics of EVs from a system-level perspective. In addition, the absence of a method for defining module/part target performance to meet system performance and requirements from a module/part level point of view is pointed out as a limitation of the prior art. Against this background, the need to establish a module/parts development strategy and concept considering the xEV system performance was required. In this paper, research was conducted with the goal of developing an xEV system based on system engineering technology. As a research method, based on the results of literature/market research, development concept establishment, model-based architecture, and concept specification design are carried out in the following order. In addition, as the performance of the developed vehicle, the vehicle performance that can be realized with the current e-Powertrain technology was set as a target, and an architecture using only mass-produced parts was implemented.

2. BEV Market Research

In this paper, a BEV database was established through market research. The surveyed vehicles were 37 sedans and 114 SUVs, and the survey items were battery capacity, motor output and torque, mileage, GVW, and wheelbase.

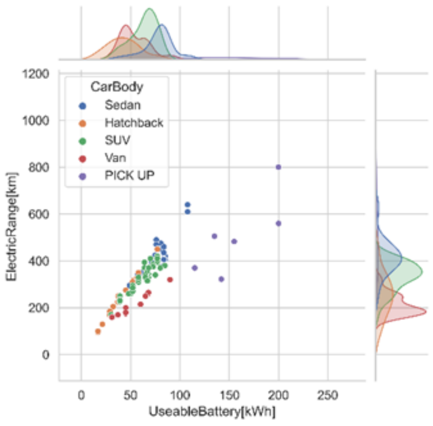


Figure 1. Batter Capacity vs Mileage

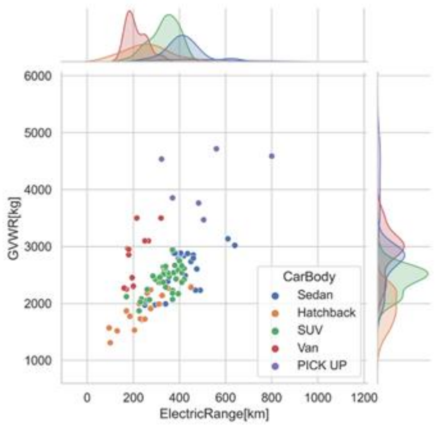


Figure 2. Mileage vs Vehicle Mass

As a result of database analysis, recently mass-produced BEVs tend to be larger and higher in performance. Also, a high correlation between GVW, wheelbase, and battery capacity was confirmed.

3. Data Driven Modeling

To derive the target performance and specifications for the concept stage BEV design, a multi-linear regression model was constructed with the mileage, wheelbase, and vehicle weight as independent variables, battery capacity, motor torque, and motor power as dependent variables. Cross validation technique was used to improve accuracy.

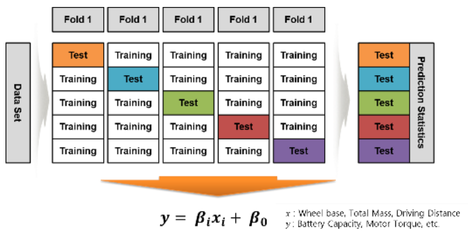

Data Set
Fold 1
Fold 1
Fold 1
Fold 1
Fold 1
Test
Training
Prediction Statistics
$y = \beta_i x_i + \beta_0$
x : Wheel base, Total Mass, Driving Distance
y : Battery Capacity, Motor Torque, etc.

Figure 3. Cross Validation

Through the above process, the following multi-linear regression equations can be found. $Y_{predict}$ is the motor power, motor torque and battery capacity, and obtained the shown in Table 1.

$$Y_{predict} = \alpha_1 * wheelbase + \alpha_2 * Range + \alpha_3 * GVW + \beta$$

$Y_{predict}$	α_1	α_2	α_3	β	R^2
P_{motor}	-9.9e-4	3.6e-1	1.2-e+1	-204.9	0.72
T_{motor}	8.7e-3	1.2e-1	4.0e-2	-99.3	0.97
$C_{battery}$	5.9e-1	1.3e-1	1.5e-1	-1607	0.85

Due to the distribution of data in the database, different results are obtained each time the process is performed. To obtain more reliable results, the above process was repeated 10,000 times to collect data, and the average value was set as the final target performance and specifications by confirming the distribution of the collected data. Figure 4 summarizes the entire process for deriving target performance and specifications.

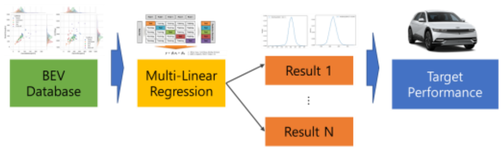


Figure 4. Process to derive target performance and specifications based on database

The design target vehicle is an SUV, with a target mileage of 400 km, wheelbase of 2.9 m, and weight of 3,000 kg. To derive the specifications suitable for the set vehicle, the above-mentioned process was performed, and the data distributions a shown in Figure 5 was obtained.

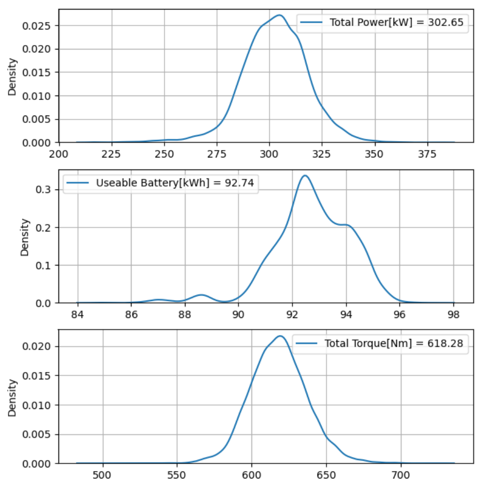


Figure 5. Process progress result data distribution

From the above results, it was found that the motor power required 300kW, a torque of 620Nm, and a battery capacity of 92kWh for the vehicle we intended.

4. Design Validation

To verify the results of obtained from Chapter 3, BEV was modeled and simulated.

4.1. Detail Modeling

BEV modeling was carried out to confirm the target specifications and performance of the BEV concept vehicle derived earlier. The BEV model consists of the Chassis model and the E-Powertrain architecture. The E-Powertrain architecture consists of a battery model and a motor model, and the motor model transmits driving force to the chassis model through Driveline.
Chassis model consists of body model, suspension model, and wheel model. For the body, 3,000 kg of GVW was applied as a lumped mass model. For the front/rear suspension, a simple model consisting of force elements such as spring and damper and suspension mass was used, and the the Pacejka model was used for the tire model.

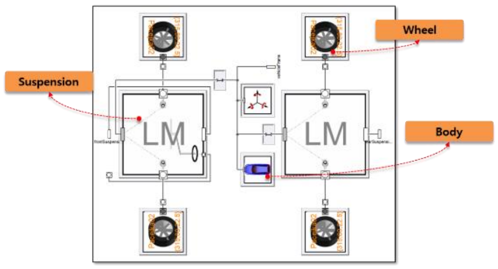


Figure 6. Chassis Model

The e-Powertrain architecture consists of a battery model and a motor model, and three architectures are configured for each drive type. The AWD type of the skateboard platform type using two motors and the FWD/RWD type composed of one motor and driveline were modeled.

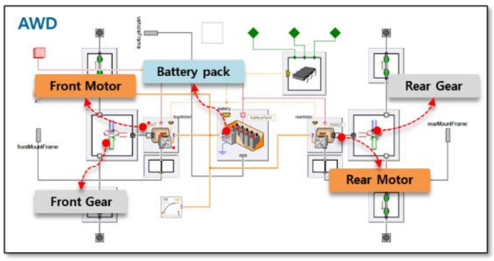


Figure 7. e-Powertrain Model(AWD)

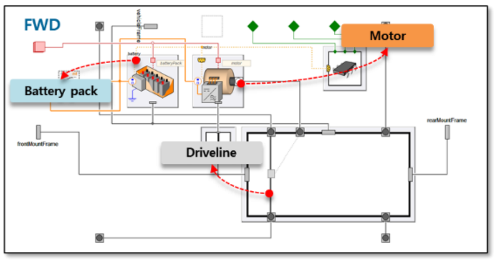
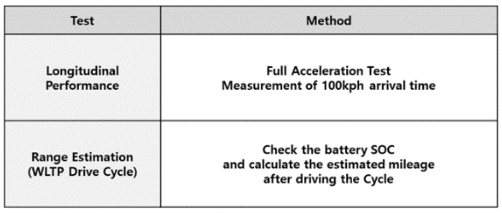


Figure 8. e-Powertrain Model(FWD)

4.2. BEV Simulation

A total of two verification evaluation scenarios for the development concept model were selected. Acceleration performance was confirmed by checking the time to reach the driving speed of 100 kph through the acceleration test, and the total mileage was predicted through the drive cycle driving test, and the final architecture was selected through the e-Powertrain architecture evaluation.



Test	Method
Longitudinal Performance	Full Acceleration Test Measurement of 100kph arrival time
Range Estimation (WLTP Drive Cycle)	Check the battery SOC and calculate the estimated mileage after driving the Cycle

Figure 9. Estimation Scenario

4.2.1. Acceleration Test

The acceleration test was conducted by setting the same motor output and torque for each drive type. As a result of the test, the AWD drive type, which showed the shortest time to reach 100 kph, showed the best acceleration performance. However, the difference in the maximum speed according to the driving method was not significant, and as the maximum speed indicates a high maximum speed of 200 kph or more, there is a need to set and limit an appropriate maximum speed.

4.2.2. Driving Test

For the development concept vehicle model, the total drivable distance was confirmed through the method of driving the WLTP Drive Cycle. The total mileage can be calculated using the formula below.

$$\text{E.M} = \left(\frac{b}{a}\right) * 1\ \mathit{Cycle\ Distance}$$

E.M = Estimated Mileage
a = SOC Reduction
b = Battery Operation Range SOC

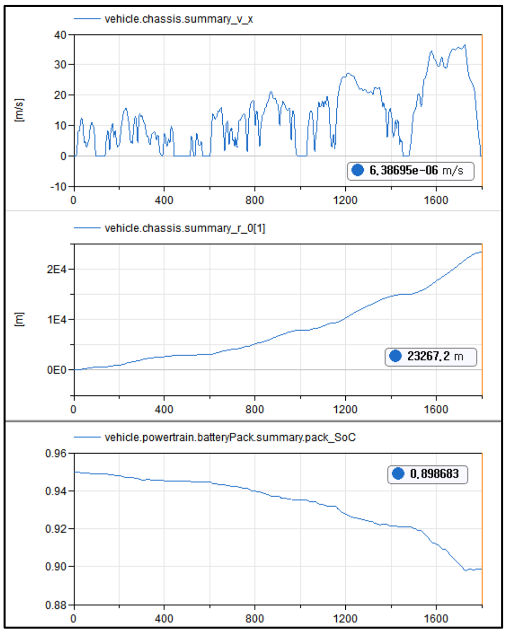
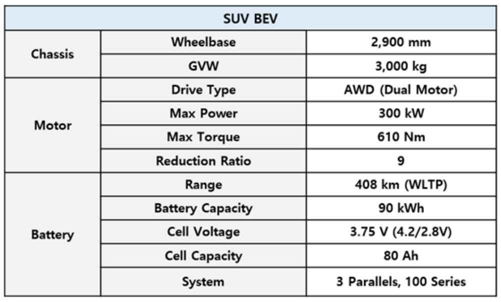


Figure 10. Driving Test Result (WLTP)

5. Conclusion

Various databases were created through market research related to EV development. The target performance for the development concept vehicle model was derived using the created database, and the vehicle performance was predicted through the EV System Architecture model configuration and virtual test environment. As a representative result of the driving performance results, a total mileage of 408 km could be predicted.
Based on the performance prediction results, an optimal EV System Architecture model was established, and through this, the main specifications of the vehicle to be developed were derived.



SUV BEV		
Chassis	Wheelbase	2,900 mm
	GVW	3,000 kg
Motor	Drive Type	AWD (Dual Motor)
	Max Power	300 kW
	Max Torque	610 Nm
	Reduction Ratio	9
Battery	Range	408 km (WLTP)
	Battery Capacity	90 kWh
	Cell Voltage	3.75 V (4.2/2.8V)
	Cell Capacity	80 Ah
	System	3 Parallels, 100 Series

Figure 11. Development Vehicle Specification

References

Micha Lesemann et al., Integrated Architectures for Third Generation Electric Vehicles – First Results of the ELVA Project, Brussels, Belgium, October 26-28, 2011.

Societal scenarios and available technolo-gies for electric vehicle architectures in 2020, Project deliverable, ELVA project consortium, Aachen, 2011

Griffin, J., Batteh, J., and Andreasson, J.: Modeling Vehicle Drivability with Modelica and the Vehicle Dynamics Library 9th International Modelica Conference, Munich, Germany, Sep. 3-5, 2012.

Andreasson, J. and Gäfvert, M.: The VehicleDynamics Library — Overview and Applications 5th International Modelica Conference, Vienna, Austria, pp. 43-51, Sep. 4-5, 2006.

Hirano, Y., Inoue, S., Ota, J.: Model-based Development of Future Small EVs using Modelica 10th International Modelica Conference, Lund, Sweden, Mar. 10-12, 2014.

Modelling and Optimal Design of Gas Engine CCHP System in Hospital

Qian Zheng Zhang Xuemei[*] Li Zhiang

School of Mechanical Engineering, Tongji University, China

Abstract

The combined heating and power (CHP) system and the combined cooling, heating and power (CCHP) system have attracted great attention during the last decade. However, many CHP systems don't perform well in the actual operation. This paper presents a complete hierarchical modeling tool of the gas engine CHP/CCHP system which is built on the software Dymola. Meanwhile, a gas-engine CHP hybrid energy system serving a hospital in Shanghai is studied as a case. To validate the accuracy of newly-built models, the operating data of the CHP part of the system in 2017 is compared with the simulation results, it is found that the minimum error is 2.1%, and the maximum error is 7.0%. Then, the original gas engine CHP hybrid energy system is reconstructed to a gas engine CCHP system. To analyze the feasibility of the optimal design, the conventional energy supply system which was used in the hospital before 2013, the original gas engine CHP hybrid energy system and the optimized gas engine CCHP system are modeled and simulated. From the simulation results, it is found that the primary energy ratio is increased from 72.55% to 133.37%, the payback period of investment is decreased from nearly 11.8 years to 3.9 years, and the CO_2 emissions reduction rate is increased from 4.83% to 93.72%. Therefore, the optimization scheme is feasible.

Keywords: Gas engine, Combined cooling heating and power, System model, Dynamic simulation, Optimal design

1 Introduction

Nowadays, the energy crisis and the environmental impact of fossil fuels have been increasingly serious globally (Moussawi et al, 2016; Wei et al, 2016; Ameri et al, 2016; Yousefiet et al, 2017; Zheng et al, 2018; Jiang et al, 2018). Efficient technology for energy conservation is urgently needed to ensure energy supplies and reduce environmental emissions (Jiang et al, 2018). Combined heating and power (CHP) system and combined cooling, heating and power (CCHP) system have received widespread attention due to the advantage of substantial reliability, energy-saving, environmental friendliness and cost-saving (Wei et al, 2016; Zheng et al, 2018; Jiang et al, 2018; Das et al, 2018; Afzali et al, 2018; Zhang et al, 2018). CCHP system is defined as an effective energy system that generates cooling, heating and power simultaneously, while CHP system removes cooling from the list , mainly through the cascade utilization of energy (Wei et al, 2016; Kavvadias et al, 2018). In recent years, CHP and CCHP have been introduced to small-medium scale places, such as hospitals, hotels, domestic houses and office buildings (Wei et al, 2016; Zhang et al, 2018; Santo, 2012; Kavvadias et al, 2010).

CHP has developed rapidly since the first CHP energy supply technology was introduced in the 1990s. However, it is worth noting that many CHP systems suffer from the uncertainty of the actual economic results. Because these CHP systems only provide a fraction of energy for buildings, such as hot water and a part of power, while excess energy is still fed by the conventional energy system. A possible solution is optimizing the CHP hybrid energy system to a CCHP system.

During the last decade, many researchers used system modeling and simulation to optimize the performance and the design procedure of CHP and CCHP systems. Wei et al. proposed a multi-objective optimization model to provide a guiding principle for CCHP system optimization (Wei et al, 2016). They adopted software MATLAB and TRNSYS to identify a series of compromised optimal operation strategies with different operational parameters using Non-dominated Sorting Genetic Algorithm-II (NSGA- II). Ameri et al. described a mixed integer linear programming (MILP) model to determine the optimal capacity and operation of seven CCHP systems in eastern Tehran (Iran) (Ameri et al, 2016). Results showed that compared with generating heat by boilers and purchasing electricity from the local grid, the optimal CCHP system was able to save costs and reduce CO_2 emissions. A mixed integer non-linear programming (MINLP) model was developed by Zheng et al. to achieve multi-objective optimization of a smart micro-grid using the modeling environment GAMS (Zheng et al, 2018). Results described by four scenarios showed that net present value, primary energy saving and CO_2 emissions were reduced significantly by installing roof-top PV, ground source heat pump, natural gas-based CCHP and storage systems. Espirito Santo proposed a computational hourly profile simulation methodology (Santo, 2012) and performed an integrated thermal system simulation (Santo, 2014) using software COGMCI. An effective method for the design optimization of CCHP coupled multi-energy system was developed by Lu et al. (Lu et al, 2018). They established a correlation model for configuration and operation optimization based on a bi-level model construction method, proposed a solution method, and developed an optimization tool using MATLAB. Mago et al. optimized CCHP systems for an office building in Columbus (USA) following the thermal load (FEL), the electrical load （FTL） and a hybrid electrical-thermal load (HETS) strategies (Mago et al, 2009). Results showed that HETS was better than FEL and FTL. Pagliarini et al. studied the feasibility of integrating an existing natural gas-fired-boiler central plant in Parma (Italy) into the CCHP system (Pagliarini et al, 2012). The space heating and cooling loads were calculated by TRNSYS. The national policies supporting CHP were found to have a strong influence on the results. TRNSYS was also used by Rosato et al. to simulate the performance of a micro-CHP system and a conventional system (Rosato et al, 2013). Results showed that the micro-CHP system could significantly reduce primary energy consumption, carbon dioxide emissions and operating costs. Hu et al. proposed a stochastic multi-objective optimization model to optimize the CCHP operation strategy for different climate conditions based on operational cost, primary energy consumption (PEC) and carbon dioxide emissions (CDE) and added a higher reliability level of the probability constraint to it (Hu et al, 2014). Moreover, an incentive model was developed to support the multi-objective decision analysis. The feasibility of integrating air-conditioning system and heat storage tank into the CCHP system was studied by Li et al. (Li et al, 2014). They formulated the optimal problem as a nonlinear programming problem using genetic algorithm (GA). Furthermore, a sensitivity analysis was conducted to explore the impact of natural gas prices on system economics. Jannelli et al. developed a 0-1-dimensional model of a small-size CCHP based on the integration of a 20 kW diesel engine and a double-effect water-LiBr absorption chiller on platform AVLBOOST (Jannelli et al, 2014). The manufacturer's sample data was used to validate the performance parameters of the gas engine under different operating conditions and the average error was found to be less than 5%. Particle Swarm Optimization (PSO) was used by Hajabdollahi et al. to optimize the gas engine CCHP system for the purpose of comparing a new operational strategy named variable electric cooling ratio (VER) with constant electric cooling ratio (CER) for different climates (Hajabdollahi et al, 2015). Piacentino et al. used a decision tool to optimize the layout, design and strategy of a CCHP plant simultaneously in the hotel sector (Piacentino et al, 2015). In addition, two sensitivity analyses were performed on tax exemption for the fuel consumed in “high-efficiency cogeneration mode” and on the dynamic behavior of the system. Moussawi et al. conducted a simulation study using TRNSYS software for diesel engine-driven CCHP systems used to provide electricity, space heating, space cooling and sanitary hot water (SHW) to a typical residential family house in Beirut (Moussawi et al, 2015). Wang et al. simulated and evaluated four different gas-engine CCHP systems applied for a remote island using TRNSYS, and the results showed that the one adopting the double-effect absorption chiller and the gas-fired boiler was the best option (Wang et al, 2016). Based on the environment, economy and energy criteria simultaneously, Zeng et al. optimized the CCHP–GSHP coupling system model by GA and demonstrated the practicality of the optimization model by case analysis (Zeng et al, 2016). Mat Isa et al. developed a CHP system consisting of grid-connected photovoltaic (PV), fuel cell and battery, and performed the techno-economic analysis of the proposed system using hybrid optimization model for electric renewable simulation (HOMER) software in order to assess the feasibility of applying the system for a hospital building in Malaysia (Isa et al, 2016). Calise et al. developed a detailed dynamic simulation model of the CCHP system using TRNSYS and evaluated three different system operating strategies, namely: Thermal Load Tracking mode (TLT), Maximum Power Thermal Load Tracking mode (MPTLT) and Electricity Load Tracking mode (ELT) (Calise et al, 2017). Yang et al. proposed a gas turbine-driven CCHP system combining solar thermal energy and compressed air energy storage (S-CAES) and developed system off-design models. In comparison with the corresponding optimized CCHP system without S-CAES, the system with S-CAES performed better (Yang et al, 2017).

However, few models are built with hierarchical architecture, so models and sub-models lack reusability and are difficult to debug separately. In this study, the gas engine CHP/CCHP system is modeled and dynamically simulated in Dymola software (“DYMOLA Systems Engineering, Multi-Engineering Modeling and Simulation based on Modelica and FMI.”) using Modelica language. Based on the results of dynamic simulation, the feasibility of optimization from the gas engine CHP hybrid energy system to the gas engine CCHP system will be discussed. The main contributions in this research are summarized as follows: (1) Simulation models of important equipment and the whole system of CHP hybrid energy system and CCHP system are built and validated. (2) The original gas engine CHP hybrid energy system is optimized to a gas engine CCHP system for a case study. (3) The feasibility of optimization is analyzed by comparing the performance of two systems.

This paper is organized as follows: Section 2 describes the case for optimization, presents the modeling approach and validation results for simulation models of the important equipment in CHP and CCHP systems. Section 3 gives the load calculation results and system optimal design for the case building. Section 4 presents the performance evaluation method used in this study. Section 5 analyzes and compares the performance of the original gas engine CHP hybrid energy system and the gas engine CCHP system. The conclusions are drawn in the last section.

2 Methodology

2.1 Case study

The case building, Ruijin Hospital North, is located in Jiading District, Shanghai, China, with a floor area of 72000 m^2. A conventional energy supply system was applied to this hospital before 2013. By replacing the hot water boilers with a 334kW gas engine and two heat exchangers, the conventional system was optimized to a gas engine CHP system in October 2013. The energy supply layouts of the conventional system and the gas engine CHP system are shown in Figure 1 and Figure 2, respectively, and the main parameters of system components are shown in Table 1.

Ruijin Hospital North has stable electrical load and hot-water heating load, and the former is much higher than the latter. Due to the current policy that power generated by self-provided units can only be grid-connected but not exported to the grid, the gas engine CHP system applied to this hospital operates in the "power determined by heat" energy supply mode, i.e. only the demand for sanitary hot water is considered to be necessarily met by the CHP unit, based on which the corresponding generated power output is connected to the hospital power supply system.
To ensure the stability of power generation and waste heat output, and to prolong the service life of the equipment, there are only two states of the gas engine generator set in actual operation: rated state (100% load) and shutdown (0% load). In consideration of the resulting mismatch between the stable sanitary hot water input on the supply side and the ever-changing hot-water heating load on the demand side, a heat storage tank was installed to control the start and stop of the gas engine according to its water level. The gas engine will be started when the water level drops to 0.2 meters and shut down when the water level rises to 4.8 meters.

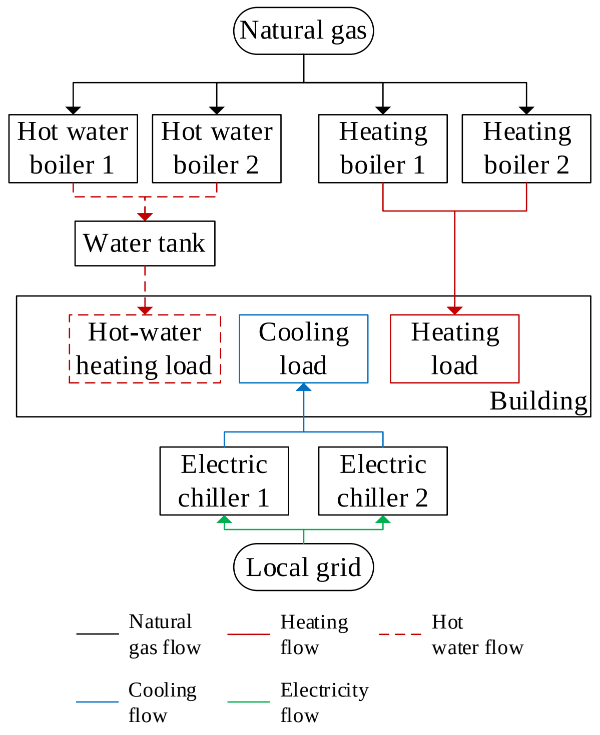


Figure 1. Energy supply layout of conventional system

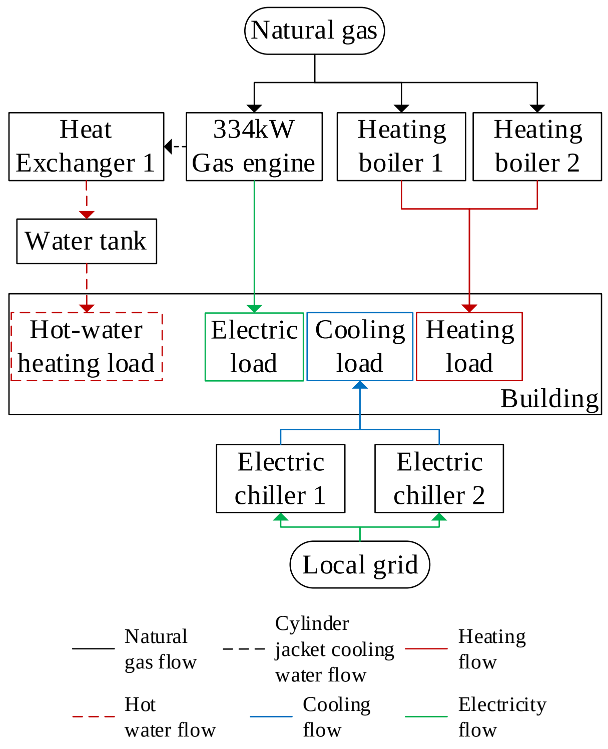


Figure 2. Energy supply layout of gas engine CHP system

Table 1 Main parameters of system components

Affiliated System	*Component*	*Parameter*	*Value*
Both	Heating boiler 1&2	Rated heat supply/kW	2000
		Efficiency/%	90
	Hot water tank	Volume/m^3	50
	Electric chiller 1&2	Rated cooling capacity/kW	4515
		Rated COP	6.22
Convent-ional system	Hot water boiler 1&2	Rated heat supply/kW	200
		Efficiency/%	90
Gas engine CHP system	334kW gas engine	Model	Schmitt 334
		Rated electrical power generation/kW	334
		Rated jacket water waste heat/kW	485
	Heat exchanger 1	Flow form	Counter flow
		Quantity	2
		Nominal heat transfer coefficient/ W/(m^2*K)	5.476
		Heat transfer area/m^2	15

2.2 Simulation model

Simulation models of individual devices and overall systems are established in Dymola software by employing Modelica language. Dymola software supports hierarchical model composition, libraries of truly reusable components, connectors and composite acausal connections. The modeling method of this study is to build the complete hierarchical simulation model of systems based on the connection of equipment models (gas engine generator set model, LiBr absorption chiller set model, plate type heat exchanger model, electric chiller set model, hot water tank model and gas boiler model) and system control models used to control on-off conditions, operating hours and operating strategies. Since Modelica Standard Library includes most of the required equipment models, it is only necessary to build the additional gas engine model and LiBr absorption chiller model.

2.2.1 Gas engine

This study mainly focuses on the system's overall performance. Since the gas engine is just one component of the whole system, its performance parameters, such as electrical power, total heat recovery, exhaust gas heat, coolant heat, mixture heat, fuel input, natural gas consumption, electrical efficiency, thermal efficiency and total efficiency, rather than internal structural parameters, should be mainly concerned. The performance documentation provided by manufacturers contains specific performance parameters for different gas engine models at 50% load, 75% load and 100% load, based on which the performance parameters of gas engines operating in the range of 50% load to 100% load can be calculated by interpolation method. Therefore, the gas engine performance parameter model consists of two parts: a performance parameter sheet model used to store the datasheet of different samples and an interpolation model used to read the datasheet and output the corresponding performance parameters according to the input parameter (electricity demand). Up to now, this performance parameter sheet model library has stored datasheets of more than 20 samples of different brands such as Mannheim, Caterpillar and Schmitt.

2.2.2 LiBr absorption chiller

There are many types of LiBr absorption chiller, among which single-effect hot water type and double-effect flue gas type are used in the present study. The internal structure of the whole LiBr absorption chiller model is shown in Figure 3. The LiBr absorption chiller model also consists of a performance parameter sheet model and an interpolation model.

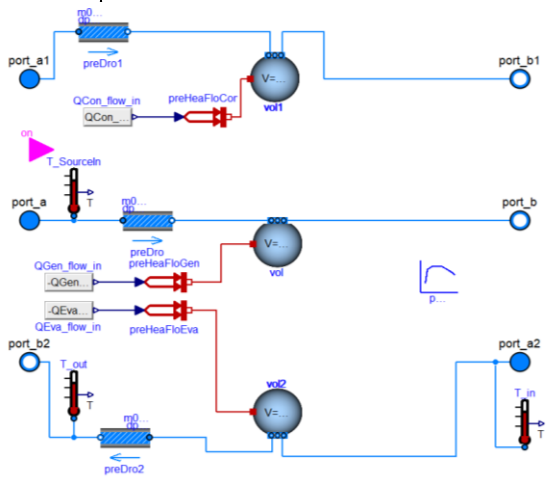


Figure 3. Internal structure of the LiBr absorption chiller model

1. Performance parameter sheet model

The performance parameter sheet model stores the datasheet of main rated parameters of LiBr absorption chiller samples, such as cooling capacity, heat consumption, heat source temperature, inlet and outlet temperature of cooling water, inlet and outlet temperature of chilled water, COP, etc. Unlike the stable performance parameters of gas engines at rated operating conditions, the most important performance parameter of LiBr absorption chiller, coefficient of performance (COP), is influenced by several factors and is therefore given as the COP curve by manufacturers. For ease of reading, the COP curve is converted into 4 data tables: chilled water temperature correction table, cooling water temperature correction table, heat source temperature correction table and cooling capacity correction table. Each table contains two columns, where the first column is the independent parameter (chilled water temperature, cooling water temperature, heat source temperature, cooling capacity) and the second column is the corresponding COP.

2. Interpolation model

The input parameter of the interpolation model is the cooling load, and the output parameters are the actual cooling capacity, the heat exchange capacity of the generator and the heat removed by the cooling water. The specific calculation process is as follows.

The actual COP of LiBr absorption chiller can be calculated by the following formula, which is provided in the LiBr absorption chiller technical manual.

$$COP = \frac{COP_{T,chilled} + COP_{T,cooling} + COP_{T,source} + COP_{cap}}{4} \quad (1)$$

Since the cooling capacity datasheet needs to be read on the basis of partial load rate, it's necessary to convert the cooling load into partial load rate.

The cooling load (C_{load}) can be calculated as follows:

$$C_{load} = \dot{m}c(t_{in} - t_{set}) \quad (2)$$

where, $\dot{m}$ is the flow rate of chilling water, kg/s; c is the specific heat of chilled water, kJ/(kg·°C); t_{in} and t_{set} respectively represent the inlet temperature and set temperature of chilling water, °C.

The maximum load rate and the minimum temperature limit of the heat source must be taken into consideration when calculating the partial load rate (PLR) of LiBr absorption chiller. PLR is between the minimum and maximum load rate, and is 0 when the heat source temperature is lower than its minimum.

Then, the actual heat consumption ($Q_{generator}$) of LiBr absorption chiller can be calculated as follows:

$$Q_{generator} = C_{operation}/COP \quad (3)$$

where, $C_{operation}$ represents the actual cooling capacity read by the performance parameter sheet model, kW.

The heat removed by the cooling water includes the heat released by the absorber and the condenser, and the heat absorbed by the LiBr absorption chiller includes the heat absorbed by the evaporator and the generator. Neglecting the heat dissipation of pump, the energy balance equation can be expressed as:

$$Q_{condenser} = C_{operation} + Q_{generator} \quad (4)$$

where, $Q_{condenser}$ represents the heat removed by the cooling water, kW.

2.3 Validation

2.3.1 Actual operation data

Monthly operation data of the case CHP unit in 2017 is investigated, as shown in Table 2, to validate the accuracy of models.

Table 2. Operation data of the case CHP unit in 2017

Month	*Boot hour (h)*	*Waste heat recovery (kWh)*	*Power generation (kWh)*	*Natural gas consumption (Nm^3)*	*Thermo-electric ratio*
1	402	159,760	134,165	35,303	1.19
2	357	123,790	119,135	35,937	1.04
3	336	132,550	112,212	33,010	1.18
4	249	102,780	83,103	25,112	1.24
5	181	72,170	60,318	17,456	1.20
6	163	52,000	54,572	15,717	1.02
7	96	44,940	32,129	9,319	1.40
8	121	57,460	40,397	11,716	1.42
9	126	52,960	42,159	12,253	1.26
10	156	59,440	52,028	15,071	1.14
11	174	65,470	57,975	16,851	1.13
12	221	88,340	73,692	20,925	1.20
Total	2,580	941,660	861,885	248,670	1.09

2.3.2 Model validation

As shown in Figure 4 and Figure 5, discrepancies between the simulation results and operation data are less than 10% for both system power generation and natural gas consumption, and thus, the model accuracy is validated. Therefore, the subsequent results of system operation characteristics and optimization are expected to be of sufficient accuracy. An error or robust design analysis would be required to predict expected accuracy of additional model predictions.

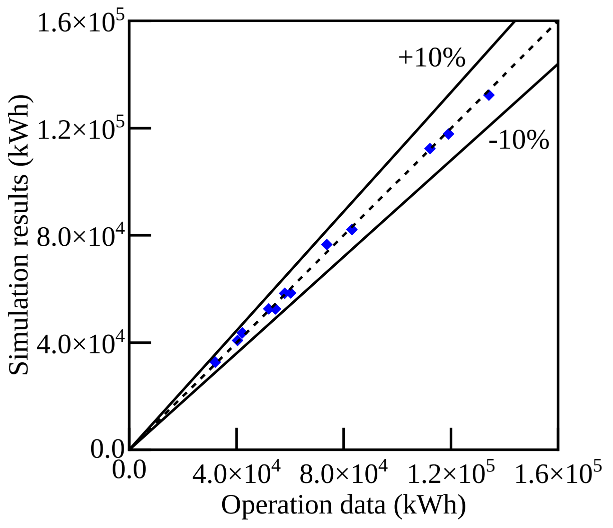


Figure 4. Model validation of system power generation

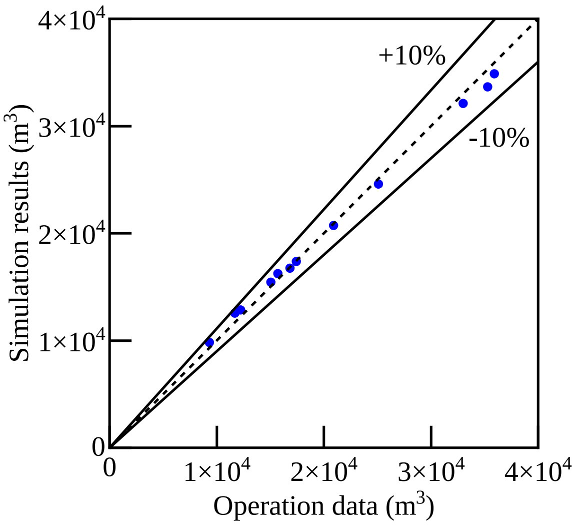


Figure 5. Model validation of natural gas consumption

3 System optimization

3.1 Load calculation of the case building

To optimize the design of system and further simulate it, it's necessary to calculate the hourly cooling load, heating load and hot water load of the case building.

The specific flow direction and flow distribution of the hot water are of no importance in modeling, hence only the hot water load is considered. During the simulation, the monthly hot water load is distributed to the hourly load according to the load factor method (Shan, 1989). The hourly hot water load is shown in Figure 6.

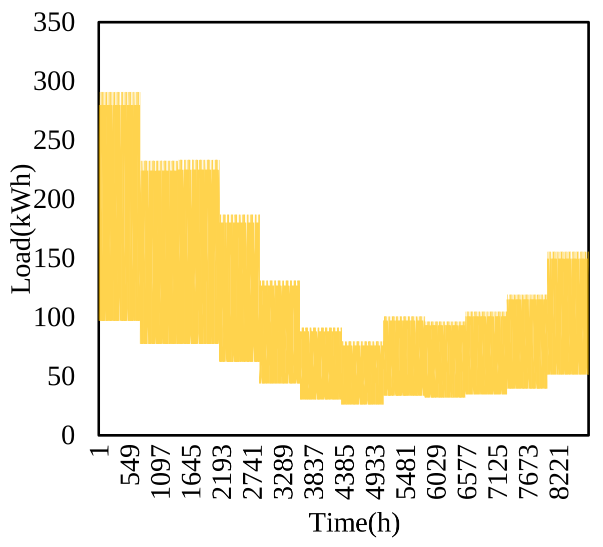


Figure 6. Hourly hot water load of the case building

The cooling load and heating load are calculated by the software HDY-SMAD (" HDY-SMAD, HVAC Load Calculation and Analysis Software."). The hourly cooling and heating load are shown in Figure 7. According to the calculation results, the maximum hourly cooling load and heating load are 9102.63 kW and 3723.11 kW, respectively.

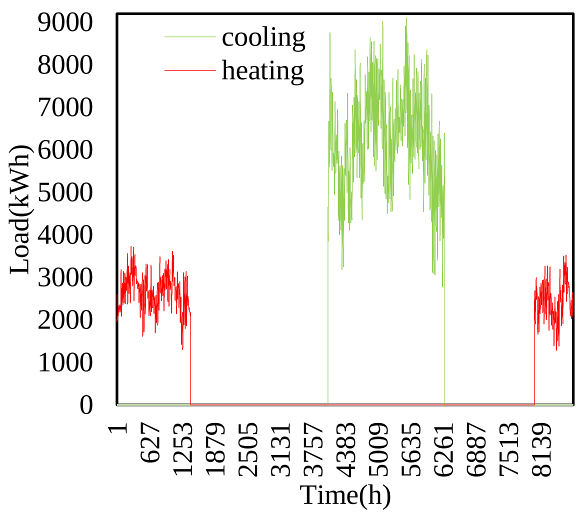


Figure 7. Hourly cooling and heating load of the building

3.2 Optimal design

To improve the comprehensive performance and economy of CHP unit, the original gas engine CHP system is optimized to a gas engine CCHP system. The CCHP part of optimized CCHP system will provide all the heating load and hot water load of the hospital, along with most of the cooling load and electric load. The remaining part of the cooling load is provided by the electric chiller and excess electric load is fed with power purchased from the local grid.

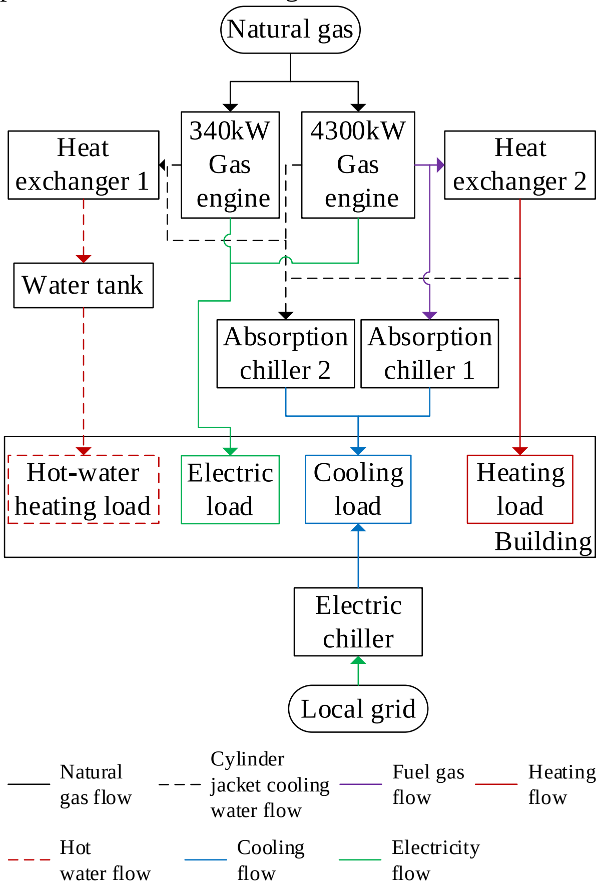


Figure 8. Energy supply layout of gas engine CCHP system

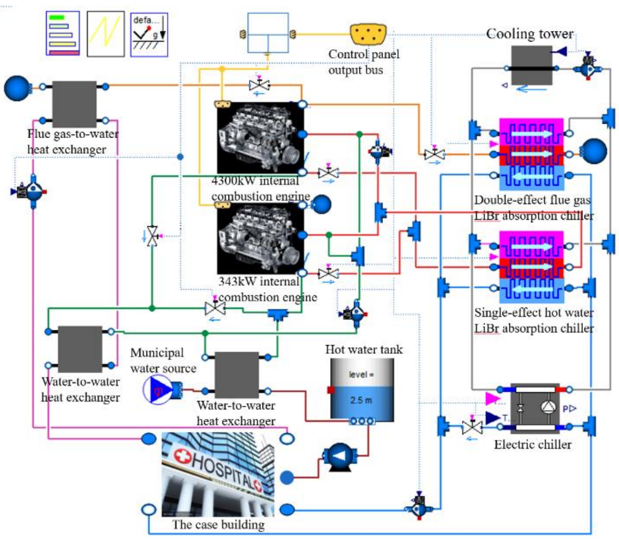

Control panel output bus
Cooling tower
Flue gas-to-water heat exchanger
4300kW internal combustion engine
343kW internal combustion engine
Double-effect flue gas LiBr absorption chiller
Single-effect hot water LiBr absorption chiller
Municipal water source
Water-to-water heat exchanger
Water-to-water heat exchanger
Hot water tank
Electric chiller
HOSPITAL
The case building

Figure 9. Model schematic of gas engine CCHP system

The energy supply layout and model schematic of optimized CCHP system is shown in Figure 8 and Figure 9, respectively. A 4300kW gas engine and four heat exchangers are used to replace heating water boilers, while two absorption chillers are used to replace an electric chiller. Main parameters of new system components are shown in the Table 3. The system process is as follows. The 334kW gas engine supplies all the sanitary hot water. In summer, two absorption chillers are preferred to meet the cooling load of the hospital, and the shortage is met by the electric refrigeration unit. The flue gas generated by the operation of the 4300kW gas engine is passed into the double-effect flue gas type absorption chiller for cooling, and the jacket water (rated at 98 °C) generated by it and the remaining jacket water of the 334kW gas engine are accessed to the single-effect hot water type absorption chiller. In winter, the flue gas of the 4300kW gas engine is introduced into the flue gas-hot water plate heat exchanger for heating. The jacket water (rated at 98 °C) generated by it and the remaining jacket water of the 334kW gas engine are also used for heating.

The system follows the hybrid operating mode. The 334kW gas engine is operated at full load in winter and summer. It is started and stopped according to the water level in the transition season. The 4300kW gas engine is only operated in winter and summer and it is controlled by return water temperature.

Table 3. Main parameters of gas engine and heat exchanger

Component	*Parameter*	*Value*
4300kW gas engine	Model	Mannheim 4300
	Rated electrical power generation	4300 kW
	Flue gas waste heat	2304 kW
	Rated jacket water waste heat	1379 kW
Heat exchanger 2	Flow form	Counter flow
	Quantity	4
	Nominal heat transfer coefficient	2.728 $W/(m^2 \cdot K)$
	Heat transfer area	25 m^2
Absorption chiller 1	Model	BROAD BE300
	Type	Double-effect smoke type
	Rated cooling capacity	3489 kW
Absorption chiller 2	Model	BROAD BDH200
	Type	Single-effect hot water type
	Rated cooling capacity	2046 kW

4 Economy, energy efficiency, environment (3E)

In order to evaluate the comprehensive performance of gas engine system and gas engine CCHP system, the 3E analysis (Gao et al, 2022) is conducted in this section.

4.1 Energy efficiency analysis

Primary energy ratio (η_t) is adopted to evaluate the energy efficiency of system. Primary energy ratio is defined as the ratio between the output energy and the input primary energy of the system, and can be expressed as follows:

$$\eta_t = \frac{E_p + E_s + Q_h + Q_c}{V_g Q_L + E_s/\eta_w} \tag{5}$$

where, E_p and E_s respectively represent the system electrical power generation and electrical power use from local grid, kJ. Q_h and Q_c respectively represent the system heat supply and cooling capacity, kJ. V_g represents the system fuel consumption, Nm^3. Q_L represents the low calorific value of fuel, kJ. η_w represents the product of electrical power generation efficiency and transmission efficiency of local grid, %.

4.2 Economy analysis

Economic analysis is based on two indicators: annual operating cost and payback period of investment.

System annual operating cost (C_n) is the sum of annual fuel cost (C_g), annual electrical power cost (C_e) and system maintenance cost (C_w), and can be expressed as follows:

$$C_n = C_g + C_w + C_e \tag{6}$$

For reconstruction, system payback period of investment (n) is the recovery period of system's incremental investment (L), and can be expressed as follows:

$$n = \frac{L}{C_{n,con} - C_{n,re}} \tag{7}$$

where, $C_{n,con}$ and $C_{n,re}$ respectively represent the annual operating cost of conventional system and reconstructed system.

4.3 Environment analysis

CO_2 emissions per unit capacity (P_C') and CO_2 emissions reduction rate (R_{CO_2}) are important indicators for environment analysis, which can be calculated as follows:

$$P_C' = \frac{P_C}{E_p + Q_h + Q_c} \tag{8}$$

$$P_C = E_s \cdot EF_e + V_g \cdot EF_g \tag{9}$$

$$R_{CO_2} = \frac{P_{C_{con}}' - P_{C_{re}}'}{P_{C_{con}}'} \tag{10}$$

Where, P_C represents the system CO_2 emissions, Nm^3/s. EF_e and EF_g respectively represent the CO_2 emission coefficient of local grid and natural gas. $P_{C_{con}}'$ and $P_{C_{re}}'$ respectively represent the CO_2 emissions per unit capacity of conventional system and reconstructed system, Nm^3/kJ. (In engineering calculation and trade settlement, China stipulates that the volume at a pressure of 101.325 KPa and a temperature of 293.15 K is defined as a standard cubic meter, expressed in Nm^3. For the convenience of analysis and calculation, this unit is used uniformly in the following.)

5 Results and discussion

5.1 Simulation results

The simulation results of system electricity generation, natural gas consumption and electricity consumption are obtained by dynamic simulation with Dymola software, as shown in Table 4.

Table 4. Simulation results

	Conventional energy supply system	*Original gas engine CHP hybrid energy system*	*Optimized gas engine CCHP system*
Cooling load (kWh)	14,001,281.7		
Heating load (kWh)	5,566,748.1		
Hot water load (kWh)	818,322.0		
Power generation (kWh)	0	865,709.6	15,358,726.5
Natural gas consumption (Nm^3)	169.0	184.4	846.1
Power consumption (kWh)	17,958,094.1	17,092,384.5	1,113,431.5

5.2 Analysis results of 3E

5.2.1 Energy efficiency analysis

According to the test of the hospital's energy station, the average low calorific value of natural gas is 34.308 MJ/Nm^3. Moreover, product of electrical power generation efficiency and transmission efficiency of local grid is assumed to be 40% in this study. Then, the primary energy ratios can be obtained by Eq. (5), and the results are shown in Figure 10.

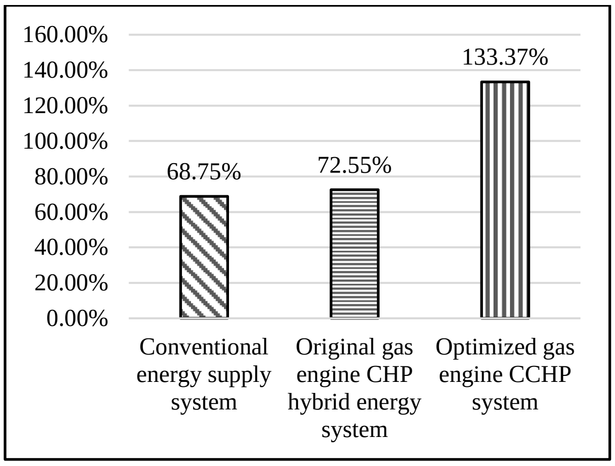


Figure 10. Primary energy ratio

5.2.2 Economy analysis

In Shanghai, the price of natural gas is 2.45 yuan/m³ for distributed energy systems and is 3.82 yuan/m³ for gas boilers. Besides, the electricity price is 0.641 yuan/kWh for non-resident users. For conventional energy supply system, original gas engine CHP hybrid energy system and optimized gas engine CCHP system, the average system maintenance cost is 0.024 yuan/kWh, 0.12 yuan/kWh, 0.15yuan/kWh, respectively.

Through investigation and calculation, the incremental investment of original gas engine CHP hybrid energy system and optimized gas engine CCHP system is 4 million yuan and 27.58 million yuan respectively. Based on Eq. (6)-(7), the payback period of investment of three systems are shown in Table 5.

Table 5. Payback period of investment calculation

	Conventional energy supply system	*Original gas engine CHP hybrid energy system*	*Optimized gas engine CCHP system*
Incremental investment (yuan)	-	4,000,000	27,580,000
Annual net operating cost (yuan)	16,567,599	16,227,434	9,581,721
Payback period of investment (year)	-	11.8	3.9

5.2.3 Environment analysis

According to the survey, the CO_2 emission coefficient of natural gas source in Shanghai is 1.04Nm^3/Nm^3 natural gas and the national average CO_2 emission coefficient of electrical power generation is 0.412 Nm^3/(kWh). Based on these, Eq. (8)-(10) are used to calculate total CO2 emission, CO2 emissions per unit capacity and CO2 emissions reduction rate of these systems. Calculation results are shown in Figure 11 and Figure 12.

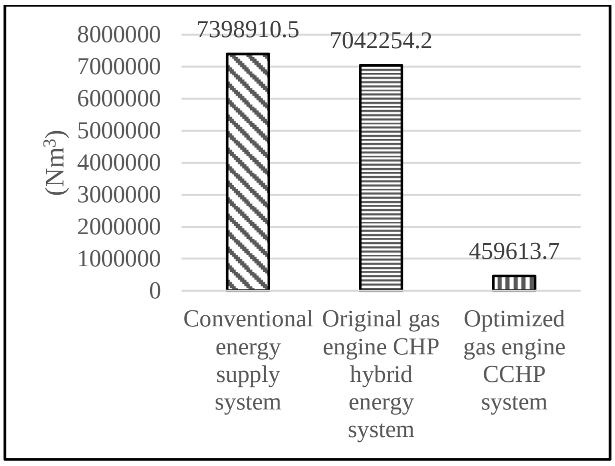


Figure 11. Total CO_2 emission

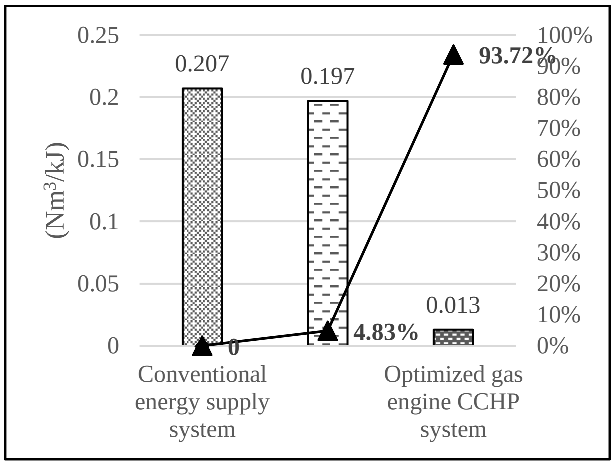


Figure 12. CO_2 emissions per unit capacity and CO_2 emissions reduction rate

5.3 Feasibility analysis

Through the above evaluation calculation, it is easy to find that, compared with the original gas engine CHP hybrid energy system, the optimized gas engine CCHP system is improved a lot. Firstly, the optimized system solves the economic problem efficiently. The payback period of investment is decreased from 11.8 years to 3.9 years. Meanwhile, the energetic and environmental performance of the system are also optimized. Primary energy ratio is increased by 83.83% and CO_2 emissions reduction rate is increased by 93.40%. Therefore, it can be concluded that reconstructing a CHP hybrid energy system to a CCHP system has a high feasibility.

6 Conclusions

In this paper, a complete hierarchical modeling method of the gas engine CHP/CCHP system was presented. And the models of gas engine generator set, heat exchanger and LiBr absorption chiller were then built based on theoretical analysis, mathematical equation and the performance curve of equipment. Then we validated the accuracy of the presented model by taking the gas engine CHP system of a hospital in Shanghai as an example. By modeling and simulating the system, we found that the minimum error between the calculated results of the model and the operational data of this system in 2017 is 2.1% and the maximum error is 7.0%. The results meet the requirement that the simulation deviation is less than 10%, which validates the accuracy of the model.

Furthermore, we reconstructed the original gas engine CHP hybrid energy system to a gas engine CCHP system in our model. Then the conventional energy supply system used in the hospital before 2013, the original gas-engine CHP hybrid energy system and the optimized gas engine CCHP system were modeled and simulated to analyze the feasibility of this optimization. The simulation results show that the gas engine CCHP system can increase the primary energy ratio from 72.55% to 133.37%, shorten the payback period from nearly 11.8 years to 3.9 years and increase the CO_2 reduction rate from 4.83% to 93.72%, which validates the feasibility of this optimization.

References

Sayyed Faridoddin Afzali, Vladimir Mahalec. Novel performance curves to determine optimal operation of CCHP systems. *Applied Energy*, 226:1009-1036, 2018.

Mohammad Ameri, Zahed Besharati. Optimal design and operation of district heating and cooling networks with CCHP systems in a residential complex. *Energy and Buildings*, 110:135-148, 2016.

Francesco Calise, Massimo Dentice d'Accadia, Luigi Libertini, Edoardo Quiriti, Maria Vicidomini. A novel tool for thermoeconomic analysis and optimization of trigeneration systems: A case study for a hospital building in Italy. *Energy*, 126:64-87, 2017.

Barun K. Das, Yasir M. Al-Abdeli, Ganesh Kothapalli. Effect of load following strategies, hardware, and thermal load distribution on stand-alone hybrid CCHP systems. *Applied Energy*, 220:735-753, 2018.

Hassan Hajabdollahi, Abdolsaeid Ganjehkaviri, Mohammad Nazri Mohd Jaafar. Assessment of new operational strategy in optimization of CCHP plant for different climates using evolutionary algorithms. *Applied Thermal Engineering*, 75:468-480, 2015.

Hassan Hajabdollahi, Abdolsaeid Ganjehkaviri, Mohammad Nazri Mohd Jaafar. Assessment of new operational strategy in optimization of CCHP plant for different climates using evolutionary algorithms. *Applied Thermal Engineering*, 75:468-480, 2015.

Normazlina Mat Isa , Himadry Shekhar Das, Chee Wei Tan, A.H.M. Yatim, Kwan Yiew Lau. A techno-economic assessment of a combined heat and power photovoltaic/fuel cell/battery energy system in Malaysia hospital. *Energy*, 112:75-90, 2016.

E. Jannelli, M. Minutillo, R. Cozzolino, G. Falcucci. Thermodynamic performance assessment of a small size CCHP (combined cooling heating and power) system with numerical models. *Energy*, 65:240-249, 2014.

Runhua Jiang, Huibin Yin, Baiman Chen, Yongjun Xu, Minlin Yang, Xiaoxi Yang. Multi-objective assessment, optimization and application of a grid-connected combined cooling, heating and power system with compressed air energy storage and hybrid refrigeration. *Energy Conversion and Management*, 174:453-464, 2018.

K.C. Kavvadias, A.P. Tosios, Z.B. Maroulis. Design of a combined heating, cooling and power system: Sizing, operation strategy selection and parametric analysis. *Energy Conversion and Management*, 51:833-845, 2010.

Longxi Li, Hailin Mu, Weijun Gao, Miao Li. Optimization and analysis of CCHP system based on energy loads coupling of residential and office buildings. *Applied Energy*, 136:206-216, 2014.

Shilei Lu, Yuwei Li, Hongwei Xia. Study on the configuration and operation optimization of CCHP coupling multiple energy system. *Energy Conversion and Management*, 177:773-791, 2018.

P.J. Mago, L.M. Chamra. Analysis and optimization of CCHP systems based on energy, economical, and environmental considerations. *Energy and Buildings*, 41:1099-1106, 2009.

Houssein Al Moussawi, Mohammad Mahdi, Farouk Fardoun, Hasna Louahlia Gualous. Recovery Storage Tank Size: An Optimization Approach for Trigeneration Systems on Diesel Power Generators. *Energy Procedia*, 74:788-798, 2015.

Houssein Al Moussawi , Farouk Fardoun , Hasna Louahlia-Gualous. Review of tri-generation technologies: Design evaluation, optimization, decision-making, and selection approach. *Energy Conversion and Management*, 120:157-196, 2016.

G. Pagliarini, C. Corradi, S. Rainieri. Hospital CHCP system optimization assisted by TRNSYS building energy simulation tool. *Applied Thermal Engineering*, 44:150-158, 2012.

A. Piacentino, R. Gallea, F. Cardona, V. Lo Brano, G. Ciulla, P. Catrini. Optimization of trigeneration systems by Mathematical Programming: Influence of plant scheme and boundary conditions. *Energy Conversion and Management*, 104:100-114, 2015.

Antonio Rosato, Sergio Sibilio, Giovanni Ciampi. Dynamic performance assessment of a building-integrated cogeneration system for an Italian residential application. *Energy and Buildings*, 64:343-358, 2013.

D.B. Espirito Santo. Energy and exergy efficiency of a building internal combustion engine trigeneration system under two different operational strategies. *Energy and Buildings*, 53:28-38, 2012.

Denilson Boschiero do Espirito Santo. An energy and exergy analysis of a high-efficiency engine trigeneration system for a hospital: A case study methodology based on annual energy demand profiles. *Energy and Environment*, 76:185-198, 2014.

Zhe Tian, Jide Niu, Yakai Lu, Shunming He, Xue Tian. The improvement of a simulation model for a distributed CCHP system and its influence on optimal operation cost and strategy. *Applied Energy*, 165:430-444, 2016.

Lang Wang, Jianfeng Lu, Weilong Wang, Jing Ding. Energy, environmental and economic evaluation of the CCHP systems for a remote island in south of China. *Applied Energy*, 183:874-883, 2016.

Dajun Wei, Alian Chen, Bo Sun, Chenghui Zhang. Multi-objective optimal operation and energy coupling analysis of combined cooling and heating system. *Energy*, 98:296-307, 2016.

Cheng Yang, Xusheng Wang, Manman Huang, Su Ding, Xiaoqian Ma. Design and simulation of gas turbine-based CCHP combined with solar and compressed air energy storage in a hotel building. *Energy and Buildings*, 153:412-420, 2017.

Hossein Yousefi, Mohammad Hasan Ghodusinejad, Younes Noorollahi. GA/AHP-based optimal design of a hybrid CCHP system consideringeconomy, energy and emission. *Energy and Buildings*, 138:309-317, 2017.

Rong Zeng, Hongqiang Li, Runhua Jiang, Lifang Liu, Guoqiang Zhang. A novel multi-objective optimization method for CCHP–GSHP coupling systems. *Energy and Buildings*, 112:149-158, 2016.

Jiaxuan Zhang, Sheng Cao, Lijun Yu, Yaodong Zhou. Comparison of combined cooling, heating and power (CCHP) systems with different cooling modes based on energetic, environmental and economic criteria. *Energy Conversion and Management*, 160:60-73, 2018.

Xuyue Zheng, Guoce Wua, Yuwei Qiu, Xiangyan Zhan, Nilay Shah, Ning Li, Yingru Zhao. A MINLP multi-objective optimization model for operational planning of a case study CCHP system in urban China. *Applied Energy*, 210:1126–1140, 2018.

“DYMOLA Systems Engineering, Multi-Engineering Modeling and Simulation based on Modelica and FMI.” Dymola - Dassault Systèmes®. October 23, 2022. https://www.3ds.com/products-services/catia/products/dymola.

Jiping Shan. Practical calculation method of air conditioning load. Beijing: China Architecture & Building Press, 1-6, 1989.

“HDY-SMAD, HVAC Load Calculation and Analysis Software.” HUADIAN YUAN. October 23, 2022. https://http://www.hdy.com.cn/smad.html.

Lei Gao, Yunho Hwang, Tao Cao. An overview of optimization technologies applied in combined cooling, heating and power systems. *Renewable and Sustainable Energy Reviews*, 114: 109334, 2019.

eMobility & Energy Management

Development for Sustainable Mobility on 3DEXPERIENCE Platform

Kharisma NITIPUTRA[2] Stéphane RIOU[1] Guillaume VIRY[2]

[1]Dassault Systèmes SE, France
[2]Dassault Systèmes K.K, Japan

stephane.riou@3ds.com
kharisma.nitiputra@3ds.com
guillaume.viry@3ds.com

Abstract

Due to the rapid shift towards vehicles partially or fully powered by electricity (eMobility), Dassault Systèmes has developed an Industry Solution Experience to address new challenges of electrification and energy management.

Our goal is to demonstrate how to apply the precepts of Model-Based Systems Engineering (MBSE) and to converge towards a virtual twin in a zero-prototype approach.

Keywords: eMobility, virtual twin, Modelica, MBSE, battery, EV, simulation

1 Introduction

In the race towards sustainable mobility (reduction of CO_2 footprint, design for recycling …), the massive shift towards eMobility has already begun. Acknowledging the fact that personal vehicles are one of the major sources of CO_2 emissions, and consumers are encouraged to adopt electric vehicles due to their significantly lower CO_2 output, more and more countries are coming up with a roadmap to ban ICE vehicles.

Along with that push from policy and regulations, the OEMs are now providing a more mature offering and are ready to ramp up for mass production.

By 2026, we expect EVs to oversell ICE on the European market, and by 2030 we are looking at around 85 million EVs for Europe only.

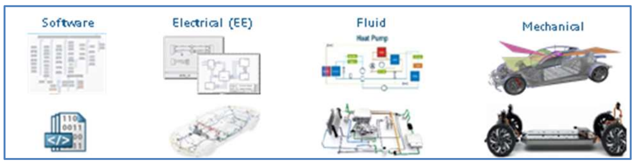
2 Towards Sustainable Mobility

We cannot complete such an ambitious paradigm change by only focusing on vehicles. A holistic approach combining Energy, Vehicle and Mobility Service, is required to reach our goals of sustainable mobility.

- Energy: consider the complete energy stream, including production, transportation, storage and distribution
- Vehicle: increase vehicle energy efficiency (less loss); and improve vehicle performance (range, charging time, …)
- Mobility Service: take into account all services contributing to the reduction of CO_2 in usage: smart charging and Vehicle to Grid, optimization of route planning taken into account location, speed and availability to charging points

ENERGY & TERRITORIES
VEHICLE
MOBILITY SERVICE

It is no more enough to optimize battery, propulsion and thermal systems separately.
It is now admitted that a significant gain will come from the optimization of the whole energy chain integration and control command.

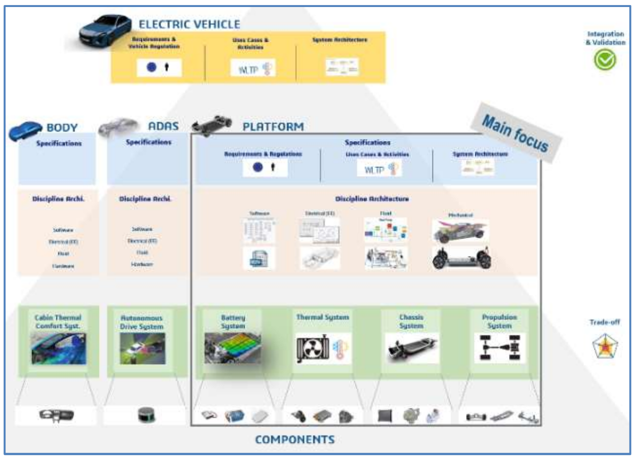


To do so, we must combine all relevant disciplines (fluidics, mechanical, electrical & electronics and software) from the initial stages to talk a common language across departments with traceable continuity from architecture level to hardware and software designs.

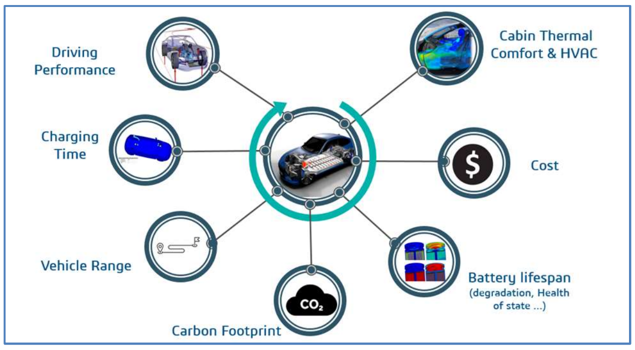
3 Efficient Multi-Energy Platform

The 3DEXPERIENCE Platform can help to tackle those challenges. Not only are we providing some of the best-in-class tools for 1D and 3D simulation, they are also integrated in one single platform, along with complete PLM, powerful data analytics, all this to help you make the right decisions, backed by science and data evidence.

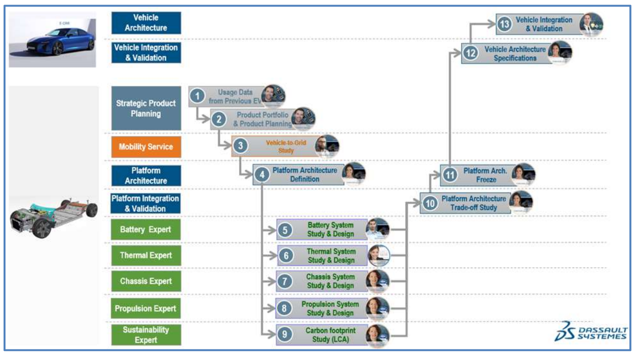
A more concrete manifestation of this thought process is a Dassault Systèmes Industry Solution Experience (ISE) developed specifically for this use case, named Efficient Multi-Energy Platform. By leveraging all Dassault Systèmes has to offer in its solution portfolio, we wish to offer and end-to-end integrated solution that brings value to our customers in all relevant stages of the development.



Efficient Multi-Energy Platform has of course a very strong modeling and simulation focus, since eventually complex 1D and 3D will be key elements within the design process (from battery system to powertrain, including chassis, thermal systems, …).
One of key industrial challenges being to find the best compromise for all performance metrics.


Driving Performance
Cabin Thermal Comfort & HVAC
Charging Time
Cost
Vehicle Range
Battery lifespan (degradation, Health of state ...)
Carbon Footprint
CO₂

Nevertheless, this should not lessen the importance of the other key functionalities covered by this Industry Solution Experience:


Vehicle Architecture
Vehicle Integration & Validation
Strategic Product Planning
Mobility Service
Platform Architecture
Platform Integration & Validation
Battery Expert
Thermal Expert
Chassis Expert
Propulsion Expert
Sustainability Expert
1 Usage Data from Previous EV
2 Product Portfolio & Product Planning
3 Vehicle-to-Grid Study
4 Platform Architecture Definition
5 Battery System Study & Design
6 Thermal System Study & Design
7 Chassis System Study & Design
8 Propulsion System Study & Design
9 Carbon footprint Study (LCA)
10 Platform Architecture Trade-off Study
11 Platform Arch. Freeze
12 Vehicle Architecture Specifications
13 Vehicle Integration & Validation
DASSAULT SYSTEMES

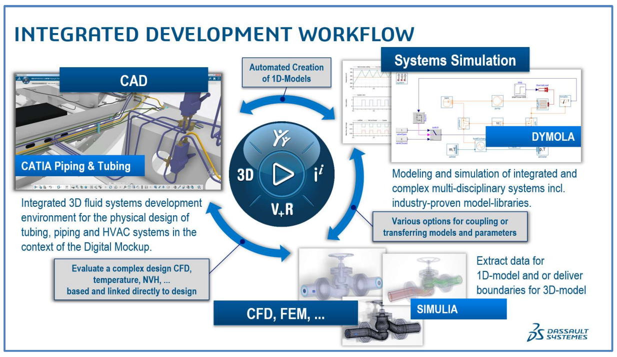
- Data intelligence: analysis of EV operational data from an Energy Management standpoint, simulation result analytics (decision making)
- Environmental footprint and Life Cycle Assessment

Smart electricity grid management with Vehicle-to-Grid is also an important lever toward decarbonization.

4 Simulation framework

As written above, several key parts of this ISE involve complex simulations. Some of them like chassis system study for instance, rely heavily on 3D structural analysis, in order to investigate the impact on complex geometries of nonlinear plastic deformations. However, most of the simulation activity is instead leveraging the Modelica language. Because those tasks require investigating at the same time multiple physical domains with tight interactions (electrical, mechanical, thermal, control…) to provide relevant results, Modelica, as an equation-based and multi-discipline oriented language, is a natural fit for those.

Dassault Systèmes is already known for developing Dymola, one of the main Modelica tools on the market. In addition to this stand-alone version, Dassault Systèmes also decided to enrich its 3DEXPERIENCE Platform by integrating the Dymola kernel and positioning Modelica as a simulation backbone, in order to enable simulation in as many areas as possible. Not only can we take advantage of the Dymola technology for its own merits, but we can also benefit from the new synergies made possible by the 3DEXPERIENCE Platform.


INTEGRATED DEVELOPMENT WORKFLOW
CAD
Automated Creation of 1D-Models
Systems Simulation
DYMOLA
CATIA Piping & Tubing
3D
V+R
Integrated 3D fluid systems development environment for the physical design of tubing, piping and HVAC systems in the context of the Digital Mockup.
Modeling and simulation of integrated and complex multi-disciplinary systems incl. industry-proven model-libraries.
Various options for coupling or transferring models and parameters
Evaluate a complex design CFD, temperature, NVH, ... based and linked directly to design
Extract data for 1D-model and or deliver boundaries for 3D-model
CFD, FEM, ...
SIMULIA
DASSAULT SYSTEMES

In the move towards the Digital Twin, many models and diagrams that were just static until now came to life thanks to the Modelica language. We make use of another open standard as well, the Functional Mock-up Interface (FMI), when the need to integrate external models or complex 3D simulations in our workflow occurs.

5 Take-away

With this solution, we aim at accelerating the energy transition by designing, simulating and optimizing the Digital Twin of the eMobility, from battery to platform and vehicle. The ultimate target being to reach Zero Prototype, through virtual vehicle development.

This is possible by:

- combining software, electrical and mechanical architectures in a unique collaborative environment with digital traceability and continuity
- managing assets in a consistent way for all vehicle engineering and study in evolution and configuration
- assessing product performance combining ADAS and energy management
- managing legacy data and solution in an Opened platform
- handling vehicle complexity by embracing Systems Engineering concepts and methods, thanks to an integrated business and science platform

Physical modeling of daily electric appliances for education by Modelica

Yutaka Hirano[1]   Koichi Ohtomi[2]

[1]Woven Planet Holdings, Inc., Japan, `yutaka.hirano@woven-planet.global`
[2] Ohtomi Design Lab., Japan, `ohtomi@1dcae.jp`

Abstract

Modelica is very useful to make physical models in various engineering fields such as mechanical, electrical, thermal, fluid systems, etc. This capability of Modelica is also useful to educate students and engineers about many physical areas using simulation. The authors are posting serialized articles in a technical magazine about physical modeling of daily electric appliances by Modelica to educate readers about both physics and Modelica language in Japan. This paper introduces some examples of physical modeling of various appliances such as electric minicar, dryer and speaker by Modelica.

Keywords: *physical modeling, control, Modelica*

1 Introduction

Modelica is an equation based, object-oriented language for efficient modeling of complex, multi domain cyber physical systems described by ordinary differential, difference and algebraic equations. This feature of Modelica language is very useful for not only various industrial applications but also for education about physics and simulation for students and engineers. The authors are posting serialized articles in a technical magazine about physical modeling of daily electric appliances by Modelica to educate readers about both physics and Modelica language in Japan. This paper introduces some examples of physical modeling of various appliances.

In section 2, modeling and simulation of a 4x4 electric minicar is described. In section 3, dryer is modeled and simulated. In section 4, modeling about a speaker is introduced.

2 Modeling of 4x4 electric minicar

4x4 electric minicars are a very popular toy in Japan. It consists of body, batteries, motor, gears, drive shafts and tires as shown in Figure 1. The physical model of the electric minicar is assumed as shown in Figure 2.

The modeling was done to solve the following questions.

1) What is the maximum speed of the car?
2) How long can this car keep running?

Thus, the battery model should consider the effect of voltage drop by the SOC (State Of Charge) change. In the body model running resistances should be considered.

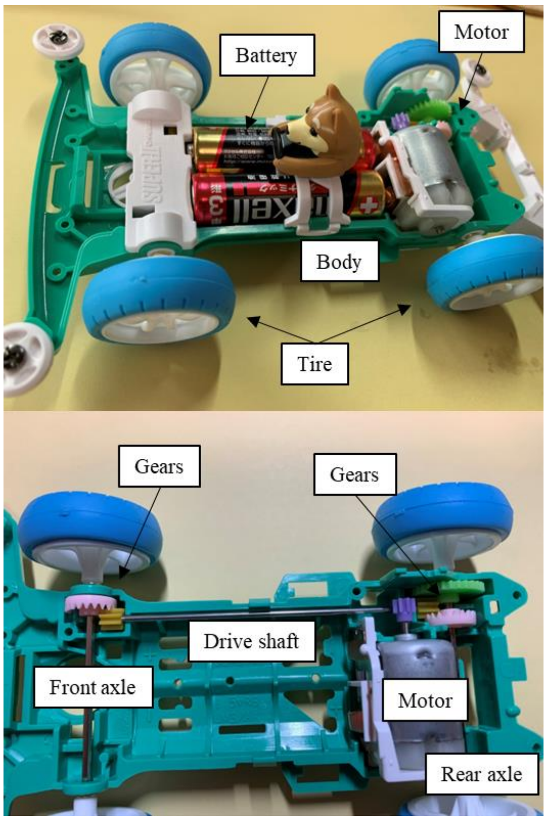


Figure 1. Structure of electric minicar

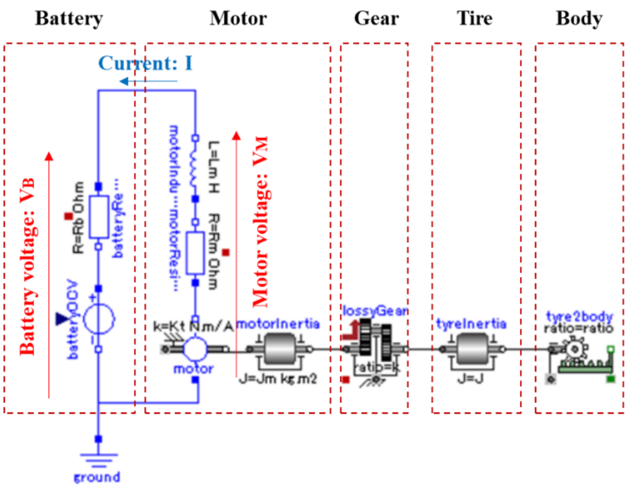


Figure 2. Physical model structure of electric minicar

2.1 Structure of the physical model

Battery model is built by OCV (Open Circuit Voltage) generator and inner resistance. OCV is a function of SOC. Motor model consists of a DC motor electric machine, a motor inertia and electric circuit including armature coil's resistance and inductance. Gear and tire model are assumed as just a simple model of lossy constant ratio reducer. As for the running resistances, aerodynamic resistance, rotating resistance and gravity resistance by road slope are considered. Consequently, the system of equations of this model become as follows.

(Battery)

Battery voltage: $V_B = V_{OCV}(SOC) - R_B I$ (1)

State Of Charge: $\mathrm{SOC} = 1 - \frac{V_{bo}\int I dt}{W_{total}}$ (2)

Open circuit voltage: $V_{OCV}(SOC) = V_{b0} \times SOC$ (3)

For the simplicity, V_{OCV} is assumed to be proportional to the *SOC*. It will be more precise if the actual table of SOC vs OCV based on measurement will be used.

(Motor)

Motor voltage: $V_M = V_{emf} + R_M I + L\frac{dI}{dt}$ (4)

Back electromotive force voltage:

$V_{emf} = K_E \omega_M$ (K_E : coefficient) (5)

Motor torque: $\tau_M = K_T I$ (K_T : coefficient) (6)

Motor inertia: $J_M \frac{d\omega_M}{dt} = \tau_M$ (7)

(J_M : Innertia moment, ω_M : Angular velocity)

(Circuit)

$V_B = V_M$ (8)

(Gear)

Gear torque: $\tau_t = \varepsilon K_g \tau_M$ (9)

(K_g : Gear ratio, ε : Gear efficency)

Gear rotation speed: $\omega_M = K_g \omega_t$ (10)

(Tire and Body)

Tire inertia torque: $J_t \frac{d\omega_t}{dt} = \tau_t - \mathrm{r} f_t$ (11)

(J_t : Tire innertia, ω_t: Tire rotation speed)

Car velocity: $V = \mathrm{r}\omega_r$ (r : Tire radius) (12)

Tire rotating resistance force:

$f_t = m\alpha + \mu mg + mg\sin\theta + \rho A C_D V^2/2$ (13)

(μ : Rotating resistance coefficient, θ : Road slope, ρ : Air dencity, A : Frontal area, C_D: Aero resistance coefficient)

Vehicle speed: $V = \frac{dx}{dt}$ (14)

Vehicle acceleration: $\alpha = \frac{dV}{dt}$ (15)

2.2 Modelica code of text-based model

By using the system of equations shown in the section 2.1, Modelica code of the text-based model becomes as below.

```
model miniCarText_SOC
   import SI = Modelica.Units.SI;
   import Modelica.Constants.g_n;
  parameter SI.Resistance Rb = 0.8
  "Battery inner resistance";
  parameter SI.Resistance Rm = 1
   "Motor inner resistance";
  parameter SI.Inductance Lm = 1e-6
   "Motor inner inductance";
  parameter SI.MomentOfInertia Jm = 1.8e-3
* 0.005 * 0.005 "Motor innertia";
  parameter SI.MomentOfInertia Jt = 5e-3 *
0.01 * 0.01 "Tire innertia";
  parameter Real Kt = 1.2e-3
   "Motor torque coefficient";
  parameter Real Ke = 1.2e-3
   "Motor rotational voltage coefficient";
  parameter Real Kg = 5 "Gear ratio";
  parameter SI.Efficiency Eg = 1
   "Gear efficiency";
  parameter SI.Mass m = 0.1
   "Vehicle mass";
  parameter SI.Radius r = 0.015
   "Tyre radius";
  parameter SI.CoefficientOfFriction myu =
0.1 "Tyre rotating friction coefficient";
  parameter Real Cd = 0.3
   "Air drag coefficiency";
  parameter SI.Area area = 0.004
   "Vehicle frontal area";
  parameter SI.Density rho = 1.205
   "Air density";
  parameter Real batteryPowerCapacity = 1
   "Battery power capacity [Wh]" ;
  parameter SI.Voltage Vb0 = 3
   "Battery initial voltage" ;
// Variables;
  SI.Current i(start = 0) "Motor current";
  SI.Voltage Vocv
   "Battery open circuit voltage";
  SI.Voltage Vb "Battery voltage";
  SI.Voltage Vm "Motor voltage";
  SI.Voltage Vemf
   "Motor rotational voltage";
  SI.Torque taum "Motor torque";
  SI.Torque taut "Tire torque";
  SI.AngularVelocity omgm
   "Motor angular velocity";
  SI.AngularVelocity omgt
   "Tire angular velocity";
  SI.Force f "Vehicle total force";
  SI.Force fa "Vehicle acceleration force";
  SI.Force fr
   "Vehicle rolling resistance force";
  SI.Force fair
   "Vehicle air resistance force";
  SI.Distance x(start = 0)
   "Vehicle running distance";
  SI.Velocity v(start = 0)
   "Vehicle velocity";
  SI.Acceleration a "Vehicle acceleration";
  Real soc "Battery SOC" ;
  Real usedWh "Used power capacity" ;
```

```
equation
  der(usedWh) = Vb0 * i  / 3600;
  soc = if (1-usedWh/batteryPowerCapacity
>=0) then (1-usedWh/batteryPowerCapacity)
else 0;
  Vocv = Vb0 * soc;
  Vb = Vocv - Rb * i;
  Vm = Vemf + Rm * i + Lm * der(i);
  Vemf = Ke * omgm;
  Vb = Vm;
  taum = Kt * i;
//Jm*der(omgm) = taum;
  taut = Eg * Kg * taum;
  Kg * omgt = omgm;
//Jt*der(omgt) = taut - r*f;
  0 = taut - r * f;
  r * omgt = v;
  f = fa + fr + fair;
  fa = m * a;
  fr = myu * m * g_n;
  fair = rho * area * Cd * v * v / 2;
  a = der(v);
  v = der(x);
end miniCarText_SOC;
```

Please note that the equations (7) and (11) were ignored and the equation

```
  0 = taut - r * f;
```

is used instead. This is because the variables `omgm` and `omgt` are dependent on the variable `v` by the constraint of constant gear ratio. Additionally, to cope with the calculation chattering when the vehicle speed crosses zero, it is effective to use the base class of friction elements prepared in the Modelica Standard Library (MSL), but in this case it is omitted.

2.3 MSL-based model

Modelica model of this minicar can be built by using MSL as shown in Figure 3. Here a new class 'calcOCV' was created to model the SOC dependent OCV calculation as shown in the equations (2) and (3). Thanks to the Modelica feature of 'StateSelect' the rigidly coupled inertias of the motor, tire and the vehicle mass can be modeled separately.

```
StateSelect stateSelect =
StateSelect.default;
```

2.4 Simulation results

To make the students understand both physics and Modelica features, both of the text code model and the MSL model were made and simulated. Figure 4 shows the results for short time range (upper figure) and for long time range (lower figure). In the upper figure, the results of vehicle speed for both the text code model (v) and the MSL model (vehicle.v) are compared. The effect of inertial elements can be seen. Also, as for the answers for the questions above, the results became as follows.

(1) The maximum speed of the car is about 6.1 m/s.
(2) The car can keep running for about 3840 sec.

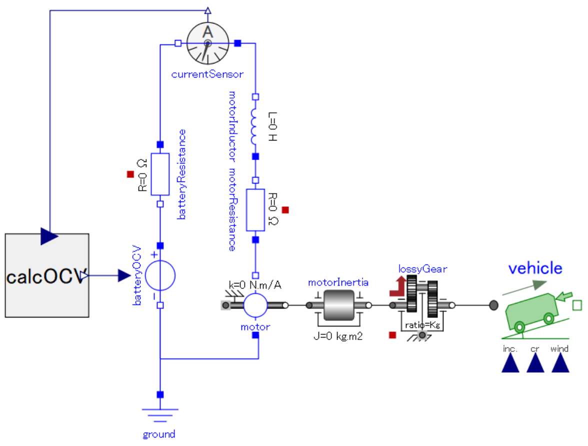


Figure 3. Minicar model using MSL

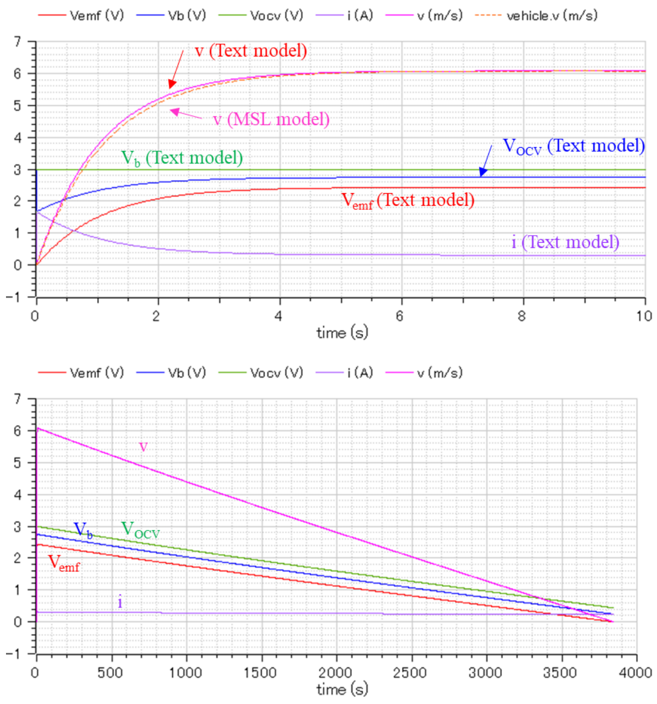


Figure 4. Simulation results of the minicar model

3 Modeling of a dryer

3.1 Structure of the dryer model

The structure of the target dryer is shown in Figure 5.

For each part of the structure, the system of the equations and the model structure were considered as below.

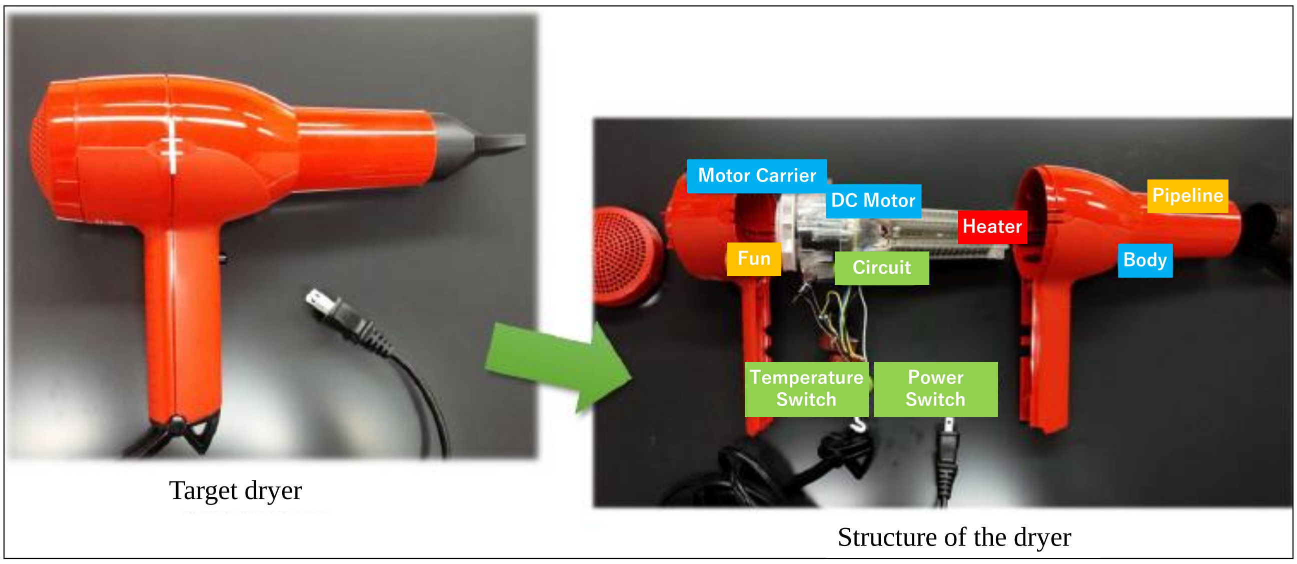


Figure 5. Structure of the dryer

About the motor and the circuit, the system of the equations is same as the equations (4) to (6). As for mechanical loss, a damping loss and friction loss are considered as below.

$$J_M \frac{d\omega_M}{dt} + c \cdot \omega_M + T_f \ = \tau_M \tag{16}$$

The physical model of the motor and circuit using MSL becomes as shown in Figure 6. The part surrounded by a red dashed line is used as the component model of the motor assembly mentioned below.

The model of the fan and heater is shown in Figure 7. The fan is modeled as an ideal air pump.

$$p = -\left(\frac{b}{a}\right) * q * n + b * n^2 \tag{17}$$

(p: pressure, q: air flow rate, n:normarized speed)

Air flow resistance in the pipe is modeled as below.

$$p = \ R * q * |q| \tag{18}$$

(R: Coefficient of air flow resistance)

The operating point of the pump is calculated from the crossing point of the equation (17) and (18) as shown in Figure 8.

The necessary heat flow to increase the temperature of the air is calculated by the following equation.

$$h = \ q * \rho * C_p * \Delta T \tag{19}$$

(h: necessary heat flow, ρ: air density, C_p: air specific heat, ΔT: target temperature increase)

By solving the simultaneous equations of the equations (17), (18) and (19), we can obtain the necessary design parameters of the dryer as n=1, h=60 and r=10 by solving the following Modelica model.

```
model DryerDesign
  Real n(start = 2);
  Real h;
  Real r;
  parameter Real Ro = 1;
  parameter Real Cp = 1;
  parameter Real q = 1;
  parameter Real a = 1.5;
  parameter Real b = 30;
  parameter Real Dt = 60;
  parameter Real p = 10;
equation
  h = q*Ro*Cp*Dt;
  p = -(b/a)*q*n + b*n^2;
  p = r*q*abs(q);
end DryerDesign;
```

The electric circuit of the power supply using the full-wave rectifier circuit can be modeled as shown in Figure 9. Here switches are ignored.

Finally the whole model of the dryer becomes as shown in Figure 10 by combining the component models of the each assembly parts (shown as red dashed rectangular in Figure 6, Figure 7 and Figure 9).

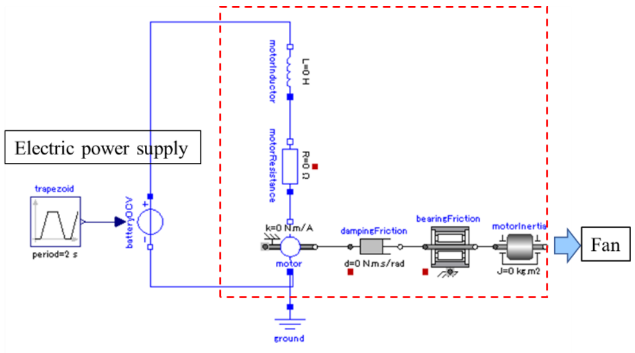


Figure 6. Model of the motor and circuit

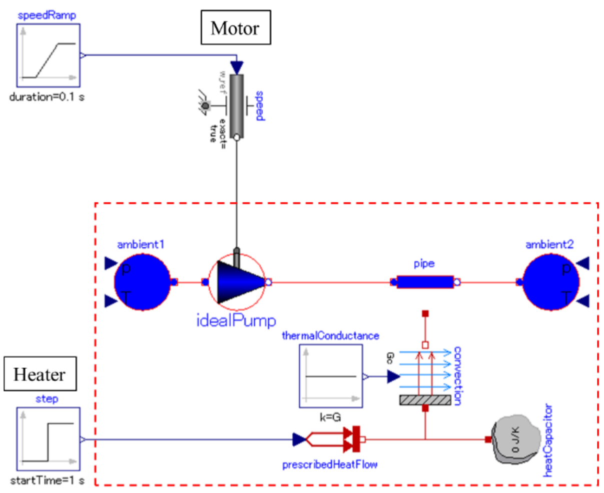


Figure 7. Model of the fan and heater

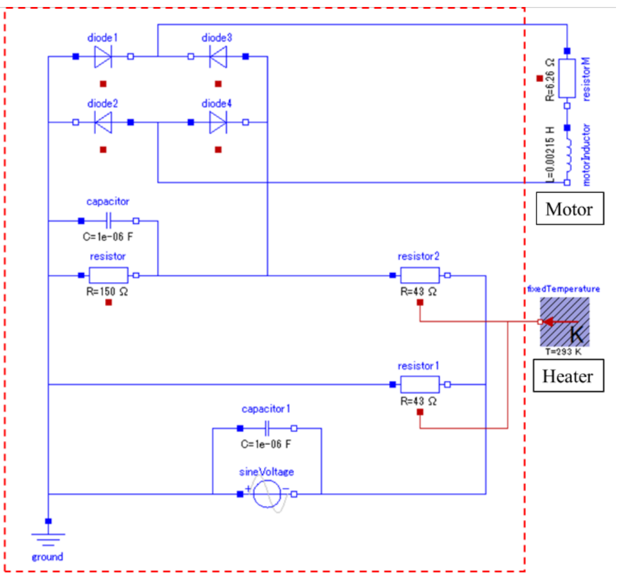
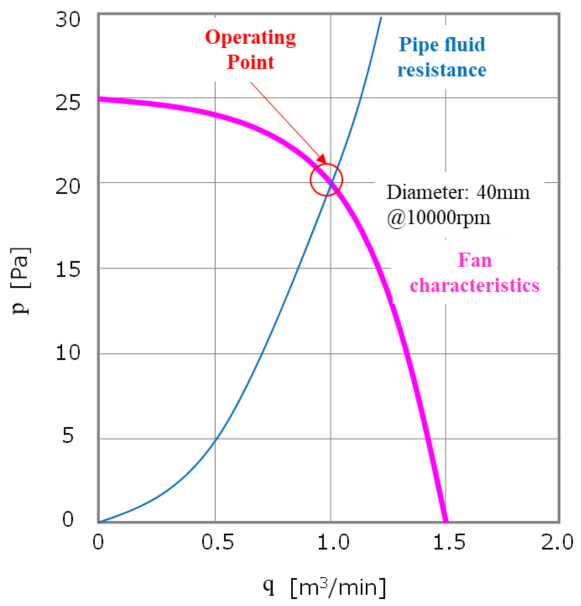


Figure 9. Model of the electric power supply and heater



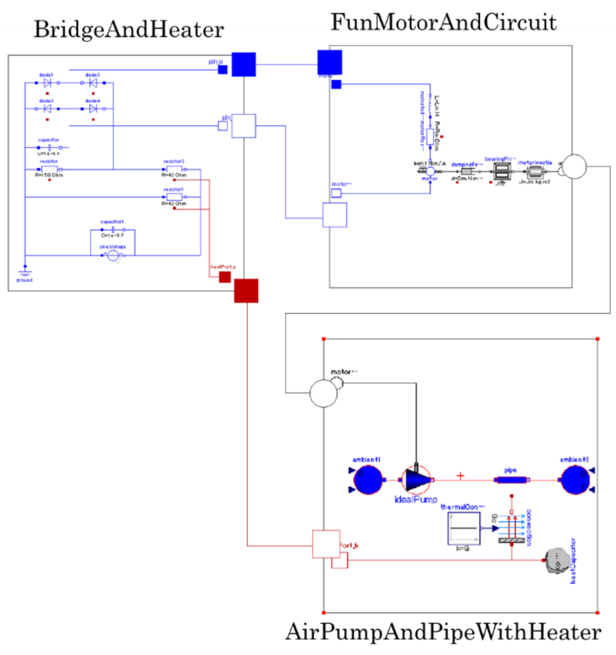


Figure 10. Model of the whole dryer

p : Pressure
q : Flow rate
n : Normalized speed

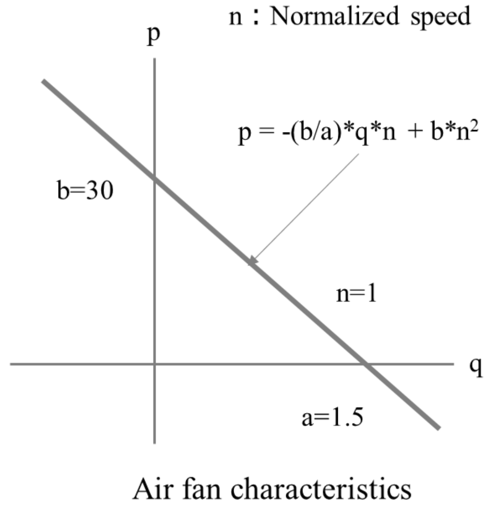


Figure 8. Characteristics of air fan and pipe resistance

3.2 Simulation results

Figure 11 shows one result of the dryer model shown in Figure 10. It is confirmed that the motor voltage is full-wave rectified from the sinusoidal input voltage and the pump speed is controlled according to the motor voltage. Finally, the pipe air flow temperature is raised from 20 degC to about 65 degC.

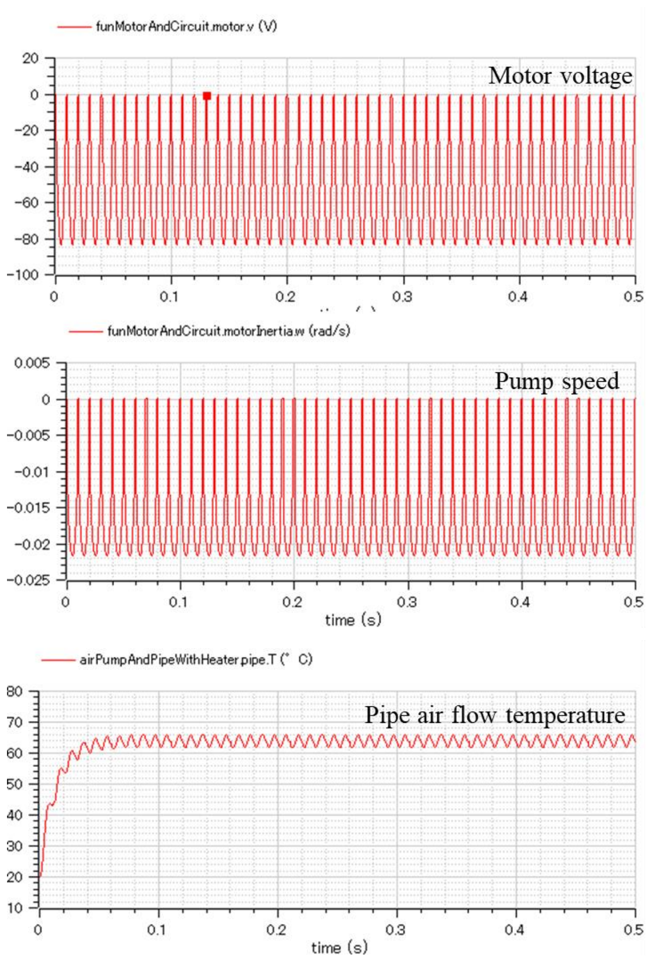


Figure 11. Simulation results of the dryer model

4 Modeling of a speaker

4.1 Structure of the speaker model

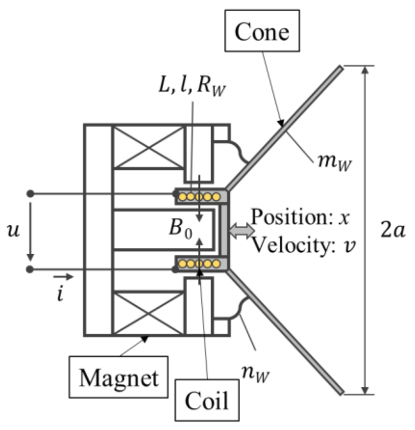


Figure 12. Structure of a speaker

The structure of a speaker is shown in Figure 12. Electric input of voltage u and current i are given to the solenoid coil. The coil generates magnetic flux density B_0 and transfer the electromotive force generated between the magnet to the cone. Finally, the cone is oscillated by the electromotive force and generates the oscillation of the air resulting in the acoustic sound. The system of equations become as follows.

(Voltage source)

Voltage: $u = k_0 \sin(\omega t)$ (20)

(Electric circuit with solenoid coil)

$$u = R \cdot i + L\frac{di}{dt} + u_{emf} \tag{21}$$

(Solenoid coil)

Electromotive voltage: $u_{emf} = B_0 l v$ (22)

Electromotive force: $F_w = B_0 l i$ (23)

(l : coil length)

(Mechanical part)

Mechanical mass:

$$m_w\frac{dv}{dt} = F_m + F_w - F_p \tag{24}$$

Spring damper force:

$$F_m = -\frac{v}{r_w} - \frac{x}{n_w} \tag{25}$$

Here, m_w: Mass of the movable part, r_w: Friction admittance (= inverse of damping coefficient), n_w: Compliance (= inverse of spring constant), F_p: Reaction force of acoustic vibration.

(Acoustic characteristics of the cone)

To model the acoustic characteristics by an analogy of mechanical and electrical system, the equivalent elements shown in Figure 13 are considered [Lenk, 2011]. For the speaker shown in Figure 12, the acoustic resistance and the acoustic mass of the air oscillated by the cone become as below [Lenk, 1995].

Acoustic resistance:

$$Z_{aL} = \frac{1}{2}\frac{\rho_L c_L}{\pi a^2}\left(\frac{\omega}{c_L}a\right)^2 \tag{26}$$

Acoustic mass:

$$M_{aL} = \frac{8}{3}\frac{\rho_L}{\pi a^2} \tag{27}$$

(c_L: sound speed of air, ρ_L: density of air, a: cone radius)

Above equations are valid when the following condition is met.

$$\omega < \omega_g = \sqrt{2}\frac{c_L}{a}$$

Between the mechanical characteristics and the acoustic characteristics of the cone, the following equations hold.

$$v = \frac{1}{A}q \tag{28}$$

$$F_p = A \cdot p \tag{29}$$

$$p = p_m + p_z \tag{30}$$

$$p_m = M_{aL}\frac{dq}{dt} = M_{aL}A\frac{dv}{dt} \tag{31}$$

$$p_z = Z_{aL}q = Z_{aL}Av \qquad (32) \qquad (A = \pi a^2\text{: Cross sectional area of the cone})$$

	Electrical	Mechanical	Acoustic	Symbol
Potential variable	u : Voltage	v : Velocity	p : Pressure	
Flow variable	i : Current	F : Force	q : Volume flow rate	
Inductive element	L : Inductance $u = j\omega L\, i$	n : Compliance $v = j\omega nF$	M : Acoustic mass $p = j\omega Mq$	
Capacitive element	C : Capacitance $u = \frac{1}{j\omega C}i$	m : Mass $v = \frac{1}{j\omega m}F$	N : Acoustic compliance $p = \frac{1}{j\omega N}q$	
Dissipative element	R : Resistance $u = R\, i$	h : Friction admittance $v = hF$	Z : Acoustic friction $p = Zq$	

Figure 13. Analogy of elements in electrical, mechanical and acoustic system

4.2 Text code model of the speaker

Considering the equations (20) to (32), the text code model of the speaker becomes as follow.

```
model ModelByText
 import SI = Modelica.Units.SI;
 import Modelica.Constants.pi;
 parameter SI.MagneticFluxDensity B0 = 0.8;
 parameter SI.Length l = 8 "Coil lenght";
 parameter SI.Resistance R = 6 "Electric
 register";
 parameter SI.Mass mw = 0.03 "Mechanical
 movable mass";
 parameter SI.Inductance nw = 2.1e-3
 "Mechanical inductance of spring";
 parameter SI.Admittance rw = 1e-3
 "Mechanical admittance of damper";
 parameter SI.Mass ml = 3.2e-3 "Mechanical
 movable mass of air";
 parameter SI.Admittance rl = 8e-3
 "Mechanical admittance of air damper";
 parameter SI.Admittance rel =6.82
 "Mechanical admittance of electric
 register";
 parameter SI.Area A=pi*0.1^2 "Area of
 speaker cone";
 parameter SI.Frequency freq = 5
"Oscillation frequency" ;
 SI.ElectricCurrent i;
 SI.Voltage u0;
 SI.Length s;
 SI.Velocity v;
 SI.Acceleration a;
 SI.Force fw;
 SI.Force fm;
 SI.Force fp;
 SI.Force f;
 SI.Pressure p(start=0, fixed=true);
 SI.MassFlowRate q;
equation
 v = der(s);
 a = der(v);
 u0 = 1*sin(2*pi*freq*time);
 u0 = i*R + B0*l*v;
 fw = B0*l*i;
 fm =  -v/rw - s/nw;
 mw*a = fm + fw - fp;
 f = fm - fp;
 p = ml*A*a + rl*A*v;
 q = A*v;
 fp = A*p;
end ModelByText;
```

4.3 Speaker model using MSL

To make the MSL based model of the speaker, the system is translated to the integrated mechanical model. From the equations (29) to (32), we obtain the following equation.

$$F_p = A \cdot p = M_{aL}A^2\frac{dv}{dt} + Z_{aL}A^2v \qquad (33)$$

By using the equations (24), (25) and (33), the integrated equation as the mechanical region is obtained.

$$m_w\frac{dv}{dt} = F_w - \frac{1}{r_w}v - \frac{1}{n_w}s - M_{aL}A^2\frac{dv}{dt} - Z_{aL}A^2v \qquad (34)$$

From the equations (21) to (23), we also obtain an equation about electrical system converted to the mechanical model. Here, the inductance of the coil L is ignored for the simplicity.

$$F_w = B_0li = B_0l\left(\frac{u}{R} - \frac{B_0lv}{R}\right) \qquad (35)$$

Finally, from the equations (34) and (35), an integrated equation of the total system as a mechanical expression is obtained as below.

$$\frac{B_0 l}{R} u = \frac{(B_0 l)^2}{R} v + m_w \frac{dv}{dt} + \frac{1}{r_w} v + \frac{1}{n_w} s + M_{aL} A^2 \frac{dv}{dt} + Z_{aL} A^2 v \tag{36}$$

To make a model of the equation (36) based on MSL, the following classes of the mechanical inductance, mechanical mass and mechanical admittance shown in the Figure 13 were made as follows.

```
model MechanicalInductance
 "Sliding inductancde"
  parameter Real n(min = 0, start = 1)
 "Mechanical Compliance";
  extends
Modelica.Mechanics.Translational.Interface
s.PartialCompliantWithRelativeStates;
equation
  n * der(f) = v_rel;
end MechanicalInductance;

model MechanicalMass
 "Sliding mass with inertia"
  parameter SI.Mass m(min = 0, start = 1)
 "Mass of the sliding mass";
  parameter StateSelect stateSelect =
StateSelect.default
 "Priority to use s and v as states" ;
  SI.Velocity v(start = 0, stateSelect =
stateSelect)
 "Absolute velocity of component";
  SI.Acceleration a(start = 0)
 "Absolute acceleration of component";
equation
  v = der(s);
  a = der(v);
  m * a = flange_a.f + flange_b.f;
 end MechanicalMass;

model MechanicalAdmittance
 "Linear 1D translational damper"
  extends
Modelica.Mechanics.Translational.Interface
s.PartialCompliantWithRelativeStates;
  extends
Modelica.Thermal.HeatTransfer.Interfaces.P
artialElementaryConditionalHeatPortWithout
T;
  parameter Modelica.Units.SI.Admittance h
(final min = 0, start = 0)
 "Damping conductance";
equation
  h * f = v_rel;
  lossPower = f * v_rel;
end MechanicalAdmittance;
```

The icons of each class are shown in Figure 14.

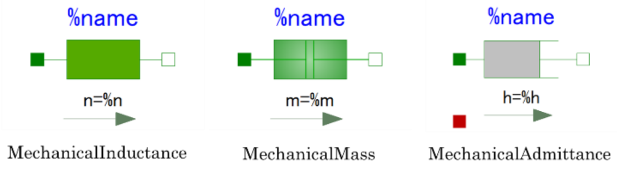


Figure 14. Icons of the mechanical elements

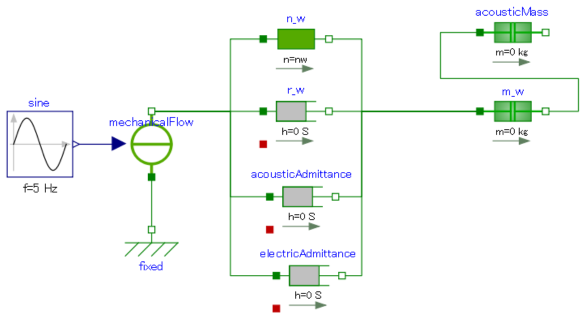


Figure 15. The speaker model based on MSL

Finally, the model of the speaker based on MSL becomes as Figure 15. Here the class "mechanicalFlow" converts a scalar input to force output.

4.4 Simulation results

Figure 16 shows sample results of both the text code model and MSL based model of the speaker model. The results of the both models seem almost identical.

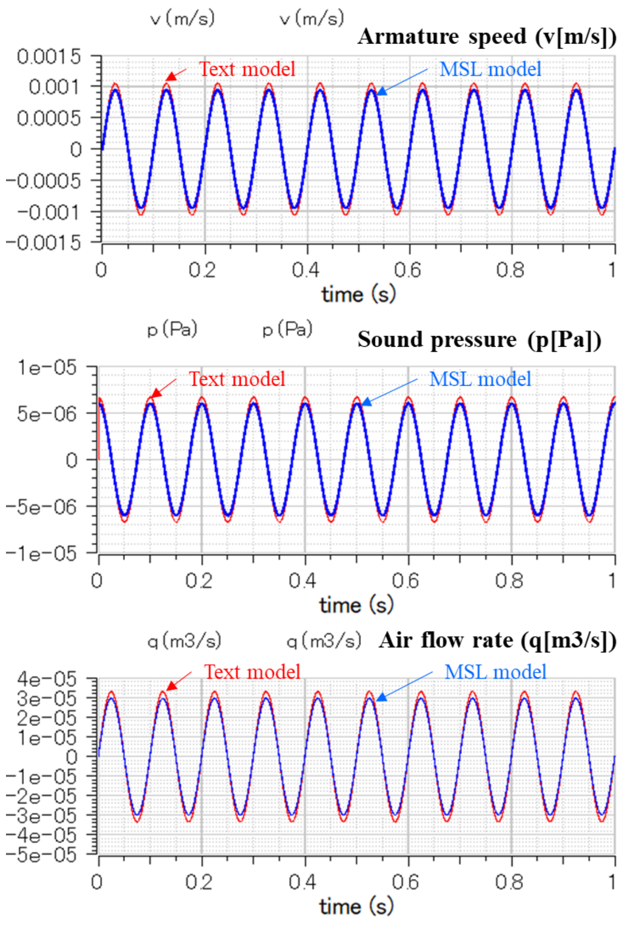


Figure 16. Simulation results of the speaker model

5 Conclusion

Many physical models of daily appliances by Modelica were presented. Once the physical equations were determined, it was very easy and efficient to make the simulation models by Modelica. Both of text code model and MSL based model were developed and simulated. To learn about those physics and also Modelica modeling was very efficient for the education of students and engineers. For some examples such as fluid system of dryer, Modelica was also useful to solve the simultaneous equations for obtaining the design parameters.

References

Hirano, Y. Development of New Concept Vehicles Using Modelica and Expectation to Modelica from Automotive Industries, Proceeding of Modelica Conference 2012.

Fujimoto, H. et al. Range Extension Control System for Electric Vehicle Based on Searching Algorithm of Optimal Front and Rear Driving Force Distribution, Proc. 38th Annual Conference of IEEE Industrial Electronics Society, pp. 4244-4249, 2012.

Pelchen, C. et al. Modeling and Simulating the Efficiency of Gearboxes and of Planetary Gearboxes, Proceedings 2nd International Modelica Conference, pp. 257-266, 2002.

A. Lenk, R.G.Ballas, R.Werthschützky, and G.Pfeifer. Electromechanical Systems in Microtechnology and Mechatronics. Springer, 2011.

A. Lenk. Grundlagen der Akustik. Skript zur Vorlesung. TU Dresden. Institut für Technische Akustik, Dresden, 1995.

Importing FMU-3.0: challenges in proper handling of clocks

Masoud Najafi   Ramine Nikoukhah

Altair Engineerng, France {masoud,ramin}@altair.com

Abstract

Compared to FMI-2.0, FMI-3.0 provides support for events and clocks. The behavior of the FMU in the presence of events and clocks introduces new challenges for importing FMUs in block diagram environments such as **Altair Activate** and **Scicos**. This paper discusses some of these challenges and proposes implementation strategies for supporting the import of FMI-3.0 in **Activate**.

Keywords: FMI-3.0, Synchronous clock, Signal based tool, Modelica tool

1 Introduction

The Functional Mock-up Interface (FMI) (Modelica Association, 2022) has become a de-facto tool independent standard for the exchange of dynamic models and for co-simulation. FMI-3.0 (Specification, 2022) version of the standard introduces many new features that allow for more advanced modeling and support for co-simulation algorithms. Clocks allow the synchronization of events between Functional Mock-up Units (FMUs) and the simulator (importer). Several new data-types and multi-dimensional arrays are also supported (Junghanns et al., 2021).

Activate is a modeling and simulation tool developed by Altair Engineering based on the open-source academic simulation software **Scicos** (INRIA). **Activate** environment can be used to create models of dynamical systems as signal-based block-diagrams. The basic blocks, such as FMUs can be interconnected to build complex model. This is very similar to the way diagrams are built in the SSP [1] (System Structure and Parametrization) standard.

Activate can also be used to create Modelica diagrams (Nikoukhah and Furic, 2009). The integration of the Modelica part of the model is done first by the aggregation of the Modelica components and creation of a Modelica program which is then processed by the Modelica compiler.[2] In **Activate**, the Modelica compiler provides an FMU block replacing the Modelica components in the original model.

Because of this FMI based integration of Modelica in **Activate**, **Activate** has been providing FMU import support through an **Activate** FMU block. More generally this block is also used for importing FMUs from other sources. The support of FMI-2.0 in the **Activate** environment had already been challenging and specific solutions had to be developed; the main problem being the way input-output dependencies are defined and treated in **Activate** and in FMU. See (Nikoukhah et al., 2017).

With FMI-3.0 and the introduction of the notions of clock, activation and synchronization, the FMU import in **Activate** presents new challenges. Even though the activation signals and synchronism have been part of the **Activate** semantics from the beginning, the small semantic differences between FMI-3.0 and **Activate** formalism makes it so that an FMU cannot be imported as a basic block in **Activate**. This was already not the case in some situations with FMI-2.0, as was presented in (Nikoukhah et al., 2017). With FMI-3.0, the problem becomes more involved.

This paper presents the difficulties and the solutions envisaged to provide maximum support for FMI-3.0 import in **Activate**. First a short overview of the way **Activate** handles activations (clocks) is provided and the differences with the FMI-3.0 treatment of clocks are discussed.

In Section 4, the solutions for importing FMI-3.0 in **Activate** are presented by considering different types of clocks. Each section provides an FMU example to illustrate the process.

2 Activate environment and activation signals

2.1 Double layer implementation

In the **Activate** environment, a model is constructed using blocks. The compiler however does not operate on these blocks; it interacts with Atomic Units[3] (AU). In many cases a block is associated with a single AU, but not always: a block may produce a diagram containing multiple connected AUs. This diagram produced programmatically by the block may depend on the values of the block parameters. Specifically, the choice of the AU(s), their parameters, and the topology of the diagram is specified by an OML[4] function associated with the block, which constructs the diagram based on the values of the block parameters.

The ability to programmatically instantiate an AU or a diagram of AU(s) is a powerful mechanism which is used

[1] `https://ssp-standard.org/`

[2] The Modelicac compiler is used to to compile and generate code for the modelica program in **Scicos**; the MapleSim compiler is used to generate an FMU in **Activate**.

[3] Also called basic blocks.

[4] A matrix based interpreted matrix-based language similar to Scilab, Octave, Matlab.

to present to the user as a block, for example through a library, a complex construction based on a diagram of AUs. The FMU block is an example of such a construction.

In general, an AU provides computational function APIs to be used by the simulator. The APIs are C functions that are called by the simulator at different stages of the simulation: computations of the outputs, of the state derivatives, of the next discrete state, etc. **Activate** compiler uses AU properties to construct the compiled structure of the model to be used by the simulator. These properties include for example the feedthrough properties of the AUs used by the compiler for proper scheduling of the activation order of the AUs. The computations done by the corresponding APIs however are transparent to the compiler.

Two very special AUs *IfThenElse* and *SwithCase* basic blocks play a fundamental role in defining conditional operations, used for example for subsampling. They are actually language constructs similar to *if* and *switch* statements in most programming languages, and are treated in a special way by the compiler. Other "special" AUs include activation sources such as the *InitialActive*, *AlwaysActive* and *SampleClock* blocks. The latter produces an activation signal containing a series of periodic events. Multiple *SampleClock* blocks can be used within a model with identical or different periods. The compiler treats them as synchronous clocks even if they don't have identical periods.

2.2 AU interactions

AUs have input and output ports. These ports are connected by links which represent the sharing of data between the ports. The value of an input port is provided by the output port linked to it. The AU of the output port computes the signal to be read by the AU of the input port, which in turn computes its outputs, when activated.

In the simple case where all the AUs are "always active" (continuous-time dynamics), the **Activate** compiler determines the order in which the AUs should be activated (their APIs called) to guarantee the signals flow properly in the network. This order is stored in the compiled structure of the model and used during simulation. It is also used for code generation.

In many cases however all the AUs in a model are not "always active". Consider for example the model of a physical plant controlled by a discrete-time controller. In such a model, some of the AUs are continuously activated (so always active) and others only at the ticks of the controller clock.

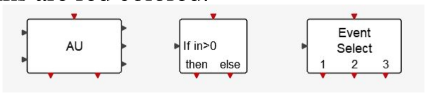
In the presence of multiple sources of activations, the compiler determines the order of block executions for all possible activation scenarios and stores them in the compiled structure to be used by the simulator[5] and the code generator.

[5] No online scheduling is ever performed by the simulator.

2.3 Activation signals and AU activation

AUs in **Activate** are activated by activation signals. The "always active" signal is an example of such a signal. An AU activated by such a signal is continuously active. Discrete activation signals define one or more isolated time instants of activation (called events). An AU activated by such a signal is activated at these discrete time instants.[6]

At the graphical level, by default the regular input and output ports are placed on the sides of the blocks and the activation input and output ports, respectively, on the top and at the bottom of the block. The activation ports and links are red colored.



The block on the left is a general AU with multiple input, output, regular and activation ports. The other two blocks are special AUs *IfThenElse* and *SwithCase* used to redirect their input activations to one of their output activation ports depending on the value of their regular inputs. These two AUs produce output activations which are synchronous with their input activations; something which is not possible with any other AU.

To simplify the construction of models at the graphical level, two mechanisms are used in **Activate** to reduce the number of activation ports and links:

- **Always active AU property:** Instead of explicitly creating a link from an always active activation source to the AU, the AU can be declared as having "always active" property.

- **Activation inheritance:** if an AU is not declared always active and does not have any activation input ports, then it can inherit its activations from its regular input signals. Specifically, it is activated by the activation signals which have activated the block which have produced its input signals.

These mechanisms are mere syntactic sugars: the corresponding activation signals are added to the model at a pre-compilation phase.

An AU may be activated by one or more activation signals, through one or more activation input ports. See Fig. 1 where the *EventDelay* block is activated by the union of two activation signals. The resulting activation signal contains then an initial event and events produced by the block itself, which are the delayed version of previous events. This model produces an activation signal consisting of a series of events evenly spaced in time (like an event clock).

For an AU having more than one activation input port, the AU is activated if any of the input ports receives an activation. In that case, the computational function API can know what activation(s) has caused the activation of the

[6]More generally an activation signal may define a union of isolated points and time intervals as activation times. But this level of generality is not pertinent to the FMI import issue considered here.

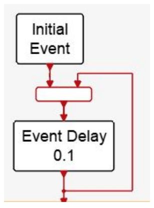


Figure 1. A new activation signal can be constructed from two or more activation signals where the activation times of the new signal is the union of the activation times of the other signals. This operation is realized by the *EventUnion* block as shown in this diagram.

AU. For example if the AU has two activation input ports, then it can be activated either because an activation signal has been received on its first input, on its second input or on both synchronously.[7] The way by which the activation has occurred is coded as an integer (the binary coding of the integer represents the input ports at the origin of the AU activation, so function call). In the case of AU with two activation input ports, the integer can take values 1, 2 and 3. The API can perform different computations for each value of the integer.

Even though the AU can be activated through different combinations of activation signals, there are no individual activation signals associated with each output port of the AU. The outputs are updated at times corresponding to the union of all the activations activating the AU. So, the activation signals associated with the outputs are all identical; the AU cannot associate individual activation signals to the AU outputs. Even if an output of the AU is not computed by the AU computational function API for a particular activation, it is considered to be up to date and is treated as if it had been recomputed (signals in **Activate** are persistent). This property is in contrast to the FMU-3 way of associating outputs to different clocks, making it impossible to represent an imported FMU as a single AU in the general case.

An AU can have activation output ports. An AU cannot generate an event which is synchronous with the event which has activated it. The generated event is delayed with respect to the activation of the block. The time delay can be set to zero, making the two events having the same time, but not synchronous. Synchronous events can only be generated by two special AUs *IfThenElse* and *SwithCase*. For details see (Campbell et al., 2010; Ext, 2022). This is another reason why a single AU cannot always represent an imported FMU-3.

The explicit treatment of Activation signals in **Activate** makes the import of FMU-3's amenable but not necessarily as single AUs (basic blocks). This was the case already for FMU-2, as we will recall in the next section. We will then show how FMU-3's can be imported as **Activate** blocks including multiple AUs. From the user point of view, this process is completely transparent. They will place the FMU block from the palette in the diagram and edit its parameter to point to the FMU to import. The block will then read the content of the FMU and programmatically create the content of the block.

[7]Here we assume that the block is not continuously active.

3 Activate FMU block for importing FMI-2.0 FMUs

This section recalls the way FMU-2's are imported in **Activate**. The process was in part presented in (Nikoukhah et al., 2017). The import of FMU-2 was a simpler task because there were no clocks and clock activations to consider; the system was always active. The main difficulty had to do with the way output/input dependencies are specified in FMI. In an AU, output/input dependencies are expressed as a vector of dependencies specifying which inputs affect any of the outputs. So, the dependency is solely a property of an input port. The reason is that an AU computes all of its outputs during a single activation, i.e., in the same API call, so all of its dependent inputs must be up to date when the call is made. An FMU on the other hand specifies output/input dependencies as a matrix specifying which output depends on which known variables including individual inputs. The FMU provides routines that allow the computation of output ports separately and take advantage of variable caching.

One way to deal with this discrepancy is to simply project the matrix of dependencies provided by the FMU into a vector of dependencies as required by **Activate**. This conservative approach properly assigns dependencies in **Activate** but "loses" information along the way. When one or more FMUs are imported in an **Activate** model, this may lead to the detection of algebraic loops by the **Activate** compiler that are not true algebraic loops (artificial algebraic loops). The result is that valid algebraic-loop-free models may end up not compilable by **Activate**.

There is no solution to this problem as long as the FMU block is to be implemented as a single AU. But as it was stated previously, **Activate** blocks can implement a diagram of AUs, the topology of which can depend on block parameters. It turns out, (Nikoukhah et al., 2017), that the matrix output/input dependency information provided by the FMUs can be implemented by a properly constructed diagram of AUs. The diagram would include a distinct AU associated with each output port, in charge of computing the output, and the diagram constructed so that its topology reflects the output input dependencies. Consider for example an FMU with 2 inputs and 2 outputs where the only output input dependency is that the first output depends on the first input. Then the FMU block could create the diagram shown in Fig. 2.

The dependency information is provided in the FMU XML file, which is available as a block parameter of the **Activate** FMU block. By reading and parsing the XML inside the FMU, the block generates a diagram of AUs. The diagram contains a central AU, always present, and an AU associated with each FMU output. The input ports of

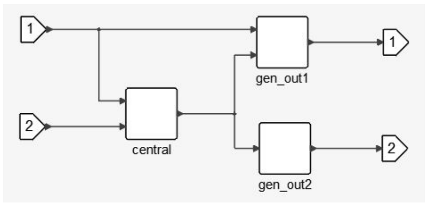


Figure 2. In this example, the first output depends directly on the first input. The second output does not depend directly on any input.

these AUs and their connections are tailored to the dependency information read from the XML file. In particular the AU associated with an output will have an input corresponding to an input of the FMU only if the corresponding output input dependency property is true.

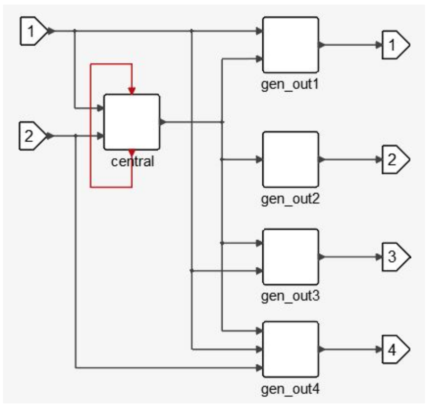


Figure 3. A central AU, always present, handles the state update tasks of the FMU and provides the FMU structure to the AUs in charge of computing the outputs. The output AUs are only connected directly to the inputs of the block if there is a corresponding output input dependency as specified in the FMU XML file.

The central AU includes the simulation APIs for state derivative computation and discrete state updates, etc., and does not have any input dependency. All the AUs in the network use the same internal structure, which is instantiated by the central AU. The central AU provides this structure to the other AUs through its output port.

The central AU is also endowed with an activation input and an activation output port. These ports are connected together with an activation links. The delayed events reactivating the central AU are used to implement time-events in FMI-2.0.

Fig. 3 shows a typical diagram resulting from the import of an FMU with 2 inputs and 4 outputs.

4 FMI-3.0 support

FMI-3.0 provides a number of new features for both Model-Exchange and Co-Simulation (Gomes et al., 2021). Some of the new features of FMI-3.0 are intrinsically supported in **Activate**. For example AU input output ports are not limited to scalars; they can be of type matrix and of different data types. But even though AUs have activation (clock) input and outputs, the semantic differences between FMI-3.0 clocks and **Activate** activations does not allow a simple mapping of FMI clocks into **Activate** activation signals.

Different types of FMI clocks require different treatments during the import process, as shown in the following sections. For each clock type in FMI-3.0, an FMU has been considered and the way it is imported in **Activate** is explained. Note that the way the clock is handled is FMI-3.0 is independent of the FMU type, *i.e.*, the FMU can be either Model-Exchange or Co-Simulation[8]. The FMU examples work identically for both FMU types.

4.1 Triggered input clocks

It may seem natural to map an FMU-3 with multiple input triggered clocks into an AU with multiple activation input ports. This however is not semantically correct because different outputs of the FMI may be associated with different clocks, *i.e.,* outputs may be differently clocked. But in an AU, all the outputs of the AU are computed on the union of all the activations activating the AU. So, a single AU can capture this aspect of the FMU behavior only if all the outputs of the FMU are associated with all of its clocks. Any other clock association requires a specific treatment.

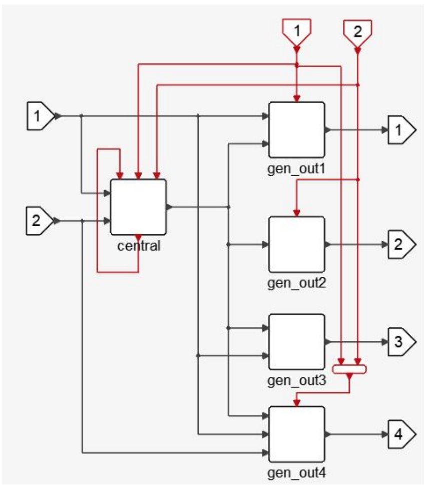
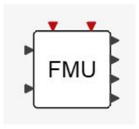


Figure 4. The diagram resulting from the import of an FMU-3 with 2 triggered clocks.

Consider the imported diagram in Fig. 3 and assume additionally that the imported FMU is an FMU-3 with two triggered input clocks where the first output is a clocked variable associated with the first clock, the second output is a clocked variable associated with the second clock and the last output associated with both. The third output is continuous-time (its `variability` attribute is `continuous`). The FMU block importing this FMU will instantiate[9] in the **Activate** model as shown below

[8]The Scheduled Execution FMU type has not been considered in this paper

[9]The block ports are automatically adjusted to the FMU specification



Its underlying diagram is shown in Fig. 4. The `gen_out3` block is declared always active.

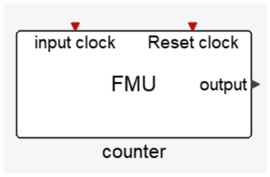
The generalization to the case where more input triggered clocks are present is straightforward.

4.1.1 Example: Clock tick counter with reset FMU

This FMU increments its output on each input clock tick.[10] The counter is reset to zero on the tick of the second input clock. The FMU has two triggered input clocks and one regular output port. The FMU model description for these ports are as follows:

```
<Clock name="input clock"   valueReference="4"
   causality="input" variability="discrete"
   intervalVariability="triggered"
   description="counter increments on ticks"/>

<Clock name="Reset clock"   valueReference="5"
   causality="input" variability="discrete"
   intervalVariability="triggered"
   description="Resets to zero on ticks"/>

<Int32 name="pre(counter)" valueReference="6"
   initial="exact"  variability="discrete"
   causality="local" description="pre(counter)"
   start="0" clocks="4 5"/>

<Int32 name="counter"      valueReference="7"
   previous="6"      initial="calculated"
   variability="discrete"   causality="local"
   description="counter internal value"
   clocks="4 5"/>

<Int32 name="output"       valueReference="8"
   variability="discrete"   causality="output"
   description="counter value"  clocks="4 5"/>
```

This FMU is imported as follows



The content of this FMU is shown in Fig. 5. Note that since there is no information in the model description of the FMU about using the time-events by the FMU, the central AU has always its first output activation port connected to its first input activation port (clock feedback).

4.2 Periodic clocks

In FMI-3.0, a time-based input clock can be defined as being periodic. The period and the offset of the clock can be constant or user-defined.

when the block parameters, in particular the FMU name and location, are provided as block parameters.

[10]The snippets of the C source code of the FMU are provided in the Appendix. FMU's presented in this paper are available upon request.

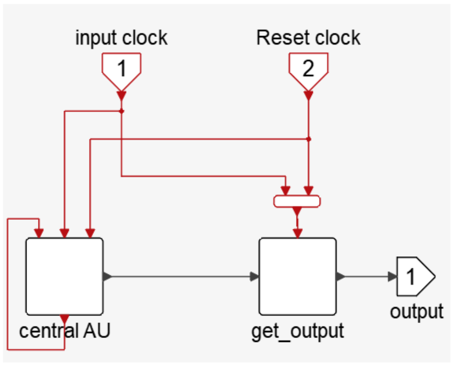


Figure 5. Importing the clock tick counter with reset FMU in **Activate**.

In the corresponding **Activate** block, such an input clock is not represented by an activation input port. Instead, the periodic clock is explicitly placed inside the diagram. Consider again the example with 2 inputs and 4 outputs and two clock inputs but now suppose the second clock is periodic with period P. The imported diagram can then be constructed as shown in Fig. 6.

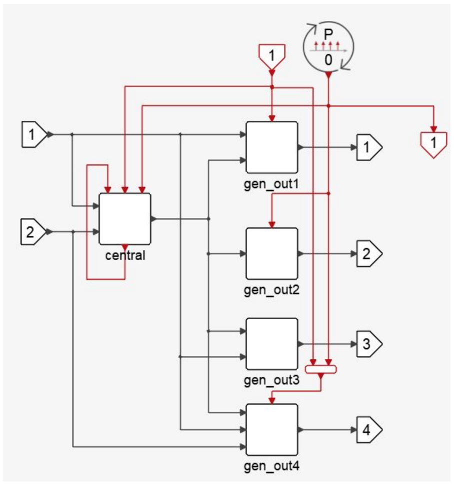
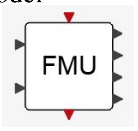


Figure 6. The second FMU clock is periodic. It does not lead to an input activation port, instead it is realized using a *SampleClock* block.

Note that the **Activate** FMU block in this case has only one activation input port. The periodic clock is placed inside the diagram and realized by a *SampleClock*.[11] In order to connect this triggered input-clock to other FMUs, an output clock port is added to the imported FMU block. This output activation port looks as follows in the **Activate** model



The generalization to more mixed triggered-periodic clock inputs is straightforward to imagine.

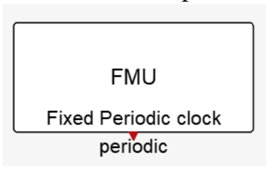
[11]All the SampleClock s in the model are synchronized by the compiler, even if they are in different diagrams.

4.2.1 Example: Periodic clock FMU

This FMU creates periodic clock ticks. The period and shift time (initial offset time) can be set by the user. The FMU does not have any regular output ports.

```
<Clock name="Fixed Periodic clock"
   valueReference="4" variability= "discrete"
   causality="input" intervalVariability="fixed"
   intervalDecimal="1.0" shiftDecimal="0.2"
   description="Fixed periodic clock "/>
```

This FMU is imported as follows



The content of this FMU is shown in Fig. 7.

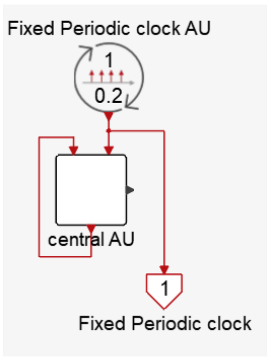


Figure 7. Importing the clock tick counter with reset FMU in **Activate**.

Although the periodic clocks of an FMU have `causality=input` attribute, these input clocks can be connected to other FMUs. For example, input periodic clock can be connected to triggered input clocks of other FMUs. In a signal based environement such as **Activate**, two input ports cannot normally be connected, due to causality incompatibility. However, with the way these FMUs are imported in **Activate**, this type of connection becomes natural.

The connection of the counter FMU and the periodic clock FMU is straightforward now and can be done in **Activate** as shown in Fig. 8.

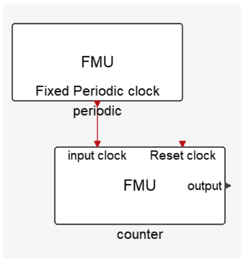
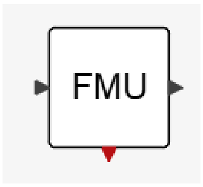


Figure 8. Connection of the periodic clock FMU to the counter FMU in the **Activate** model.

4.3 Aperiodic clocks

In FMI-3.0, a time-based input clock can also be defined as being aperiodic, i.e., `changing clock` and `countdown clock`. At each clock tick (or any event time for `countdown clock`), the time instant of the next clock tick is retrieved by the simulator (if any). This is similar to the way time events are handled in FMI-2.0, with the difference that the synchronism is ensured by the fact that the simulator (the importer) clearly activates the clock tick. Another difference between the ordinary time-event and aperiodic clocks is that unlike the time-events, clock ("input") ports can be connected to the triggered input clocks of other FMUs.

When imported, in the corresponding **Activate** block, such an input clock is not represented by an activation input port. Instead, an aperiodic clock (`changing clock` or `countdown clock`) is represented internally by an input clock and output clock in the central AU block. The output block is activated by the input clock. This is identical to the way time-events are handled in **Activate**. For example, importing an FMU with an input clock of type aperiodic results in



The content of this block is shown in Fig. 9.

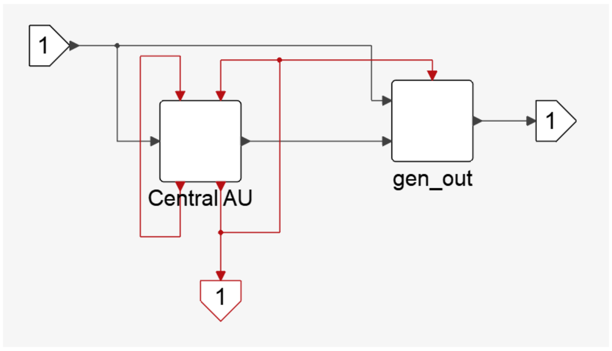


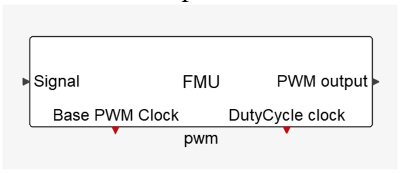
Figure 9. Importing an FMU with an aperiodic clock in **Activate**.

4.3.1 Example: PWM signal generator FMU

This FMU receives a continuous-time signal as input and creates a PWM (Pulse-Width Modulation) signal. The rate or the frequency of switching of the PWM is created by a periodic clock with `intervalVariability` attribute set to `fixed`. The duty cycle of the PWM varies as a function of the input signal, *i.e.*, at input equal to 0.0, the duty cycle is 0% and at input equal to 1.0, the duty cycle is 100%. The period and the first tick instant of the switching (defined by `intervalDecimal` and `shiftDecimal` respectively) are set by the user. In order to create the duty cycle switchings, a countdown clock is used. At every tick of the periodic clock, the next tick of the countdown clock is scheduled as a function of the input signal value. For the sake of clarity, the snippet of the C source code is given in the appendix.

```
<Clock name="Base PWM Clock" valueReference="1"
  causality="input" variability= "discrete"
  intervalVariability="fixed"
  intervalDecimal="0.1" shiftDecimal="0.0"
  description="PWM Clock" />

<Clock name="DutyCycle clock" valueReference="2"
  causality="input"      variability="discrete"
  intervalVariability="countdown"
  description="Duty cycle tick clock" />

<Float64 name="Signal"       valueReference="3"
  causality="input"    variability="continuous"
  description="input signal" start="0.1"
  clocks="1"/>

<Float64 name="PWM output"   valueReference="4"
  causality="output"    variability="discrete"
  description="PWM output"  clocks="1 2"/>
```

This FMU is imported as follows



The content of this FMU is shown in Fig. 10.

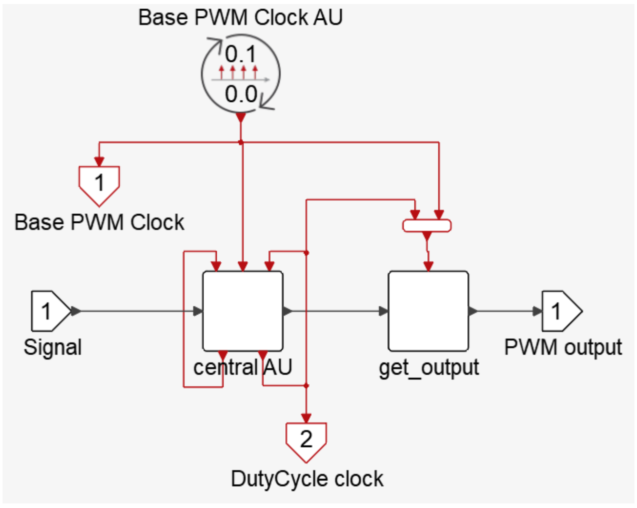
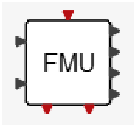


Figure 10. The content of the PWM FMU when imported in **Activate**.

4.4 Triggered output clocks

The FMI triggered output clocks correspond to output activation ports of the **Activate** FMU block. For example, if the FMI considered previously additionally has a triggered output clock, the corresponding **Activate** FMU block looks as follows in the **Activate** model



If the output clock is not synchronous with any of the input clocks, then the corresponding **Activate** event can be generated directly by the central AU. See Fig. 11.

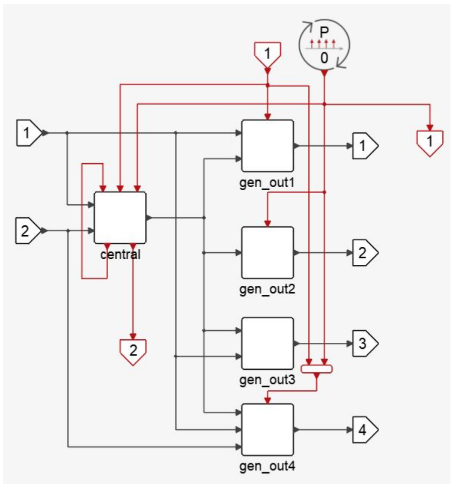


Figure 11. The diagram resulting from the import of an FMU-3 having a periodic clock and an asynchronous output clock, for example a clock triggered by an internal zero-crossing event.

On the other hand, if an output clock is dependent on (is synchronous with in the **Activate** terminology) an input clock, then it cannot be created as the output of the central AU. Only two special "blocks" *IfThenElse* and *SwitchCase* output activations are synchronous with their input activations.

Consider the same FMU again but now assume the output clock is dependent on the first input clock. The diagram can now be realized as shown in Fig. 12. In this case

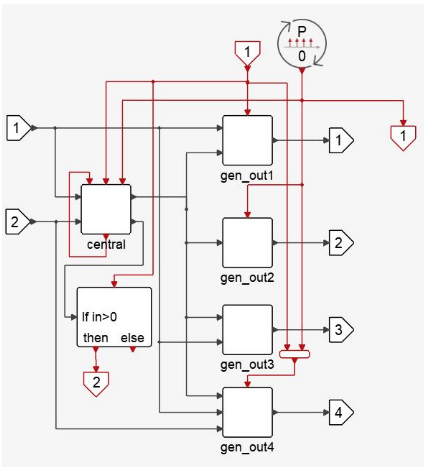


Figure 12. The diagram resulting from the import of an FMU-3 having a periodic clock and a synchronous output clock.

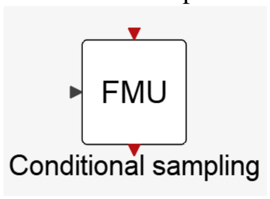
the activation of the synchronous clock is "signaled" via an additional output of the central AU. This Boolean signal has value true if the clock is to be fired. By feeding this value to an *IfThenElse* to generate (or not) the corresponding event, the event becomes synchronous with the corresponding input event (FMU clock).

4.4.1 Example: Conditional sampling FMU

This FMU has a triggered input clock, a triggered output clock, and a regular input port. The triggered output clock is activated synchronously with the input clock, only if the regular input of the FMU has a positive value. In this FMU, there is a direct dependency between the output clock and the input clock and should be handled correctly.

```
<Float64 name="Condition" valueReference="1"
 causality="input"  variability="discrete"
 description="condition for clock"  start="0" />

<Clock name="Input clock"   valueReference="2"
 causality="input"  variability="discrete"
 intervalVariability="triggered"
 description="Input clock from any source"/>

<Clock name="output clock"  valueReference="3"
 causality="output" variability="discrete"
 intervalVariability="triggered" clocks="2"
 description="Clock triggers if Condition>0"/>
```

This FMU is imported as follows



The content of this FMU is shown in Fig. 13.

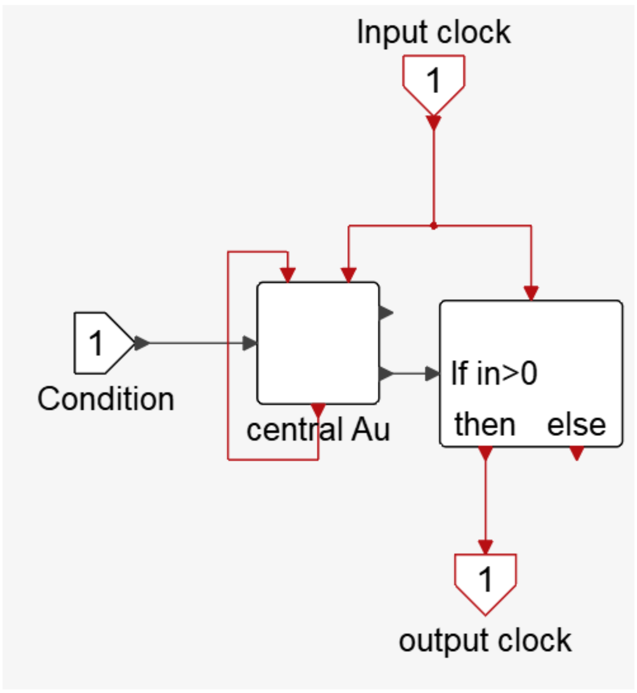
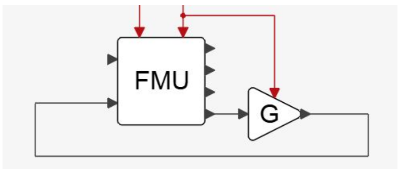


Figure 13. The content of the conditional output clock FMU when imported in **Activate** to keep the input/output synchronicity.

4.5 Clocked inputs

The FMU-3 input ports can be clocked, *i.e.,* inputs can be associated with clocks (both input or output clocks). When the FMU is imported, this information is used in the central AU to read inputs only when it can be accessed and is needed. But this information can also be used in the construction of the imported diagram so that the clock dependency is exposed to the **Activate** compiler, thus avoiding possible artificial algebraic loops.

Consider, for example, the FMU imported in the diagram in Fig. 4 and assume the corresponding **Activate** FMU block is used in the **Activate** model as follows:



If the second input is associated only with the first clock, there shouldn't be any algebraic loops in the model because there is no dependence of the forth output on the second input when the second activation input is active. There is no direct dependence in case of the first activation either. However the compiler does not see the absence of dependence of the forth output on the second input. This information is not coded in the topology of the diagram.

To include the dependence of inputs on clocks, the diagram can be modified by conditionally blocking the inputs based on corresponding clocks. In the above case, the model can be modified as shown in Fig. 14. The *SampleHold* AU is used here to block the second input except when the block is activated via the first activation port. Since it is not activated by the second activation, the model contains no algebraic loop and can be compiled.

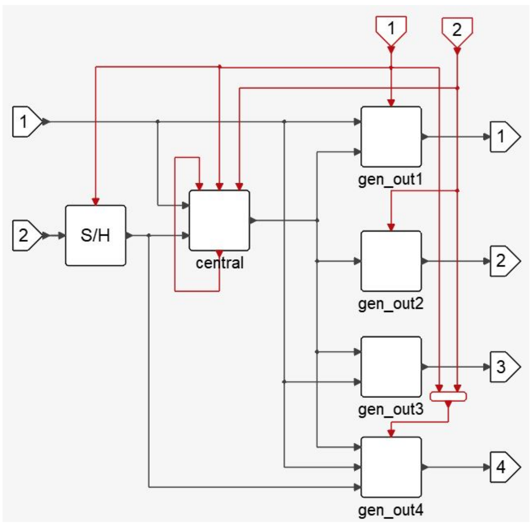


Figure 14. The *SampleHold* is used to provide the information that the second input is associated only with the first activation.

The dependence of every input on a clock can be coded in this way in the diagram.

General FMI-3.0s including multiple input and output clocks of different types can be imported by the systematic application of procedures presented above. This is done by an OML script which reads to content of the FMU model-description file (XML) and programmatically creates the required diagram.

5 Conclusion

In this paper, we showed that in general the import of a single FMU-3.0 cannot be realized by a single basic block in signal based block diagram environments such as **Scicos** and **Activate**. We examined different FMI clock types and discussed their properties and in particular their differences and similarities with the notions of activation, events and triggering in block diagram environments. We showed that there is no systematic one to one mapping of FMI clocks to block activations but the FMI clock behaviors can still be realized. The imported FMU is realized by a diagram containing multiple basic blocks depending on the type of clocks.

We presented a systematic process for creating this diagram in **Activate**. This process, which incrementally builds the imported diagram, may result in a diagram with a large number of blocks if the FMU has multiple clocks, and inputs and outputs. But the process is completely transparent to the user who sees the result as a single **Activate** block.

The import process for the user simply requires placing an FMU block, available in **Activate** palettes, inside the diagram and defining the path to the imported FMU as its parameter. The FMU block is then automatically instantiated with corresponding number of regular and activation input, output ports. It can then be used similarly to other **Activate** blocks in the construction of the **Activate** model. The corresponding internal diagram is created when the model is compiled. The diagram is not exposed to the user. It is only used internally for the compilation of the model and the construction of the compiled structure, which is used for simulation and code generation. By providing this FMU import feature, **Activate** can be used as an environment for connecting multiple FMUs (both ME and CS) to create simulation models, while respecting FMI clock semantics.

References

Altair activate, extended definitions, 2022. URL https://2021.help.altair.com/2021/activate/extended_definitions.pdf.

Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer-Verlag New York, 2010. ISBN 0-262-16209-1.

Claudio Gomes, Masoud Najafi, Torsten Sommer, Matthias Blesken, Irina Zacharias, Oliver Kotte, Pierre R. Mai, Klaus Schuch, Karl Wernersson, Christian Bertsch, Torsten Blochwitz, and Andreas Junghanns. The fmi 3.0 standard interface for clocked and scheduled simulations. *Proceedings of the 14th International Modelica Conference.*, 2021.

INRIA. URL http://www.scicos.org.

Andreas Junghanns, Torsten Blochwitz, Christian Bertsch, Torsten Sommer, Karl Wernersson, Andreas Pillekeit, Irina Zacharias, Matthias Blesken, Pierre R. Mai, Klaus Schuch, Christian Schulze, Claudio Gomes, and Masoud Najafi. The fmi 3.0 standard interface for clocked and scheduled simulations. *Proceedings of the 14th International Modelica Conference.*, 2021.

FMI Website Modelica Association, 2022. URL https://fmi-standard.org.

Ramine Nikoukhah and Sebastien Furic. Towards a full integration of modelica models in the scicos environment. *Proceedings of the 7th International Modelica Conference.*, 2009.

Ramine Nikoukhah, Masoud Najafi, and Fady Nassif. A simulation environment for efficiently mixing signal blocks and modelica components. *Proceedings of the 12th International Modelica Conference.*, 2017.

FMI-3.0 Specification, 2022. URL https://fmi-standard.org/docs/3.0.

A Snippet of the C source code of the FMU in 4.1.1

```
fmi3Status fmi3UpdateDiscreteStates(
   fmi3Instance* comp,
     /* other function arguments */
   ) {
    /* some code here */
    if (comp->clki) {
          comp->counter = comp->counter+1;
          comp->clki = 0;
    }
    return fmi3OK;
}

fmi3Status fmi3SetClock(fmi3Instance comp,
    const fmi3ValueReference vr[], size_t nvr,
    const fmi3Clock value[]) {
     /* some code here */
    if (vr[nvr-1] == 4) {
        comp->clki = value[nvr-1];
        return fmi3OK;
    }
    return fmi3Error;
}
```

B Snippet of the C source code of the FMU in 4.2.1

```
fmi3Status fmi3UpdateDiscreteStates(
   fmi3Instance* comp,
     /* other function arguments */
   ) {
        if (comp->clki) {
             comp->clki = 0;
         }
        return OK;
}

fmi3Status fmi3SetClock(fmi3Instance instance,
    const fmi3ValueReference vr[], size_t nvr,
    const fmi3Clock value[]) {
     /* some code here */
    if (vr[nvr-1] == 4) {
        comp->clki = value[nvr-1];
        return fmi3OK;
    }
    return fmi3Error;
}


 fmi3Status fmi3GetIntervalDecimal(
    fmi3Instance instance,
    const fmi3ValueReference valueReferences[],
    size_t nValueReferences,
    fmi3Float64 intervals[],
    fmi3IntervalQualifier qualifiers[])
     /* some code here */
    if (vr[nvr-1]  == 4) {
        value[nvr-1] = comp->intervalDecimal;
        return fmi3OK;
    }
    return fmi3Error;
}

fmi3Status fmi3SetIntervalDecimal(
    fmi3Instance instance,
    const fmi3ValueReference valueReferences[],
    size_t nValueReferences,
    const fmi3Float64 intervals[])  {
     /* some code here */
    if (vr[nvr-1]  == 4) {
        comp->intervalDecimal = value[nvr-1];
        return fmi3OK;
    }
    return fmi3Error;
}
```

C Snippet of the C source code of the FMU in 4.3.1

```
fmi3Status fmi3UpdateDiscreteStates(
   fmi3Instance* comp,
     /* other function arguments */
   ) {
         if (comp->clkBase) {
            comp->output = 1;
            comp->clkBase = 0;
            {
                double uu;
                uu= (comp->signal >= 1.0) ?
                 1.0 : comp->signal;
                uu = (comp->signal <= 0.0) ?
                 0.0 : comp->signal;
                comp->duty=comp->period * uu;
            }
        }

        if (comp->clkCoundown) {
            comp->output = 0;
            comp->clkCoundown = 0;
        }
        return fmi3OK;
}

fmi3Status fmi3SetClock(fmi3Instance instance,
    const fmi3ValueReference vr[], size_t nvr,
    const fmi3Clock value[]) {
     /* some code here */
    switch (vr[i]) {
    case 1: comp->clkBase = value[i];
            break;
    case 2: comp->clkCoundown = value[i];
            break;
    default:
        return fmi3Error;
    }
    return fmi3OK;
}


 fmi3Status fmi3GetIntervalDecimal(
    fmi3Instance instance,
    const fmi3ValueReference valueReferences[],
    size_t nValueReferences,
    fmi3Float64 intervals[],
    fmi3IntervalQualifier qualifiers[])
     /* some code here */
    if (vr[i] == 1) {
        value[ii] = comp->period;
    }else{
      if (vr[i] == 2) {
        qualifiers[ii]=fmi3IntervalNotYetKnown;
        if (comp->duty >= 0) {
          value[ii]=comp->duty;
          if (comp->duty==comp->PreDuty)
            qualifiers[ii]=fmi3IntervalUnchanged
          else
            qualifiers[ii]=fmi3IntervalChanged;
          comp->PreDuty=comp->duty;
          comp->duty=-1.0;
        }
      }else{
         return fmi3Error;
      }
    return fmi3OK;
}

fmi3Status fmi3SetIntervalDecimal(
    fmi3Instance instance,
    const fmi3ValueReference valueReferences[],
    size_t nValueReferences,
    const fmi3Float64 intervals[])  {
     /* some code here */
    if (vr == 1) {
        comp->period = value;
        return fmi3OK;
    }
    return fmi3Error;
}
```

[Industrial Paper] Performance Measurement and Finding Challenges in Using FMUs to Perform Scenario-Based Testing in a Cloud Environment

Katsuya Tsuzuki[1] Takashi Yamada[1] Kensuke Araki[1]
[1] dSPACE Japan K.K., Japan, {ktsuzuki, tyamada, karaki}@dspace.jp

Abstract

This paper describes the implementation of the scenario-based testing, a test method for autonomous driving software, by coupling a plant model described using MATLAB/Simulink with another plant model provided as functional mock-up unit (FMU) on a cloud platform. During the implementation of plant models into the cloud environment using the functional mock-up interface (FMI), there are problems and countermeasure challenges were identified. In addition, the impacts of integrating multiple models on simulation time by parallelizing test cases are measured.

Keywords: *Model-based Development, Cloud computing, Scenario- Based Testing*

1 Introduction

In recent years, the automotive industry has been at a turning point that is characterized by two trends. One is the trend toward electrification, and the other is intellectualization (Yamada, 2020; dSPACE Japan, 2013). The trend for intellectualization is a shift of technical development from advanced driving assistance systems (ADAS), such as automatic emergency brake (AEB) systems, to autonomous driving (AD) systems.

As vehicles become more electrified and intelligent, an in-vehicle system for controlling components in the vehicle gets more complicated. Developing components across multiple domains, including electrical, mechanical, fluid, and control systems, becomes essential, and there was an issue with interfacing components from multiple domains on a simulation platform. FMI, a standard proposed by the Modelica Association, solves this problem.

There is a solid demand for applying model-based development (MBD) to AD system development for efficient development and validation. The PEGASUS project, a public-private project funded by the German federal government, discussed what validation environment and methods for AD systems should be. As a result, a testing method using MBD tools was proposed in the project, what was called “scenario-based testing” (PEGASUS project, 2019).

Scenario-based testing is a test method characterized by automatic new test case generation based on past test results. It is helpful to efficiently perform AD system validation with a limited number of test workloads by automatically finding test parameters that should be intensively tested. Scenario-based testing was also proposed in ISO21448:2022 Road vehicle Safety of the intended functionality (SOTIF) for AD software safety functionality, which was published in June 2022 (ISO, 2022).

Compared to validating a conventional real-time control system such as an internal combustion engine control system, the number of test cases for AD system validation is still large, even when the scenario-based testing method is applied. An AD system validation environment needs to manage such a large number of tests.

People expect the adaption of a cloud computing environment to be one of the solutions to this challenge. In cloud computing, users access computation resources via the Internet. On a cloud computing platform, tests can be performed in parallel while flexibly increasing or decreasing required computing resources depending on the number / scale of the tests.

However, there are few examples using cloud computing for system validation using MBD tools, although Yamada, Araki, and Tsuzuki reported some cases at JSAE 2022 Spring and Autumn Congress (Yamada *et al.*, 2022; Araki *et al.*, 2022).

Based on this background, this paper summarizes the challenges faced in integrating multiple plant models using a FMU to realize a prototype system for AD system verification by scenario-based testing in a cloud environment. In addition, we carry out several scenario-based tests in the environment, and the simulation times are measured to compare execution time with and without the FMU. Furthermore, we parallelize the execution of several test cases, compare the impact of parallelization on the simulation time and discuss how FMU affects execution time when test cases are parallelized.

This paper is organized as follows. Section 2 describes the experimental test system and the models to be simulated. Section 3 and Section 4 describes the experiment results of the test system execution described in Section 2, and the results obtained from the experiments, respectively. Section 5 lists challenges identified during the experiments performed in Section 3 and discusses how we can solve challenges. Section 6 concludes the paper with limitations and future works.

2 Experimental Environment

An experimental environment for validating automated driving algorithms using FMU in a cloud environment was built and tested in the following three steps.

Our goal in this section is to integrate the plant model with a FMU to a cloud environment, which is suitable for scenario-based testing. Section 2.1 describes a plant model used in simulation and how it is integrated to an on-premise simulation environment on Microsoft Windows, which was commonly used to validate a conventional system so far. In section 2.2 we explain how the plant model, designed to be used on Windows, is ported to a Linux environment, which is an operating system commonly used in a cloud platform. Finally, in section 2.3 we describe how we integrate the ported plant model into a cloud simulation environment.

2.1 Plant Model on a Windows PC

The plant model used for the experiments was prepared in an on-premise Windows PC environment, which is commonly used today. The simulation model was prepared based on ASM Traffic, one of the packages in the dSPACE Automotive Simulation Models (ASM). ASM Traffic is a model designed to simulate roads, traffic, buildings, and in-vehicle sensors around vehicles, and is used for validation of AD and ADAS electric control units (ECUs).

This time, ASM in dSPACE Release 2021-A was modified in MATLAB / Simulink R2020a to remove blocks related to aerodynamics. Functionality corresponding to the removed aerodynamic block was implemented in OpenModelica 1.19.2 and output as an FMU. The output FMU conforms to FMI 2.0 co-simulation.

The ASM-modified model and the model created in OpenModelica 1.19.2 were integrated using dSPACE VEOS, a simulation platform that enables software-in-the-loop (SIL) simulation on a Windows PC.

As a controller model, which is a model that should be validated in the SIL simulation system, we used Soft ECU in this experiment, a simple controller model that comes with ASM.

Figure 1 shows a schematic of the plant model built on a Windows PC. Figure 2 shows a screenshot of dSPACE VEOS Player when the test environment is being built. It can be seen that the I/O from the controller model and the plant model are inter-connected.

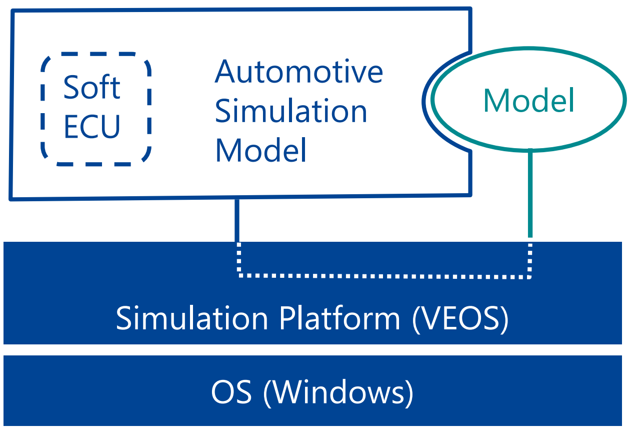


Figure 1 An overview of Windows simulation environment.

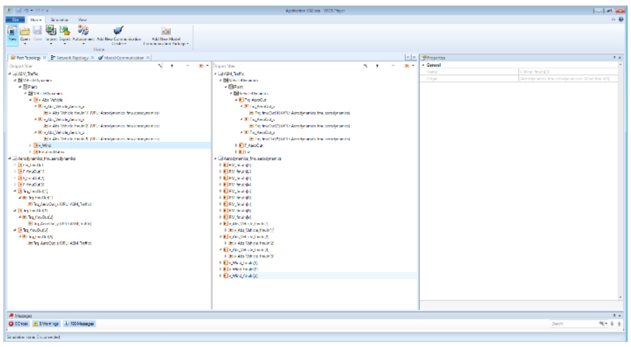


Figure 2 An example of simulation configuration in an on-premise environment with dSPACE VEOS Player.

2.2 Cross Compiling Models for Linux Environment

Linux is the most common operating system for cloud computing environments. In order to integrate the plant model in a cloud environment, it was necessary to cross-compile the model to a Linux environment.

Simulink, ASM, and VEOS support execution on Linux, and a FMU generated from OpenModelica also claims to support Linux. Cross-compilation of the plant model to a Linux environment was performed without major problems. However, since operating system is different, it was necessary to check whether there is any difference in simulation results between the Windows environment and the Linux environment.

In this paper, Ubuntu 18.04LTS was used as Linux. Figure 3 shows the plant model that was built on a Linux environment.

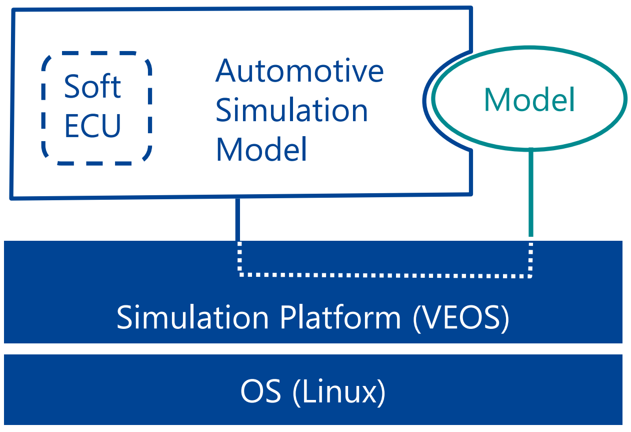


Figure 3 An overview of Linux simulation environment.

2.3 Plant Model Integration in Cloud Environment

In the cloud environment, we used dSPACE SIMPHERA version 22.4, which is an environment for SIL simulations running on various cloud platforms such as Microsoft Azure. Taking advantage of the features of cloud computing, which can secure computing resources in a scalable and flexible manner, the tests are distributed and executed in parallel on multiple computation nodes, enabling the execution of a huge number of tests, such as those for autonomous driving applications. Figure 4 illustrates an overview of simulation environment on a cloud.

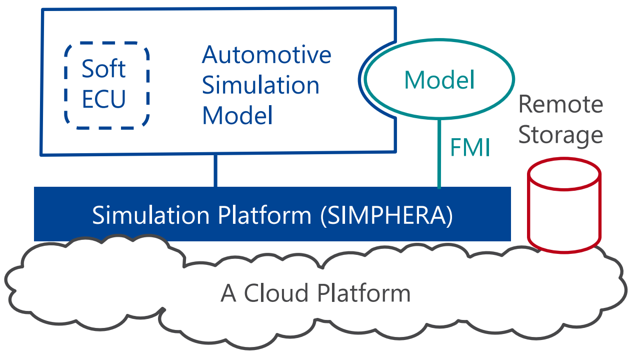


Figure 4 An overview of the simulation environment in the cloud.

SIMPHERA is developed as a Kubernetes cluster in a cloud environment. Kubernetes is an open source container orchestration software that is widely used. Container orchestration is automation of container deployment, management, scaling, and networking.

The SIMPHERA system is composed of open source software (OSS) built on a Kubernetes cluster and software from dSPACE. As OSS, MinIO, an object storage server compatible with Amazon S3 cloud storage service, is used for storage management, PosgreSQL, a relational database management system, is used as a database for input and result data, and Keycloak, personal authentication and access management software, is used for WebUI login management.

On the other hand, dSPACE software includes SIMPHERA execution agent, which is equivalent to an application execution unit that executes jobs for each parameter to compute scenario-based tests in parallel.

dSPACE VEOS runs as simulation software for scenario-based testing in SIMPHERA execution agent. The SIMPHERA system configuration diagram is shown in Figure 5.

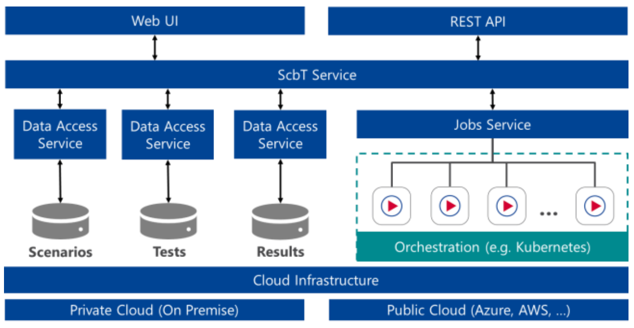


Figure 5 System overview of dSPACE SIMPHERA

In this study, the private cloud environment shown in Table 1 was prepared and SIMPHERA was built on this private cloud environment.

Table 1 Specifications of Private Cloud.

	loud Specification
CPU	28 logical cores
Memory	48GB
Storage	512GB
OS	Ubuntu 20.04LTS

3 Scenario Definition

Using the experimental environment in which the plant model was implemented in SIMPERA on the cloud environment as described in Section 2.3, simulations in the scenario-based test were performed with different parallelism levels for models with and without FMU, and the relationship between parallelism and execution time was measured.

The scenario used in the scenario-based test was the United Nations Economic Commission for Europe (UNECE) Automatic Lane Keeping System (ALKS) Cut-In Driver. This is a validation scenario for the

ALKS when another vehicle, traveling in adjacent lane, changes to the own lane.

In Figure 6, an illustration of the UNECE ALKS Cut-In Driver scenario and the definition of each parameter are shown.

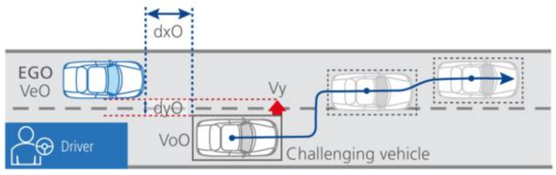


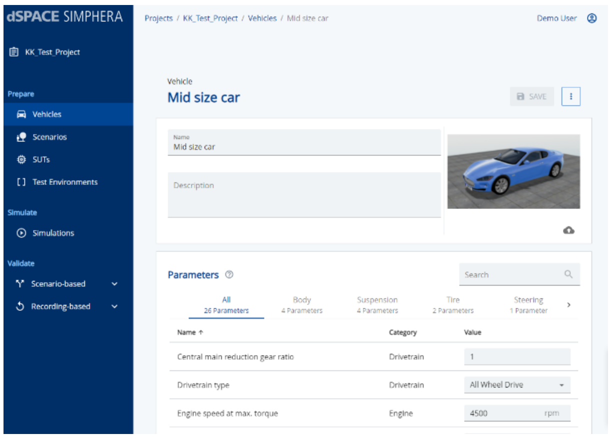
Figure 6 UNECE ALKS Cut-In Driver Scenario and Parameters.

In this experiment, the speed of the ego vehicle (Ve0) was varied from 50 km/h to 51 km/h for all two cases of testing. The scenario parameters used in the experiment are shown in Table 2.

Table 2 Scenario, parameters.

Scenario	UNECE ALKS Cut-In Driver
Parameter variation type	Cross Variation
dx0	11 m (Fix)
Ve0	50 ~ 51 km/h (Δ = 1 km/h)
Vy	3 m/s (Fix)

In SIMPHERA, the WebGUI shown in Figure 7 was used to implement visually easy-to-understand settings for the above parameters.



Vehicle
Mid size car

Name: Mid size car

Description

Parameters

Name ↑	Category	Value
Central main reduction gear ratio	Drivetrain	1
Drivetrain type	Drivetrain	All Wheel Drive
Engine speed at max. torque	Engine	4500 rpm

Figure 7 An example of simulation configuration in a cloud environment with dSPACE SIMPHERA.

4 Experimental Results

Simulations were performed on SIMPHERA for a model linking ASM and FMU and a model of ASM alone, changing the parallelism from one to two parallel jobs for all two cases of scenario-based testing, respectively, and the execution times were measured.

Parallelism is the maximum number of jobs that can be executed simultaneously on SIMPHERA and is one of the parameters that can be given to the execution environment by the user.

When the parallelism level was set to 1, the execution time of the ASM stand-alone model was 68.7 [sec], while the execution time of the model linking the ASM and FMU was 70 [sec]. When the parallelism was set to 2, the execution times were 35.7[sec] and 36.3[sec], respectively. Figure 8 shows the experimental results.

The results show that the execution times of the ASM stand-alone model and the model linking the ASM and FMU were almost the same.

In addition, when the parallelism level was changed from 1 to 2, the computation time was inversely proportional to the parallelism level and almost halved. Experimental results are shown in Figure 8.

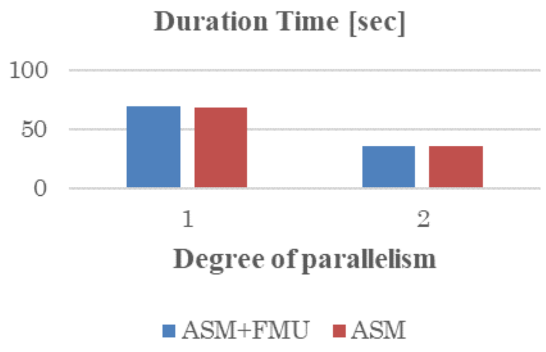


Figure 8 Duration time depending on the degree of parallelism

In the case of the model in which ASM and FMU are linked, the computation time may increase compared to the ASM stand-alone model due to overhead caused by communication between cores and overhead caused by communication between ASM and FMU, etc.

However, since the computation time for the overhead has not increased even if the parallelism level is increased The overhead due to the parallelism and communication between the ASM and the FMU are confirmed as not increased as the overhead due to the communication between the ASM and the FMU.

The reasons for the almost no difference in computation time between the model linking ASM and FMU and ASM can be attributed to the fact that the FMU model created and used for this study is stand-alone and lightweight, and that the overhead in linking the ASM and FMU models on dSPACE VEOS used for the simulation was small. The reasons for the difference are considered to be the following.

5 Challenges Identified during the Experiment

During the experimental environments and its execution, we identified some challenges. In this section we attempt to list up the challenges faced to the trial.

5.1 Challenges on Operating System Migration

Prior to the plant model construction on a Windows PC described in section 2.1, we had created an FMU using OpenModelica 1.18.1. However, when the FMU output from OpenModelica 1.18.1 was to be run in a Linux environment, additional software libraries specific to the Linux environment had to be installed.

The software libraries that needed to be installed additionally are, for example, liblapack.so. Other software libraries that were additionally required are shown in Talbe3.

Table 3 External libraries referenced by FMU.

External Libraries
liblapack.so.3
libblas.so.3
libgfortran.so.5
libquadmath.so.0

However, this issue has been resolved in OpenModelica 1.19.2, and additional libraries are pre-installed in the FMU. In this experimental environment, there was no need to install additional software libraries. In an on-premise environment, the installation of libraries may not be so much of a problem because the person who performs the simulation often has administrative privileges, but in a cloud environment, where many users are supposed to use the system, it is rare for the user to have administrative privileges, and it is difficult to install additional software libraries. Additional installation of software libraries is difficult and should be done with caution.

In addition, non-compatibility of measurement and calibration software with Linux may also be an issue. Software for measurement and on-line calibration of simulation results after the simulation environment has been built also requires attention. Although many companies provide measurement and calibration software that can be used on-premise, most of them are Windows versions, and the same software may not be available for Linux and cloud environments. On the other hand, some software, such as dSPACE ControlDesk, is designed to work with simulation environments in the cloud.

5.2 Challenges arising from the use of FMUs

In this study environment construction, we faced some issues specific to the use of FMI/FMU that are likely to occur regardless of whether the system is used in the cloud or not. Many of these are described in the FMI guidelines published by the Society of Automotive Engineers of Japan (JSAE), but are listed again in this section. The main issues faced were that "the FMU end time setting is contained within the FMU" and "a bus cannot be specified as the interface of the FMU.

First, the FMU end time settings are contained in the FMU, and any attempt to modify them requires direct editing of the files contained in the FMU. Otherwise, the operation of the FMU may be stopped unintentionally by the user, independent of the operation on the simulation platform.

Next, it is noted that buses cannot be specified for FMU interfaces; only arrays defined by a scalar value or a set of scalar values of the same type can be used to describe interfaces between FMUs and their externals. The interface between the FMU and the external model can use variables of various types, preferably in the form of structures in high-level programming languages or buses in Simulink.

However, the FMI2.0 specification used in this study does not support the connection between FMU and external models using these types and buses, and it is necessary to separate signal lines that are grouped into buses and connect them one by one. This leads to an increase in man-hours required to connect models, especially when connecting huge models. Regarding this issue, it is highly likely that FMI3.0 solves this problem; future research on experiments using FMI3.0 is expected.

6 Concluding Remarks

In this study, we performed scenario-based test jobs for validating autonomous driving algorithms using FMU in a cloud environment with different degrees of parallelism, evaluated the performance of parallel computation, and compared the computation time with and without model linkage between ASM and FMU.

The results show that the calculation speed increases with the degree of parallelism even when the model linking ASM and FMU is used. In addition, we examined the point where there is no significant difference in computation time between ASM and FMU-linked models and ASM and the reasons for this difference.

6.1 Future Works

In this study, we evaluated the performance of scenario-based tests for validating automated driving algorithms in a private cloud environment by varying the parallelism of the jobs. Since we have not yet conducted parallel computation in a public cloud environment, where computing resources can be flexibly changed, it remains to be seen whether the trend of results differs between the two environments and whether the bottleneck changes.

Also, we evaluated the performance of the scenario-based test for validating the automatic driving algorithm using FMUs when the test was run in parallel.

We compared the computation time with and without FMU collaboration. In this study we used a single FMU and a small-scale model. What will happen when multiple or large-scale FMUs are used, and how we dealing with them will be the subject of future research.

References

K. Araki, K. Tsuzuki, and T. Yamada (2022): Simulation Performance for Scenario-based Testing in a Cloud Environment, *2022 JSAE Annual Congress Autumn* (written in Japanese).

dSPACE Japan (2013): Model-based Development, Nikkei BP (written in Japanese).

ISO (2022): ISO21448:2022 Road vehicles -Safety of the intended functionality, available at https://www.iso.org/standard/77490.html, accessed on 19 Sept 2022.

PEGASUS Project (2019): PEGASUS Method – an Overview, available at https://www.pegasusprojekt.de/en/home, accessed on 19 Sept 2022.

T. Yamada (2020): Model-based Development, *Technology Roadmap 2021-2030 Automotive and Energy*, Nikkei BP, p. p. 236 – 239 (written in Japanese).

T. Yamada, K. Araki, and K. Tsuzuki (2022): Challenges and Countermeasures in Using FMUs to Perform Scenario-based Testing, *2022 JSAE Annual Congress Spring* (written in Japanese).

Simulation Scheduling of Variable-Structure Systems in OpenModelica

Rahul Paknikar[1] Nikhil Sharma[2] Priyam Nayak[2] Kannan Moudgalya[2] Bhaskaran Raman[1]

[1]Department of Computer Science and Engineering
[2]Department of Chemical Engineering

Indian Institute of Technology Bombay, Mumbai, India
`e-mail: kannan@iitb.ac.in`

Abstract

We propose and implement a generic scheduling framework for OpenModelica to eliminate the simulation code corresponding to inactive components in a system-level model. This framework allows the model developer to auto-generate models corresponding to the discrete behavior of the underlying system, and then schedule their simulations. It also provides a `Scheduler` library in the Modelica language to help the model developer easily generate the schedule. The benefit of this approach is demonstrated with and without real-time simulations of a batch distillation system. The proposed approach also helps implement a sequential modular simulation to arrive at initial guesses for flowsheets, whose equations can then be solved simultaneously using the standard, equation-oriented, approach of Modelica.

Keywords: Schedule, OPC UA, OpenModelica, Batch distillation, Steady-state, Variable-structure modeling, Sequential modular simulation

1 Introduction

The ability to model discrete behavior is an important requirement in industrial systems, even in continuous plants, such as refineries. The reason is that these plants also need to be cold started and shut down in case of emergencies. Unless startup and shutdown procedures are clearly understood, one cannot even take the lab-scale discoveries to the plant level for manufacturing. Hence, the ability to correctly model discrete behavior is an important requirement.

System-level modeling of a huge and complex system has become a common methodology for system engineering design (Pop et al. 2019). Such modeling can include several hierarchies of subsystems and a large number of components. Most general-purpose simulators broadly fall into one of the following three categories based on the way they process the information in the system under simulation: discrete event, continuous, and hybrid, i.e., a combination of the continuous and discrete-event simulation.

The simulators treat the system-level modeling as a single unit for efficient compilation to generate the simulation code and run the simulation itself. That is, the system-level modeling design is no longer preserved, and all its hierarchy and components are compiled and simulated entirely. A similar software framework is observed as a part of the typical process of translation and execution of a system-level model in most of the Modelica tools (Fritzson 2014), including the open-source OpenModelica compiler (Pop et al. 2019). Such frameworks can lead to the compilation and simulation of inactive subsystems and components, leading to some difficulties.

In order to address the main issue of constantly running simulation code that is not required, we propose a generic scheduling framework. It describes the discrete behavior of the system to be modeled in terms of a schedule. This framework also helps schedule their simulations (Section 2). Section 3 discusses the related work within and outside the Modelica context. In Section 4, we present a prototype developed for this framework that leverages the OpenModelica Simulation Environment (Fritzson et al. 2020) along with a Modelica library called `Scheduler`. Sections 5 and 6 demonstrate the benefits of the proposed approach in a batch distillation column and process flowsheeting, respectively. The last section is devoted to the conclusion and future work. In our work, the system-level modeling primarily involves variable-structure systems.

2 Scheduling Framework

To understand the motivation behind this work, let us consider the following example. Consider the cold startup of an overflow tank connected to a pumped flow tank, as shown in Figure 1. Initially, there is no liquid in both tanks. Until the liquid level rises in the first tank to the level of the outflow line, there can be no liquid in the second tank. Without the liquid, starting the pump will damage it. In addition, suppose that we have to calculate the liquid density in the second tank. Unless suitable precautions are taken, there could be difficulties in the calculation because of the zero volume of the liquid. There are three different ways to handle this situation:

1. Noticing zero volume in the tank, do not calculate the density nor implement any calculation that requires density.

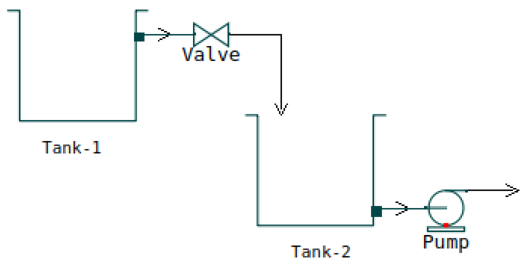


Figure 1. System-level model of two interacting tanks

2. Assume that there is always a small amount of liquid present in the second tank and calculate the density.
3. Do not even attempt to simulate the second tank until the liquid starts coming from the first tank.

The first approach increases the load on the model developer. They must anticipate all the different ways the modeling assumptions can be violated and take corrective actions. The second approach results in erroneous calculations, however small they may be.

The above two approaches necessitate solving all subsystems, including the inactive ones. This approach is not acceptable if the objective is to do minimal modeling, such as arriving at the initial guess of a large number of model equations by solving a few equations at a time, as explained in Section 5. The advantage of the minimal modeling approach is that it does not require setting up initial guesses for inactive models. The third approach does not have the difficulties mentioned above, provided we have the capability to simulate at the correct time. In this work, we attempt to create this capability to simulate only the required models.

2.1 Schedule for System-Level Modeling

The active components or subsystems of a system can be effectively determined through its discrete behavior. The same can be modeled through discrete events in terms of a schedule. Thus, the model developer constructs a schedule of dependent tasks and events for system-level modeling. An event indicates a condition based only on the currently scheduled task. It specifies the next task to be scheduled when the event triggers. I.e., the condition is satisfied. Note that the condition can involve just the simulation time so that it does not constrain the variable-structure model to generate events from its state. A task can include a set of components, subsystems, or several systems. It will be simulated only when activated based on the target of the event associated with the currently scheduled task. From the above description, the following are the observed properties of any schedule for system-level modeling:

- An event is associated with only one task and specifies only one task to be scheduled next. A task can have several events associated with it, which can schedule only one of the several next tasks depending on the order of the events being triggered.

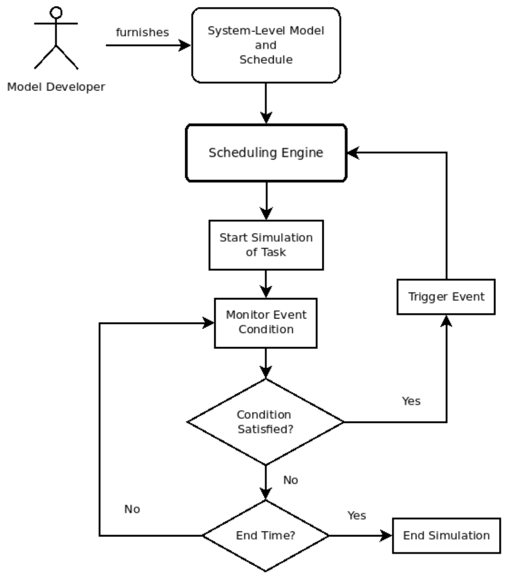


Figure 2. Generic Workflow for Simulation Scheduling

- The schedule turns out to be a directed and connected graph with an alternate sequence of tasks and events. Each node in such a graph represents the pair of a task and its associated events. The worst-case schedule can be a complete graph.
- The graph can have backward and forward edges with respect to a node. However, there is no self-loop back to a node as it has no practical significance and can be handled within the node itself.
- This graph is compliant with control workflow patterns (Russell, Van Der Aalst, and Ter Hofstede 2016) such that only one node gets activated at any point of simulation time, and the directionality between the nodes is preserved. As analyzed from the first property, the possible control patterns include unstructured loop, sequence, exclusive choice, and simple merge. These patterns are considered for implementation in Section 3.

2.2 Simulation Scheduling

Figure 2 shows the generic simulation scheduling once the model developer provides the system-level model and its schedule. The scheduling engine is an intermediate layer between the system-level modeling and the underlying simulation tool. The engine parses the model and simulates only the first task based on the schedule provided. The events associated with the task are monitored to evaluate their conditions. If any of them is satisfied, then the corresponding event gets triggered. The engine then simulates the next task in the schedule from the point in simulation time where the previous task left. The process repeats until the end of the entire simulation time.

The above framework descriptions and schedule properties divide the system-level modeling into discrete parts of the components or subsystems. Thus, it also divides the single continuous simulation into multiple but efficient smaller and faster simulations of tasks. The efficiency is achieved in terms of smaller executable code size and lesser memory requirements for each task when compared to the entire simulation done with existing frameworks. The same is noticeable from the demonstration discussed in Section 6. Otherwise, it can waste CPU cycles as the data can keep on shuffling between the CPU cache and the RAM (Pop et al. 2019). As only one task will be simulated at any given point of the simulation time, the goal is thus achieved by not simulating the rest of the inactive tasks. Therefore, the unnecessary running of the simulations for the inactive tasks is prevented, which anyway will not impact the simulation of the current task.

The framework is independent of the underlying simulation tool and the modeling language it uses. As a result, it also permits real-time and interactive simulations and does not require any rework of the existing simulation tools. Note that the interactive simulation here indicates that one can interact with the model during its simulation by monitoring and optionally modifying the state of the model as per the requirement.

3 Related Work

There is no native support, nor is there any Modelica tool that provides a generalized extension or framework of the type described here, suitable for all domains (Casella 2019; Jack 2020). Nevertheless, there are attempts at variable-structure modeling similar to the proposed simulation scheduling workflow (Briese 2018; Mehlhase et al. 2014; Stüber 2017). Stüber (2017) further discusses several implementations and their drawbacks. They use specialized forms of the proposed generic simulation scheduling workflow suitable for their applications. One approach common to all of them involves either the reimplementation of their existing models or manual work for generating models with different structures and exploiting the functionalities of proprietary tools. Also, none of these related works exhibits real-time and interactive simulation capability. Apart from these application-specific implementations, there is a need for sequential modular simulation in OpenModelica (Casella 2021). It also indicates that no such related generalized work has been done for OpenModelica.

Outside the Modelica context, the following are the prior work that attempts to have similar modeling capabilities that we have proposed but simulate as per the existing framework.

The special-purpose and open-source Ngspice circuit simulator (Vogt et al. 2021) partially avoids running simulation code that is not required. The components in the circuit are mapped to a C function through its XSPICE extension. As a result, the C function gets invoked only when the signal reaches the corresponding component. However, this behavior is applicable only for the digital components, while the analog simulation of all components is still running even though they may not be required.

A proprietary and general-purpose simulator, GoldSim, has conditional containers similar to the conditional task scheduling in our engine. However, it still keeps on constantly running the code in idle mode for those components that are not required (GoldSim Technology Group 2022). Similar functionality is observed with the general-purpose AnyLogic simulation software (The AnyLogic Company 2022) and gPROMS (Process Systems Enterprise Ltd. 2004), a special-purpose simulator focused on chemical processes. Another proprietary but discrete-event simulator, FlexSim, has conditional task functionality. However, again, the components still keep running and remain idle as described in their tutorial (FlexSim Software Products, Inc. 2022).

Another application of variable-structure modeling, which has received great attention, is the simulation of the reconfigurable manufacturing system (RMS) that implies a change in the factory structure. K et al. (2019) and Herps et al. (2022) demonstrate the simulation of their proposed manufacturing processes using FlexSim and AnyLogic respectively. However, as mentioned earlier, both software keeps executing the conditional elements even if they are redundant. Kahloul, Bourekkache, and Djouani (2016) use reconfigurable object Petri nets to model and simulate RMS. It has system-level nets that involve fire and transform transitions and a set of morphisms. It allows changing between the object net markings and structures corresponding to each configuration. Although this modeling is similar to the work described here, their entire RMS representation is simulated as a single model. It thus loses the benefit of an already discretized model and the scope to avoid running redundant simulation code.

The FMI standard in terms of System Structure and Parameterization (Modelica Association Project SSP 2019) and Distributed Co-Simulation Protocol (Modelica Association Project DCP 2019) may possibly be exploited to achieve the goal of our work. These co-simulation standards, which are out of the scope of this work, shift efforts from simulation run-time (scheduling framework) to simulation configuration time. However, as the discrete behavior of the system can be known at run-time, one has to additionally anticipate different scenarios with variable structures and generate them only at configuration time.

4 Implementation of Scheduling Framework for OpenModelica

The primary goal of our work is to run only the simulation code that is required. In the simulation context and to the model developer, it indicates that the subsystems or components are to be simulated only when activated, i.e., lazily simulated. As mentioned earlier, our implementation is based on the Modelica language. The systems are modeled using equations in this language and require all equations to be solved simultaneously during the entire simulation. The goal, hence, boils down to preventing unnecessary solving of the equations governing the behavior

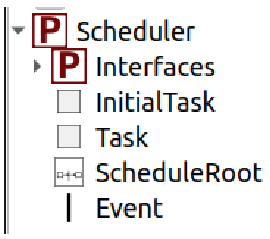
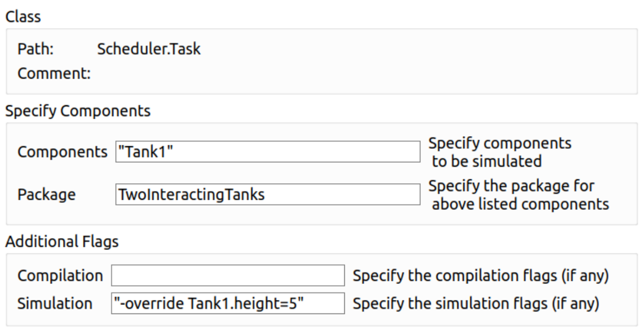


Figure 3. Structure of the `Scheduler` library shown in OMEdit - OpenModelica Connection Editor


Class
Path: Scheduler.Task
Comment:

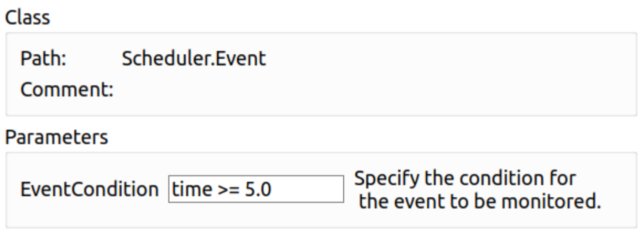
Specify Components

Components	"Tank1"	Specify components to be simulated
Package	TwoInteractingTanks	Specify the package for above listed components

Additional Flags

Compilation		Specify the compilation flags (if any)
Simulation	"-override Tank1.height=5"	Specify the simulation flags (if any)

(a) Task Interface


Class
Path: Scheduler.Event
Comment:

Parameters

EventCondition	time >= 5.0	Specify the condition for the event to be monitored.

(b) Event Interface

Figure 4. User interfaces for `Scheduler` library's important blocks as shown in OMEdit - OpenModelica Connection Editor

of inactive components or subsystems.

The overall prototype implementation has two parts. The first one is only the creation of the schedule for the variable-structure model using the `Scheduler` library without any need to re-implement the model. The second one is the simulation scheduling using an engine that leverages the OpenModelica Simulation Environment.

4.1 `Scheduler` - Modelica Library

For the model developer to construct the schedule consisting of tasks and events, we have developed a library, called `Scheduler`, in the Modelica language itself. Figure 3 shows its structure in OMEdit, the OpenModelica's connection editor. The `Task` and `Event` classes correspond to the schedule's task and event discussed in Section 2, and their user interfaces are shown in Figure 4. The `InitialTask` class is the same as the `Task` class, except that it indicates the start of the schedule. The `ScheduleRoot` class requires the model developer to specify the top-level model in the hierarchy of system-level modeling and the general simulation parameters applicable to all the tasks in the schedule. The schedule must have an instance of this class. The `Interfaces` sub-package provides the desired abstraction to the above-mentioned classes and is not meant to be used directly.

The classes in the `Scheduler` library themselves are available as components, which the model developer can drag and drop in OMEdit, and also write the Modelica code to connect the tasks and events. As shown in Figure 4, the model developer has to provide a list of components or subsystems for each task along with the package name to find these components. There is also a provision to specify compilation and simulation flags in addition to those mentioned in the instance of `ScheduleRoot` class. These flags will be applicable only to that task. The event condition needs to be specified for all the events in the schedule. The schedule can be created within the top-level model, i.e., in the same file as the top-level model or outside it as a separate file having the name as `Schedule` within the same package.

4.2 Scheduling Engine

As discussed earlier, there is no native support in Modelica for scheduling. So, the implementation of the scheduling engine needs to be outside the Modelica context, which we have done in Python language. The scheduling engine requires only the system-level model and the schedule to be provided by the model developer. It is an additional layer between the user's system-level model and OpenModelica. That is, it runs on top of the OpenModelica compiler (OMC) in an interactive mode and is loosely coupled to OMC. It leverages OMC's ZeroMQ communication interface through OMPython to take over the typical compilation and simulation process in OpenModelica. OMPython is a part of the OpenModelica Simulation Environment and acts as a Python interface to communicate with OMC. The provided schedule is parsed independently of OpenModelica per the scheduling framework. It then distributes the system-level model into several sets of active subsystems or components preserving the connections between them, where each set corresponds to a task in the schedule. As a result, it automatically generates all the models corresponding to each task on behalf of the model developer.

The scheduling technique used by the engine is similar to the next-event scheduling used commonly in the discrete-event simulation. That is, when each event is set up, it creates (schedules) the next procedure (task and event). Following this scheme, only the initial task to be run first is compiled and then simulated. The events associated with the initial task are monitored to evaluate the corresponding conditions. The next task in the schedule is activated when one of these events gets triggered. If the event condition gets satisfied immediately upon the start of task simulation, then the task simulation is terminated immediately without consuming any further simulation steps and the next task gets scheduled. The next task is compiled and then simulated by transferring the end results of the previous task to the next. This transfer of end results ensures the continuity of the state from where the previous task left. It acts as initialization conditions for the next task, by overriding the default ones provided by the model developer within the model itself. If any other additional information is not present in the previous or the next task, then the model developer can still provide it using suitable OpenModelica simulation flags applicable to that task. An example of specifying additional information to initialize the tank height with 5 units is shown in Figure 4 (a). The above entire process is repeated until the end of the entire simulation time.

The monitoring of events and evaluation of the associated conditions is achieved through the OPC UA server implementation in OpenModelica (henceforth referred to as the server). OPC UA is an interoperability standard in industrial applications. The server is suitable for interactive simulation in real-time as well (Kumar et al. 2021). The next section also illustrates the same using the scheduling framework. Note that the interactive simulation through OPC UA does not involve any kind of visual interface or animation, and is out of the scope of the work described here. The scheduling engine acts as the OPC UA client and simulates each task with an embedded server. The event monitoring process leverages the publish-subscribe model of the server. The process variables (PV) and the manipulated variables (MV) in the event conditions are subscribed for notifications by the scheduling engine. As a result, simulation time is saved by not polling continuously for the changes in PV and MV over the OPC UA client-server configuration.

As observed from the above engine implementation, the compilation and simulation of each task are done just in time. That is, the tasks are compiled and simulated only when they are required. Note that there are no repeated compilations of the same tasks. The very first compiled tasks are reused again with a different context whenever required. It is possible to manually create the model corresponding to each task, compile them and then use them for manual scheduling. However, it is not feasible and error-prone when the complexity of the system and schedule scales up. Thus, the automatic distribution of the system-level model and simulation scheduling is more efficient than the manual work for the model developer. The importance of the problem attempted in this work and the usage of the proposed framework through the above implementation is illustrated through two engineering examples, to be presented next.

5 Batch Distillation System

In this section, we explain how the proposed framework helps reduce the computations in the operation of the batch distillation column (Figure 5) studied by Kumar et al. (2021) using an OpenModelica OPC UA client-server configuration. In this example, a feed stream containing three chemicals is separated into pure components. Sharma, Moudgalya, and Shah (2021) operate the opening and closing of the valves through the StateGraph library

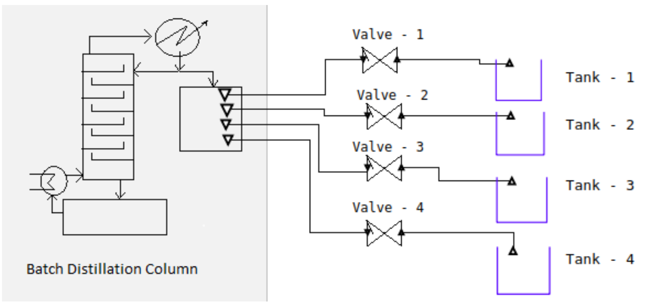


Figure 5. Batch distillation system for ternary mixture with product and slop cuts

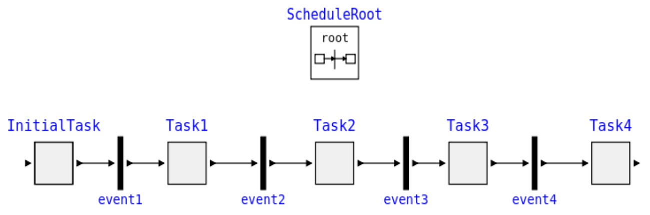
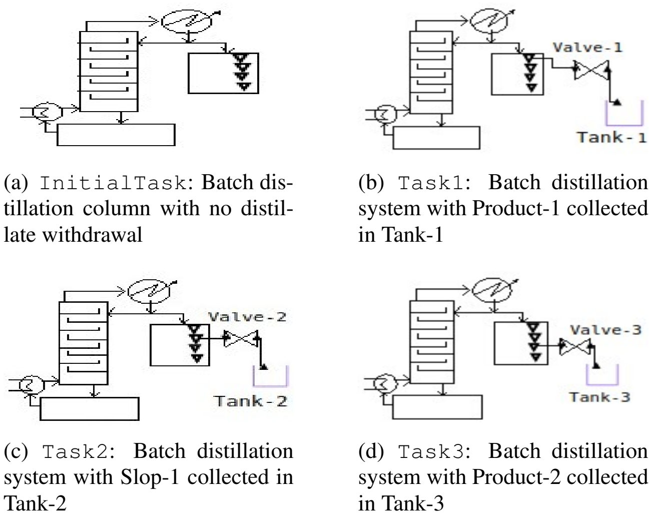


Figure 6. Schedule of the batch distillation system using `Scheduler` library



(a) `InitialTask`: Batch distillation column with no distillate withdrawal

(b) `Task1`: Batch distillation system with Product-1 collected in Tank-1

(c) `Task2`: Batch distillation system with Slop-1 collected in Tank-2

(d) `Task3`: Batch distillation system with Product-2 collected in Tank-3

Figure 7. Task-wise sequence of active components of batch distillation system simulated through the scheduling framework

in OpenModelica, which obviates the need for event constructs such as `if-else` and `when` statements. All components of the batch distillation system are simulated simultaneously in both approaches, irrespective of whether they are active or not.

The batch distillation system with the product and slop scheduling in different tanks is modeled through the `Scheduler` library. The inlet valves of the product and slop tanks are controlled according to the purity levels obtained in the distillate. This depends on the mole fraction of components in the distillate and reboiler, respectively. Figure 6 shows the sequence of events and tasks to be simulated on the occurrence of a particular time or state event or both. This schedule is created in OMEdit by drag and drop of five instances of `Task` class, four instances of `Event` class, and one mandatory instance of

ScheduleRoot class. It is defined outside the top-level model of the batch distillation system (Figure 5) in a separate file but within the same package. Figure 7 shows the sequence of subsystems corresponding to the simulated tasks as per the schedule shown in Figure 6.

Listing 1. Illustrative Modelica code for top-level model of batch distillation system

```
model BatchDistillation
  DistillationColumn Column;
  Valve Valve1, Valve2, Valve3, Valve4;
  Tank Tank1, Tank2, Tank3, Tank4;
equation
  connect(Column.outflowProduct1, Valve1.
      inflow);
  connect(Valve1.outflow, Tank1.inflow);
  connect(Column.outflowSlop1, Valve2.
      inflow);
  connect(Valve2.outflow, Tank2.inflow);
  connect(Column.outflowProduct2, Valve3.
      inflow);
  connect(Valve3.outflow, Tank3.inflow);
  connect(Column.outflowSlop2, Valve4.
      inflow);
  connect(Valve4.outflow, Tank4.inflow);
end BatchDistillation;
```

Listing 2. Illustrative Modelica code for InitialTask

```
model BatchDistillation
  DistillationColumn Column;
end BatchDistillation;
```

Initially, the batch distillation system is operated at total reflux with no distillate taken out of the system. In this condition, the valves and tanks connected to the batch distillation column are idle. So, the initial task is operated only with the batch distillation column present, and all other components are excluded as shown in Figure 7 (a). This corresponds to the InitialTask of the schedule in row 1 of Table 1. The scheduling engine parses this task by removing all of the components and their connect equations, except the batch distillation column, from the Modelica code shown in Listing 1. It communicates with OMC interactively through OMPython to achieve the same. Listing 2 shows the illustrative Modelica code for the resultant model after parsing InitialTask.

The communication involves sending commands, that invoke the appropriate scripting API available in OpenModelica, and receiving the status of the command. As mentioned earlier, the communication is done in the form of client-server configuration over the ZeroMQ interface, where the scheduling engine acts as the client and OMC as the server. In this way, the scheduling engine automatically generates the model for InitialTask without any manual intervention of the model developer. This model is kept in memory until it is compiled, but it can also be dumped into a separate file for the model developer's reference.

Once the model corresponding to InitialTask is generated, it is compiled and its simulation is started by embedding an OPC UA server with its executable. The same is already demonstrated by Kumar et al. (2021). This embedded OPC UA server allows the scheduling engine to monitor the first event (event1). As soon as the event condition being satisfied is detected, the scheduling engine sends another command to gracefully terminate the simulation. It, in turn, saves InitialTask's current state in the form of a result file. It can be visualized in OMEdit by selecting the desired variables in its interface. The end results from this result file act as the initialization condition for Task1. The model generation, compilation, and simulation for the rest of the tasks are done in a similar manner as InitialTask.

When the desired purity of the lighter component is achieved in the distillate, Valve-1 is opened, and the product is collected in Tank-1. Accordingly, the next task is scheduled with the corresponding event condition that activates the Valve-1 and Tank-1 and is shown in Figure 7 (b). This corresponds to Task1 in Table 1. The initialization of this task is done with the transfer of end results (state) from InitialTask. Note that the initialization condition for the distillation column gets overridden here as only its state information is present in the simulation of InitialTask. Thus, the initialization condition for Tank-1 and Valve-1 falls back to the default one already described within the model itself.

In the next task of the schedule, as the purity of the first component decreases below the desired level in the distillate, Valve-1 is closed. So, both Valve-1 and Tank-1 no longer need to be solved in the simulation and hence are removed. Simultaneously, Valve-2 is opened, and distillate goes to Tank-2 as slop cut, an undesirable product, and is shown in Figure 7 (c). This corresponds to Task2 in Table 1. Similar to the previous case, the initialization for this task is done only for the distillation column with the state from Task1.

When the desired purity of the second component is achieved in the distillate, it is collected in Tank-3. Correspondingly, Valve-3 is opened, and Valve-2 and Tank-2 are removed from the simulation as shown in Figure 7 (c). This corresponds to Task3 in Table 1. Similar to the previous cases, the initialization for this task is done only for the distillation column with state from Task2.

When the purity level of the second component decreases below the desired level in the distillate, the next step is collecting the slop in Tank-4 until the third component reaches the desired purity in the reboiler. As seen in Figure 9 the purity of the third component reaches desired level well before the distillate is collected in Tank-4. Hence, Valve-4 is never opened, and so Task4 is never compiled and simulated.

Figure 8 shows the moles of product and slops in the batch distillation system. Figure 9 shows the mole fractions of component-1 and component-2 in distillate, and component-3 in reboiler. These results obtained using the scheduling framework are the same when performed with and without real-time simulations. The setup for the real-time simulation and the above results are identical to

Table 1. Task-wise distribution of units for the batch distillation system simulated through the scheduling framework

Task (as per the schedule)	**Active Units** (simulated)	**Inactive Units** (not simulated)	**Corres-ponding Figure**
Initial Task	Distillation column	All valves and tanks	Figure 7 (a)
Task1	Distillation column, Valve-1 and Tank-1	Valve-2,3,4 and Tank-2,3,4	Figure 7 (b)
Task2	Distillation column, Valve-2 and Tank-2	Valve-1,3,4 and Tank-1,3,4	Figure 7 (c)
Task3	Distillation column, Valve-3 and Tank-3	Valve-1,2,4 and Tank-1,2,4	Figure 7 (d)

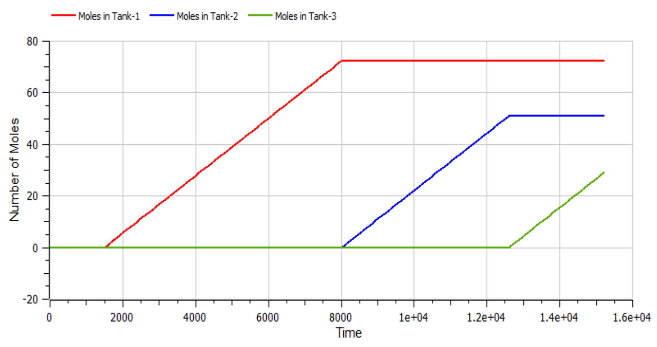


Figure 8. Moles of distillate collected in tanks

those described by Kumar et al. (2021) using Raspberry Pi performing real-time simulation in OpenModelica. The only difference in the setup is that the controller here is within the batch distillation model instead of Raspberry Pi. Furthermore, the results here are in agreement with the simulation results done using the StateGraph library by Sharma, Moudgalya, and Shah (2021).

Table 2 compares the number of equations solved by the previous methods with the current scheduling framework for each task. Instead of solving the entire flowsheet containing the distillation column, four valves, and four

Table 2. Number of equations solved per task (as given by OpenModelica) compared with previous methods

Variable-Structure	**Number of Equations Solved**		
Models	**OPC UA**	**StateGraph library**	**Scheduling framework**
InitialTask	440	440	**260**
Task1	440	440	**268**
Task2	440	440	**268**
Task3	440	440	**268**
Entire Batch Distillation System	440	440	440

tanks resulting in 440 equations, the scheduling framework solves the active units at a particular event and removes the inactive units from the simulation of the batch distillation system. All the modeling equations are solved using DASSL (Brenan, Campbell, and Petzold 1996) in OpenModelica.

6 Steady-state solution of a flowsheet through a sequence of calculations

This application is concerned with finding the steady-state solution to chemical engineering flowsheets described by a large number of equations. OpenModelica has the capability to collect the equations from different parts of a flowsheet and solve them simultaneously. It is an important capability, as design problems can be solved easily in this framework. It is also suitable for dynamic simulations. Unfortunately, having to solve a large number of equations gives rise to some difficulties. As these are generally nonlinear equations, initial guesses are required to solve them. Setting up the initial guess itself is difficult for a large number of equations, let alone converging to the steady-state solution.

Let us consider the Methanation flowsheet (Reklaitis 1983) given in Figure 10. In this system, the synthesis gas, which is a mixture of CO, H_2, and a small amount of CH_4, is converted to a higher content of Methane. The reaction taking place is:

$$CO + 3H_2 \rightarrow CH_4 + H_2O$$

The feed stream at 93.3°C and the recycle stream are fed to the mixer, followed by an adiabatic reactor with an

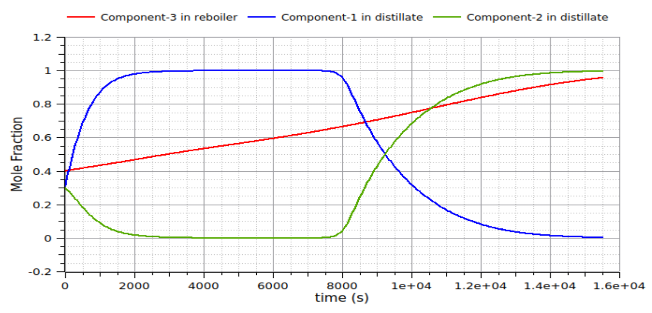


Figure 9. Mole fractions of desired components in distillate

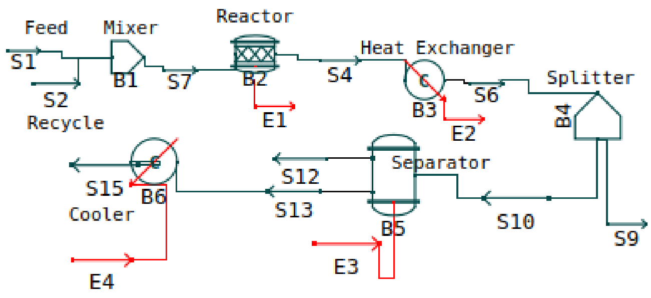


Figure 10. Recyle process for the Methanation flowsheet

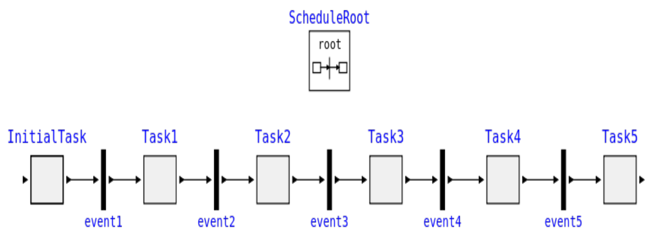


Figure 11. Schedule of the Methanation flowsheet using the `Scheduler` library

outlet temperature of 537.7°C. The effluent is then cooled to 260°C in a heat exchanger. The effluent is split into Methane rich stream containing 50% Methane at 93.3°C, and the other stream is fed to a separator to remove water.

Methanation flowsheet is modeled and simulated sequentially using the scheduling framework. The schedule is shown in Figure 11 and created in a similar manner as that for the batch distillation system. Again, similar to the batch distillation system, the initialization of only those components of subsequent tasks is done for which the state information is present in the respective previous task. The initialization of the rest of the components falls back to the default one described within the model itself. The following describes the tasks and their scheduling:

- In the first task, a pure feed stream ($S1$) and the recycle stream ($S2$) are mixed in the mixer ($B1$). The B1 output is sent to another stream, $S7$. So, the subsystems (units) $S1$, $S2$, $B1$, and $S7$ are active during `InitialTask`, and all other units do not participate in the simulation.
- In the second task, the mixed stream ($S7$), which was the output of the `InitialTask`, acts as an input for this task. It is taken to a reactor ($B2$) for the Methanation reaction. Hence the units $S7$, $B2$, $S4$, and $E1$ are active during `Task1`.
- In the third task, the product stream from the reactor acts as input for the heat exchanger. The units $S4$, $B3$, $S6$, and $E2$ are active when `Task2` is scheduled.
- During `Task3`, the cooled stream ($S6$) from the heat exchanger is taken to a splitter ($B4$) to split the stream into two material streams. During this task, the units $S6$, $B4$, $S9$, and $S10$ are active, and other units do not participate in the simulation.
- In the fifth task, the $S10$ output from the previous task is taken to the separator unit ($B5$) and separated to give two output material streams. The units $S19$, $B5$, $S12$, $S13$, and $E3$ are active during `Task4`.
- In the final task, i.e., `Task5`, one of the output streams from the separator unit ($S13$) is further cooled using the cooler unit (B6). In this task, the units $S13$, $B6$, $S12$, and $E4$ are active, and other units do not participate in the simulation.

The results obtained using the scheduling framework are identical to those shown by Reklaitis (1983). The number of equations solved and the simulation efficiency in terms of executable code size and memory requirements for each task are provided in Table 3. Since multiple models are now compiled through the tasks, one may perceive that the sum total of code size and the memory requirements increase as compared to the entire flowsheet. However, only one of the tasks is simulated at any given point in time. Thus, one has to consider only the resources corresponding to a single task's code size and memory requirements. Hence, as mentioned earlier in Section 2, the efficiency here is determined with respect to a given task only. It is nearly a three-fourth average reduction in the number of equations solved and code size, and a two-third average reduction in memory requirement compared to the simulation of the entire flowsheet. Here also, all the modeling equations are solved using DASSL in OpenModelica.

Table 3. Number of equations solved and simulation efficiency per task (as given by OpenModelica) compared to the entire Methanation flowsheet

Variable-Structure Models	**Number of Equations Solved**	**Executable Code Size (in KB)**	**Memory Requirement (in MB)**
InitialTask	648	1126	17.6
Task1	442	842	14.9
Task2	434	833	14.4
Task3	646	1126	16.9
Task4	675	1228	14.3
Task5	434	828	17.1
Entire Methanation Flowsheet	2339	3600	48.4
Average reduction per task compared to entire Methanation flowsheet	76.63 %	72.30 %	67.22 %

7 Conclusion and Future Work

An attempt has been made to tackle the problem of excluding the simulation code corresponding to the inactive components or subsystems while simulating a system. This approach generally leads to more correct results. In some cases, this may be the only way to achieve the end goals of a simulation. It is achieved by allowing the model developer to model the discrete behavior of their system through a schedule.

Construction of a schedule is made possible through a Modelica library `Scheduler`, developed in this work. The scheduling engine is implemented as a layer between the user model and the OpenModelica simulation environment. This approach is validated by applying it to the operation of a batch distillation column that separates a mixture, along with its real-time and interactive simulation. An example involving the steady-state solution to a chemical engineering flowsheet through a sequential modular simulation is also presented. In both cases, the results are identical to those reported in the literature, obtained through other approaches.

The future work would involve the performance and optimization aspects of the framework and its prototype. As there is a dependency between any two given tasks, their simulations cannot be done in parallel. But, their compilations can definitely be done in parallel, a feature available even in standard multi-core laptops. OPC UA, being a protocol on the network, can be a significant bottleneck (in terms of time and resources) for some models, and can lead to accuracy issues due to the inherent nature of the network. So, another method of simulation scheduling without OPC UA is desired. The usage of either of the two methods can be left to the model developer to decide as per their simulation requirements. Another direction to explore would be to extend the schedule's workflow pattern to include parallel routing. It would enable independent tasks to be simulated simultaneously and possibly in a distributed manner.

Acknowledgments

This work is partly funded by the National Mission on Education through ICT of the Ministry of Education, Govt. of India, through the FOSSEE project. The scheduling framework implementation for OpenModelica will be released as open source. In the meantime, those who are interested in accessing it are invited to contact the authors.

References

Brenan, K. E., S. C. Campbell, and L. R. Petzold (1996). *Numerical Solution of Initial-Value Problems in Differential Algebraic Equations*. Philadelphia: SIAM.

Briese, Lâle Evrim (2018-05). "Mission-Dependent Sequential Simulation for Modeling and Trajectory Visualization of Reusable Launch Vehicles". In: *Proceedings of the 2nd Japanese Modelica Conference*. Tokyo, Japan, pp. 230–239. DOI: http://dx.doi.org/10.3384/ecp18148230.

Casella, Francesco (2019-09). *Are variable-structure models allowed in Modelica? #2411*. URL: https://github.com/modelica/ModelicaSpecification/issues/2411 (visited on 2022-05-11).

Casella, Francesco (2021-02). "Is OpenModelica Finally Coming of Age?" In: *OpenModelica Annual Workshop 2021*. Linköping University. URL: https://openmodelica.org/images/M_images/OpenModelicaWorkshop_2021/0900_ComingOfAge.pdf.

FlexSim Software Products, Inc. (2022). *Tutorial 3 - Conditional Tasks*. URL: https://docs.flexsim.com/en/22.1/Tutorials/TaskLogic/Tutorial3ConditionalTasks/ConditionalTasksOverview/ConditionalTasksOverview.html (visited on 2022-08-24).

Fritzson, Peter (2014-11). *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Second Edition. Wiley-IEEE Press. ISBN: 978-1-118-85912-4.

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

GoldSim Technology Group (2022). *Behavior of Elements in Conditional Containers*. URL: https://help.goldsim.com/#!Modules/5/behaviorofelementsinconditionalcontainers.htm (visited on 2022-08-24).

Herps, Koen et al. (2022). "A simulation-based approach to design an automated high-mix low-volume manufacturing system". In: *Journal of Manufacturing Systems* 64, pp. 1–18. ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2022.05.013. URL: https://www.sciencedirect.com/science/article/pii/S0278612522000838.

Jack (2020-12). *An error of using multi-mode DAEs in Dymola*. URL: https://stackoverflow.com/questions/65199823 (visited on 2022-05-11).

K, Raja et al. (2019). "Implementation of Reconfigurable Manufacturing Systems in the Manufacturing of Turbo Charger Turbine Housing". In: *SAE Technical Paper Series* October. ISSN: 01487191. DOI: 10.4271/2019-28-0135.

Kahloul, Laid, Samir Bourekkache, and Karim Djouani (2016). "Designing reconfigurable manufacturing systems using reconfigurable object Petri nets". In: *International Journal of Computer Integrated Manufacturing* 29.8, pp. 889–906.

Kumar, Sudhakar et al. (2021). "OpenModelica OPC UA framework for control applications". In: *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA)*, pp. 78–83. DOI: 10.1109/ICCMA54375.2021.9646198.

Mehlhase, Alexandra et al. (2014-03). "An example of beneficial use of variable-structure modeling to enhance an existing rocket model". In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, pp. 707–713. DOI: http://dx.doi.org/10.3384/ECP14096707.

Modelica Association Project DCP (2019). *DCP Specification Document, Version 1.0*. Modelica Association. Linköping, Sweden. URL: http://www.dcp-standard.org.

Modelica Association Project SSP (2019). *SSP Specification Document, Version 1.0*. Modelica Association. Linköping, Sweden. URL: https://ssp-standard.org/.

Pop, Adrian et al. (2019-03). "A New OpenModelica Compiler High Performance Frontend". In: *Proceedings of the 13th International Modelica Conference*. Regensburg, Germany, pp. 689–698. DOI: http://dx.doi.org/10.3384/ecp19157689.

Process Systems Enterprise Ltd. (2004). *gPROMS Introductory User Guide*. URL: https://dokumen.tips/documents/gproms-introductory-guide.html (visited on 2022-08-24).

Reklaitis, G.V (1983). "Introduction to Material and Energy Balances". In: pp. 602–610.

Russell, Nick, Wil Mp Van Der Aalst, and Arthur HM Ter Hofstede (2016-02). *Workflow Patterns: The Definitive Guide*. MIT Press. ISBN: 9780262029827.

Sharma, Nikhil, Kannan Moudgalya, and Sunil Shah (2021-02). "Development of Modelica Library for Batch Distillation". In: *OpenModelica Annual Workshop 2021*. Linköping University. URL: https://openmodelica.org/images/M_images/OpenModelicaWorkshop_2021/1430_Nikhil_OpenModelica_Workshop2021.pdf.

Stüber, Moritz (2017-05). "Simulating a Variable-structure Model of an Electric Vehicle for Battery Life Estimation Using Modelica/Dymola and Python". In: *Proceedings of the 12th International Modelica Conference*. Prague, Czech Republic, pp. 291–298. DOI: http://dx.doi.org/10.3384/ecp17132291.

The AnyLogic Company (2022). *Controlling the model execution*. URL: https://anylogic.help/anylogic/running/controlling-model-execution.html (visited on 2022-08-24).

Vogt, Holger et al. (2021). *Ngspice User's Manual Version 35 (ngspice release version)*. URL: http://ngspice.sourceforge.net/docs/ngspice-35-manual.pdf (visited on 2022-08-24).

Hybrid physical-AI based system modeling and simulation approach demonstrated on an automotive fuel cell

Moritz Hübel[1] Nirmala Nirmala[1] Michael Deligant[2] Lixiang Li[3]

[1]Modelon Deutschland GmbH, Hamburg, Germany, Moritz.hubel@modelon.com
[2]Arts et Metiers Institute of Technology CNAM, LIFSE, HESAM University, Paris, France,
[3]Modelon Inc., Ann Arbor, United States

Abstract

This paper presents an approach on how to train a Neural Network model based on a detailed physical Modelica model. The necessary steps to generate training data from simulation will be explained as well as the generation process of a surrogate model. It will be shown, how the surrogate will be re-integrated into the Modelica system model. A benchmark based on accuracy and simulation performance will be performed. The tools used are Modelon Impact, an online modeling and simulation platform, the TensorFlow/Keras toolbox in a Jupyter Notebook which provides a Python-based interface for generating Neural Networks, and the Modelica Neural Network Library that provides functions for constructing Neural Networks within Modelica. The approach is demonstrated on an automotive fuel cell model which is part of an overall vehicle system model. One possible application is to train the neural network via repeated simulations and then to reuse it as an embedded software component for efficiently estimating fuel use and range for various driving cycles and ambient conditions.

Keywords: Machine Learning, Neural Networks, Hybrid Models, Hydrogen, Fuel Cell.

1 Introduction

Model-based system design and engineering plays a major role, not only in the development of new technical systems but also in supporting efficient usage or operation. On the one hand, Modelica as an open-standard multi-domain programming language can be used to describe complex technical systems on a fundamental basis. Text-book equations are often implemented on a component level, which can then be used to allow a graphical composition of system models using connection ports at the model interfaces to provide boundary conditions locally and close the equation system. Applying good modeling practices, replaceable models for different components can be implemented, allowing fidelity adaptation of the system by choosing different component models for different applications. Some applications require complex mathematical formulations that are necessary to describe a physical problem accurately and thereby sacrificing on computational performance during simulation of the model. Machine learning on the other hand allows creating models based on data without necessarily understanding the correlations between the inputs and the outputs on a fundamental basis. Neural Networks are a common approach to create models that can accurately predict the outputs based on different input combinations after the model has been trained sufficiently well. Neural Networks consists of node layers that are structurally inspired by the biological brains that can transmit signals to other neurons based. Due to their similarity with the biological counterpart, Neural Networks are categorized as a method of artificial intelligence (AI).

A hybrid physical-AI based model can consist of both components: models derived from first principal physics as well as data-based models such as Neural Networks. Especially the availability of physical component models providing an extensive data base for training, allows creation of hybrid models which can achieve better simulation performance while not sacrificing accuracy for a given question. Known physical relations in specific components can be used to train surrogate models in physics-guided machine learning processes [1]. That way, computationally expensive components can be replaced, and simulation performance can be increased if the specific component is not of interest for a specific set of internal calculations but needed to provide boundary conditions for other components in a system. In comparison to other regression techniques, like map or polynomial fitting, the Neural Network based approach allows for representation of strong non-linearities superior to polynomial fitting while allowing a higher degree of freedom and less data need compared to a 1:1 data mapping. Especially when more than two independent inputs need to be mapped, standard map-based approaches quickly run into limitations of limited matrix dimensions in various tools which is less critical for Neural Networks. In this paper, an approach will be presented that uses a detailed physical fuel cell model of a fuel cell vehicle to generate a reduced order model of the fuel cell itself to study fuel consumption for different driving cycles. Section 2 will introduce the problem and the underlying physical sub-models in detail. In section 3, the workflow to create the hybrid model will be described. Section 4 will show a benchmark of accuracy and performance of the hybrid model against the detailed physical model and section 5 will show the result for a long-term driving cycle simulation.

2 Problem Statement

Following the trends to reduce greenhouse gas emissions and save resources, one proposed approach for the mobility sector is hydrogen-powered fuel cells to generate electricity on demand and allowing a sufficient range while reducing the battery size drastically. A comprehensive model of such a fuel-cell vehicle has been developed in Modelica, a comprehensive summary on the underlying sub models has been published [2], [3].

The use-case of this paper is to calculate the range of the vehicle for a long route as quickly as possible and thereby demonstrate the performance improvement of a hybrid-model consisting of physical components and trained Neural Network models. Other potential improvements such as model solvability and robustness will not be discussed here.

A high-level schematic of the model in the Modelica modeling and simulation platform Modelon Impact is shown in Figure 1.

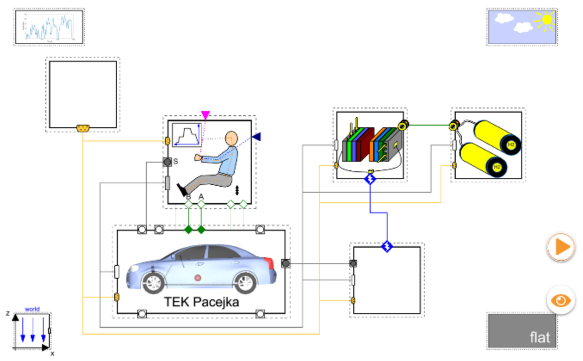


Figure 1: Top level Schematic of the vehicle system model, showing the replaceable drive cycle component (top left), the ambient condition component (top right) as well as the coupled vehicle model with driver, controls, drive train, fuel cell and hydrogen tank.

The model includes a driving cycle input defining the desired velocity trajectory for the vehicle. The WLTC2 Class 2 cycle [4], shown in Figure 2 is used here as a reference.

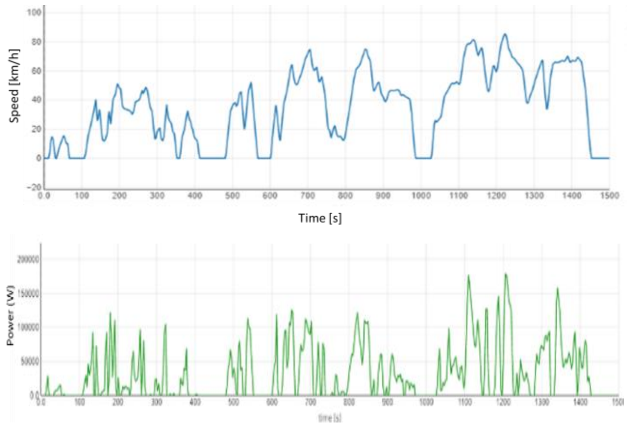


Figure 2: WLT2 driving cycle, velocity vs. time used as input for the reference scenario (top) and mechanical power at the drive-train shaft (bottom)

The vehicle model includes chassis, tires, breaks, and interacts with the driver and controls model. Thereby, it will define the propulsion power (torque and angular velocity) of the motor, considering the aerodynamic losses, rolling friction, and braking losses. The hybrid drivetrain includes a small battery, battery converter, fuel cell converter, and a DC motor. The electric power is provided by a proton exchange membrane (PEM) fuel cell stack fueled from a hydrogen storage tank.

3 Hybrid Model Generation Workflow

Generation of Proper Hybrid Models for Smarter Vehicles is the core topic of a research project funded by the German Federal Ministry for Economic Affairs and Climate Action. Different options to generate and integrate data-based models with physical Modelica models and tools are investigated. [5] have presented an approach to replace numerically inefficient and fragile non-linear equation blocks with surrogates during compilation. Another related workflow will be presented here that instead of interacting on the compiler level, utilizes the existing Modelica structure of replaceable sub-models for each component. This concept will be used to not only allow selection of different fidelity, first-principle physical models but also to integrate Neural Network surrogate models. Bringing the model back as part of the original Modelica system model will result in a hybrid model that can have superior performance and acceptable accuracy, so that it can be used for more applications such as those requiring real-time capabilities or even deployed as conventional FMU or eFMU on embedded hardware eventually.

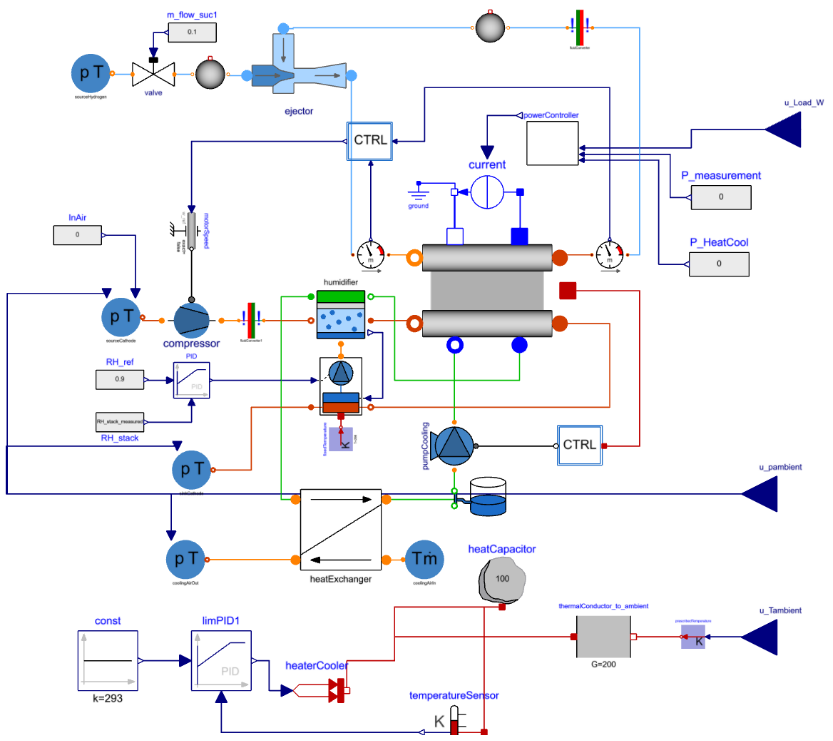


Figure 3: Overview of the physical system model including a detailed fuel cell stack and a simplified heating and cooling system of the car as well as control blocks. The independent inputs used for this study are the fuel cell load, the ambient pressure and the ambient temperature indicated by the connection ports on the right

3.1 Physical Model (Modelica)

The detailed Modelica model of the system is illustrated in Figure 3. The model includes the following components: a PEM fuel stack cell with empirical model for polarization, and dynamic mass and energy balance for anode, cathode and cooling channels; an ejector model to recirculate excessive hydrogen; a humidifier model to control the humidity of incoming air and recover some of the waste heat in the cooling loop; a cooling loop with heat exchange, pump and tank. The medium is represented as an ideal gas mixture with moisture using the NASA 7-coefficient model including the following components: H2, CO, CO2, H20, N2, O2. In addition, a simple heating and cooling system has been added, considering heat transfer to ambient and maintaining a convenient cabin temperature at 293K.

The following figures are giving an indication on the complexity of the model:

• Continuous states:	82
• Variables:	2572
• Linear equation blocks:	13
• Non-linear equation blocks:	4

To calculate fuel consumption for a given route, three independent inputs have been identified:

- Fuel cell power as a resulting output of the vehicle model for a given drive cycle (speed vs. time).
- Ambient temperature, primarily affecting the vehicles heating/cooling system but also the temperature and losses of the fuel cell.
- Ambient pressure affecting the air compressor

3.2 Modelica Simulation Tool

Modelon Impact is a cloud native Modelica modeling and simulation platform that has been used here. It can interact with Python through Rest API, e.g. using Jupyter Notebooks [6] or integrating Python scripts directly into the user interface using so called Custom Functions. Thus, allowing an easy integration of physical Modelica models with many AI-based models from Python environment.

3.3 Neural Network model generation

Classical machine learning can be categorized into supervised and unsupervised methods. The goal of the generated surrogate for fuel cell component that can be used to predict fuel consumption from requested electrical power here is to predict data from defined inputs, so to perform a regression task and falls into supervised methods. Generating Neural Networks became a very popular method for Machine Learning, yielding a range of tools. Commonly used tools in the Python environment includes TensorFlow/Keras (developed by Google) and PyTorch (developed by Meta). Also, the Julia language provides an efficient

environment for generating neural networks and combining with FMUs, Modelica [7] or Modia [8].
The approach presented here relies on using the TensorFlow/Keras toolbox in a Jupyter Notebook environment to generate a neural network from the detailed Modelica model.
For the main question addressed in the presented use-case, identifying the fuel consumption for different operating inputs, it is assumed that the fuel cell dynamics play a minor role and therefore, a quasi-static surrogate based on classical Neural Networks can be used. This assumption will be verified for a specific driving cycle in Section 4.
To train the model, samples of the three independent inputs are prepared, in this case, Saltelli [9] samples are used. Saltelli sampling is an efficient way to reduce the number of necessary data sets while keeping representative behavior over the considered data ranges. The number of Saltelli samples will be

$$\#Sa = N\,(2 * D + 2)$$

Where `D` is the number of free parameters, three in this study and `N` is the requested number of samples each, 10 here. Saltelli's extension of the popular quasi-random low-discrepancy Sobol sequence is used to generate coniform samples of the parameters space. To derive the training data set, a range for the inputs was specified as follows:

Power:	40kW to 140kW
Ambient Temperature:	253K to 333K
Ambient Pressure:	90kPa to 105kPa

As 10 samples in each range where created, the overall number of data sets or required simulation points of the detailed model is 80. Important outputs such as fuel flow, heating power and fuel cell current for the generated datasets are presented in Figure 4.

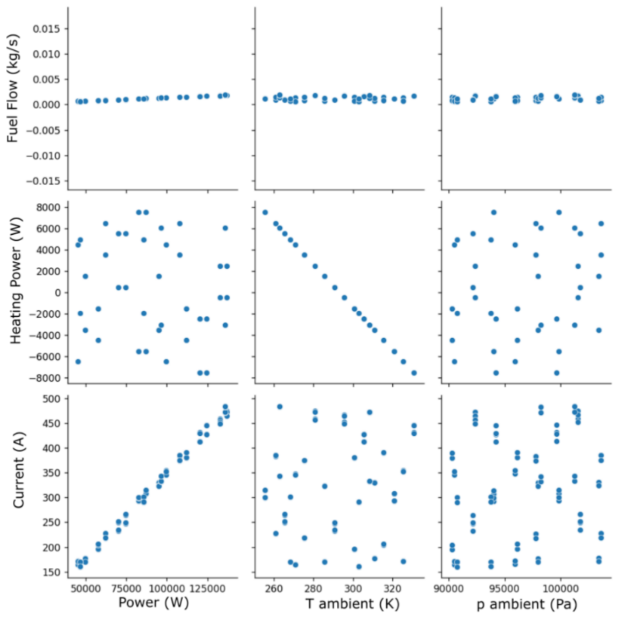


Figure 4: Physical model outputs used as training data. Showing current (bottom), heating power (middle) and fuel consumption (top) for different power (left), ambient temperature (middle) and ambient pressure (right).

Using the TensorFlow/Keras package, a structure for the Feed Forward Neural Network with an input layer, three hidden layers with 5 neurons each and an output layer is defined. Hyperbolic tangent Thanh is used as the activation function for all the neurons. The structure has been defined iteratively, increasing the number of layers and neurons until quantitative agreement of the output could be achieved without setting a specific criterion.

After defining the Neural Network model structure, the training of the weights and biases for all layers is performed using 40 sets of the physical models normalized simulation results while the other 40 normalized output sets are used as test data. During the training epochs, the accuracy (mean of squares of errors against the reference data) of the prediction improves as shown in Figure 5. Normalization of the values is required since working with physical SI units, values will differ several orders of magnitude.

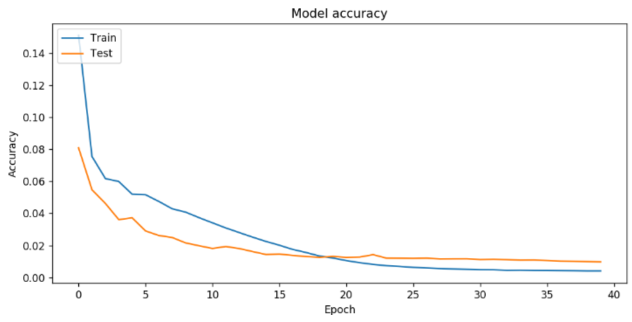


Figure 5: Accuracy of the Neural Network against the reference data from the physical model during training and test.

3.4 Hybrid Model

In the next step, the generated neural network model needs to be transferred back into the Modelon Impact platform. This permits users to benefit from a graphical model representation and convenient parameterization, structural adaptations, and post-processing. The transfer could be implemented in different ways, e.g.:

- FMU: As most system simulation tools, Modelon Impact allows direct import of Functional-Mockup units. This basically provides a wrapped-C-code with a standardized interface. However, FMU export from Python environments is currently under development and turned out to be not reliably working here.
- Using external C-code directly: a similar approach consists in converting the surrogate model into C-code. This approach is similar to the previous FMU approach with a less strict requirement on compliant FMU wrappers, however, since the graphical representation would first need to be created, this approach was not followed here.
- Implementing in Modelica: Introduced and published as "Neural Network Library" by [10], the structure of the Neural Network can be stored as Modelica code directly. Individual Layers can be represented by models, a set of pre-defined activation functions are available, the coefficients for weights and biases for each layer can be stored in the Modelica code or as external data file.

The approach presented in this paper will rely on the Neural Network Modelica library. The main advantage is the absence of compatibility issues and the availability of a graphical network representation. A potential disadvantage might be the adaption towards larger and more complex networks and regular structural updates during iterative surrogate generation processes. Figure 6 Shows the Neural Network structure as a Modelica model. It contains three generic inputs (u1, u2, u3), the input layer as well as the three hidden layers as introduced in the previous section. Connections between the layers and to the output (y) are vectorized.

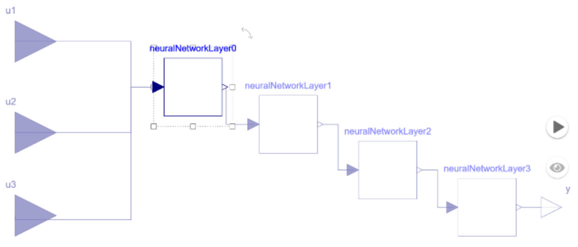


Figure 6: Neural Network Modelica model structure

4 Benchmark

To benchmark and validate the Neural Network surrogate, a comparison on accuracy, model complexity metrics and performance data against the original Modelica model is done. Figure 7 shows normalized fuel consumptions for the original model vs. the TensorFlow prediction and the Modelica surrogate. While both predictions usually match well as expected, some deviations from the original model can be observed due to the simple Neural Network structure used here. Also, minor deviations between the surrogate from TensorFlow against the surrogate based on Modelica can be observed.

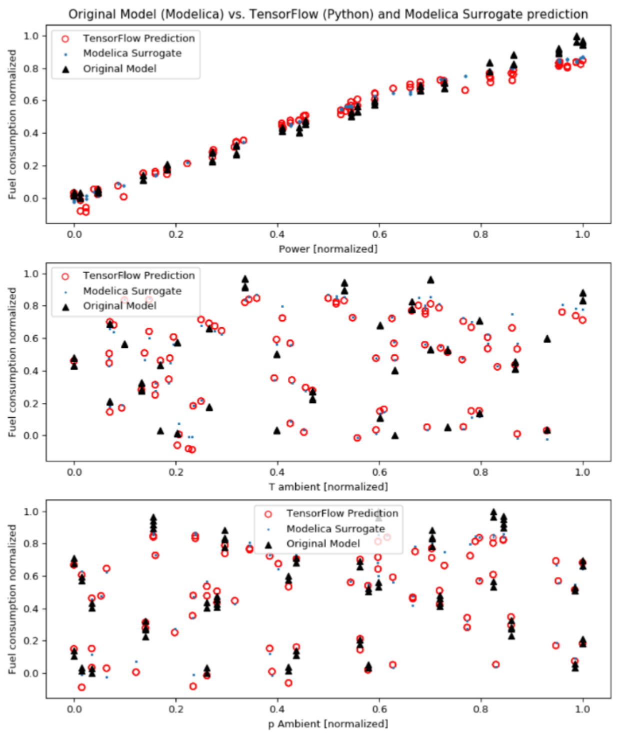


Figure 7: comparing steady-state result points for varying power (top), varying ambient temperature (center) and varying ambient pressure (bottom)

In addition to the steady-state analysis, a transient scenario has been considered for comparison, involving a scheduled load change of the fuel cell power setpoint as shown in Figure 8 while keeping the ambient temperature and pressure constant.

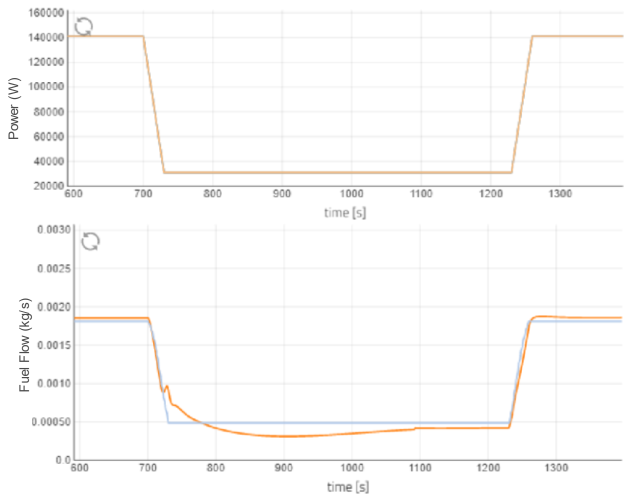


Figure 8: Fuel Cell load setpoint for both the physcial model (orange) and the surrogate (blue) model (top) and consumed hydrogen fuel flow for the load change scenario comparing the results of the original model with the surrogate (bottom)

The presented comparison of the fuel consumption showing well matching steady-state points with maximum differences of 2.7% in full load and 3.6% in low load. On the transients, deviations of physical model from the Neural Network surrogate model can clearly be seen. Effects, such as a control oscillation around 720s caused by the internal cooling flow supply of the fuel cell stack can not be reproduced by the steady-state surrogate.

Key metrics for complexity and performance are presented in Table 1, showing the superior performance of the Neural Network which is around 500 times faster compared to the physical model for this scenario.

Table 1: Complexity statistics and CPU time of surrogate model vs. physical model

	Surrogate	Physical Model
CPU Time	0.12s	49s
Continuous states	0	82
Variables	202	2572
Linear-Equations Blocks	0	13

Based on the performed analysis, it can be concluded that for prediction of the fuel consumption for a certain load variation within the trained data range, the Neural Network model can give reasonable results while being significantly better performing due to the removal of unused complexity. For the overall fuel consumption calculation in this artificial scenario, the accuracy is considered sufficiently well comparing with state-of-the art range predictions in modern fuel-cell vehicles that usually don't consider as many input parameters.

5 Simulation Scenario

A use-case scenario for the Neural Network model of the fuel-cell car could be a fuel consumption calculation of the vehicle for a given route the driver selects in the cars navigation system at different ambient conditions. While the physical model is validated and would be able to predict accurately from first principles, the model execution would take too long for this use-case. This fact becomes even more important considering the usage of lower-performance hardware used in automotive applications due to cost and weight advantages. Therefore, the usage of the surrogate model is proposed for predicting the fuel consumption of the car in a driving cycle, specifically the WLT2P-C2 introduced in section 2. The resulting fuel consumption for this scenario including the possible variation with changing boundary conditions is illustrated in Figure 9. Plausible outcome can be assumed based on the benchmark tests carried out in the previous sections.

The overall CPU time answering the specific question on "how much hydrogen will the vehicle consume for the given route under the different environmental conditions?" was about 2 seconds for a varying load including 5 sets of ambient temperature and 5 sets of ambient pressures, resulting in a total of 25 simulation scenarios.

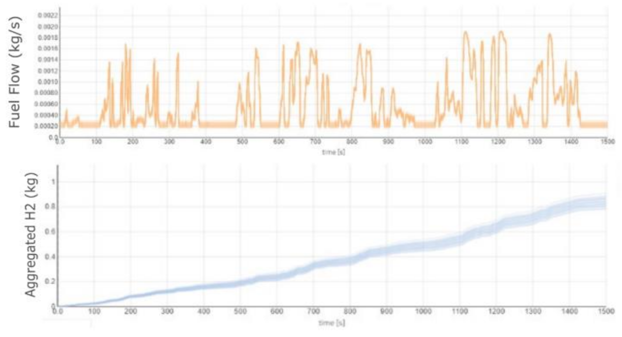


Figure 9: Resulting hydrogen consumption of the fuel cell for the WLTP2 driving cycle scenario with varying ambient temperatures and pressures (top) and aggregated hydrogen consumption of the fuel cell (bottom)

6 Summary and Outlook

The presented hybrid approach provides a powerful complementary feature to first principle based physical modeling which is typically used in Modelica models. The potential performance improvement has been demonstrated on an automotive fuel-cell use case, showing that simulation speed can easily be improved by a factor of 500 when only few outputs of a detailed model are relevant. The integrated Python interface in Modelon Impact allowed a convenient, scripting interface to TensorFlow/Keras and an easy data usage of physical model results for Neural Network training. The Modelica Neural Network library [10] enabled the usage of the generated Neural Network within the Modelica environment. The developed workflow can therefore be easily used for a variety of applications, including the speedup process of complex physical models or their sub-models for faster model-based design or improvement processes. In addition, application specific proper models can be generated and exported, e.g. as an FMU allowing the utilization for a subset of relevant questions while benefiting from tremendous performance improvements. One commonly known limitation of Neural Network models not addressed here is the usage outside the training data range. Unlike physical models, Neural Network models cannot be expected to predict behavior that has not been sufficiently covered by training data. This can result in very wrong predictions. In addition, further work is needed on capturing transient effects, as Feed-Forward Neural Network approaches only allow a steady state representation. However, the tight integration into the Modelica environment presented here allows the coupling with transient state representations at various points of a system model thus providing a promising solution for this challenge.

Acknowledgements

This research work has been partly funded by the Federal Ministry for Economic Affairs and Climate Action in the framework of PHyMoS – Proper Hybrid Models for Smarter Vehicles, Ref: 19I20022B, 2021-2024.

The models and workflows presented here have been developed by multiple contributors. Special thanks to Bryan Eisenhower for pioneering the work on surrogate modeling and developing workflows around the Modelica / Modelon eco-system.

7 References

[1] A. Daw, A. Karpatne and W. Watkins, Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling, http://arxiv.org/abs/1710.11431, 2017.

[2] S. Sigfridsson, "Fuel Cell Hybrid Vehicle Modeling," Masther Thesis, Lund University, Department of Automatic Control , 2018.

[3] S. Sigfridsson, L. Li, H. Runvik, J. Gohl, A. Joly and K. Soltesz, "Modeling of Fuel Cell Hybrid Vehicle in Modelica: Architecture and Drive Cycle Simulation," in *Proceedings of the 2nd Japanese Modelica Conference, Tokyo, Japan, May 17-18, 2018*, Tokyo, Japan, 2018.

[4] "Informal working group on Worldwide harmonized Light vehicles Test," 2012. [Online]. Available: https://unece.org/dhc-12th-session. [Accessed 16 08 2022].

[5] A. Heuermann, P. Hannebohm and B. Bachmann , "Replacing Strong Components with Artificial Neural Network Surrogates in an Open-Source Modelica Compiler," Linköping, 2022.

[6] P. Jupyter, "https://jupyter.org/," 2022. [Online].

[7] T. Thummerer, . J. Tintenherr and L. Mikelsons, "Hybrid modeling of the human cardiovascular system using NeuralFMUs," *Journal of Physics: Conference Series*, vol. 2090, 2021.

[8] F. Bruder and L. Mikelsons, "Modia and Julia for Grey Box Modeling," Linköping, 2021.

[9] J. Herman and W. Usher, "SALib: An open-source Python library for sensitivity analysis.," *Journal of Open Source Software*, vol. 2(9), 2017.

[10] F. Codeca and F. Casella, "Neural Network Library in Modelica," 2006.

7

A Dymola-Python framework for data-driven model creation and co-simulation

Sandra Wilfling[1] Basak Falay[2] Qamar Alfalouji[1] Gerald Schweiger[1]

[1]Institute of Software Technology, Graz University of Technology, Austria, sandra.wilfling@tugraz.at
[2]AEE INTEC, Austria, b.falay@aee.at

Abstract

The introduction of cyber-physical systems has been a recent development in energy systems. Cyber-physical systems contain digital components for applications such as monitoring or control. In many cases, modeling multiple aspects of such cyber-physical systems poses a challenge to conventional simulation tools. In addition, recent modeling approaches, such as data-driven modeling, are being applied. The combination of such data-driven models, which may consist of a different architecture than traditional models, with traditional models can be implemented through co-simulation methods. In co-simulation, components created from different simulation tools can be combined and coupled through standardized interfaces. This work presents a framework for data-driven model generation and co-simulation. The framework is implemented in Python and Dymola and is based on the Functional Mock-up Interface (FMI) standard. The framework implements the creation of data-driven models in Python, the generation of Functional Mock-up Units (FMUs) through the frameworks *uniFMU* and *pythonFMU*, as well the creation of a testbench model in Dymola and the co-simulation of this model. The framework is demonstrated on the application of a solar collector from a single family house heating system.

Keywords: Energy Systems, Modeling and Simulation, Data-driven Modeling, Co-Simulation

1 Introduction

The area of energy systems covers a wide range of applications, such as heating, cooling or electrical power systems. All these systems have in common that their demand for energy must be met by the energy providers while their energy demands are constantly growing. To respond to the increasing demand, energy providers have recently been focusing on embedding cyber-technologies into their systems in order to monitor and optimize system operation. This means that state-of-the-art energy systems are being extended into complex cyber-physical systems (Lund et al., 2017). The analysis of such cyber-physical energy systems poses new challenges in the area of simulation and modeling due to these systems' complexity (Palensky, 2014). Cyber-physical systems combine computational systems with other physical systems, meaning that their analysis requires combined modeling techniques for different system types. While the modeling of certain components can be implemented in specialized simulation tools, the full modeling of a combined system is a more difficult task. To model cyber-physical energy systems, different approaches exist, which can be classified into three groups: white-box, gray-box and black-box modeling (Arendt et al., 2018). White-box methods include traditional physical modeling methods based on system dynamics. Gray-box models may also be based on system dynamics, but may contain assumptions or approximations. Black-box models may consist of a completely different architecture than the underlying system. Traditionally, energy systems are modeled in simulation tools based on the physical relations of their components. Physical models are created by analysing the physical properties of the system, and these models are implemented mostly as white-box or gray-box models and based on the knowledge of the system dynamics and parameters. In order to model and simulate these systems, often numerical solvers are used to solve the underlying differential equations, as described by (Gomes et al., 2018). The numerical simulation methods are then implemented by simulation tools such as, for instance, Dassault Systemes Dymola®, MathWorks® Matlab/Simulink or EnergyPlus™. In contrast to traditional modeling, the data-driven modeling approach has recently been gaining popularity. Data-driven models are mainly based on modeling the underlying system as a black box. This means that the architecture and the parameters of the system are arbitrary, any structure can be used as a model. Data-driven models are mainly implemented by machine learning (ML) methods, such as linear regression models, decision-tree based models or neural networks. In the data-driven approach, the models are trained on existing measurement data by using optimization methods. This approach was applied for instance in (Ghofrani et al., 2020) and (Xu et al., 2019). The advantage of the data-driven modeling approach is that the ML models are trained based on measurement data and do not require exact system knowledge and parameters. While domain knowledge is helpful in creating the models, it is not necessary to know all features of the underlying system beforehand. A recent approach in cyber-physical systems modeling is the combination of physical and data-driven models in a co-simulation (CS) environment. The term co-simulation describes the combination of different simulation tools or environments. This may include a combination of continuous-time and discrete-time models, as well as simulation tools like Dymola or Matlab/Simulink. In co-simulation, different systems are integrated into a global environment. The co-simulation approach is used in applications such as building control systems, especially in model-predictive control (Wang et al., 2019). Applications in energy systems modeling or control often contain feedback loops containing components implemented in different simulation tools. These components must be coupled with each other through a defined interface. For this purpose, organizations such as the Modelica Association or the Institute of Electrical and Electronics Engineers (IEEE) have developed standards for co-simulation interfaces, such as the High-Level Architecture (HLA) (IEEE, 2010) or FMI (Modelica Association, 2020) standard. These standardized interfaces can be implemented by various tools without having to adapt the models for each simulation environment and are supported by different simulation tools. Additionally, these interfaces can be implemented by data-driven models, which may be created in programming languages such as Python. For our work, the FMI standard was selected. The FMI standard is developed by the Modelica Association, with current version FMI 2.0 (Modelica Association, 2020). The standard defines an interface for coupling models of different types and architectures. The FMI standard defines the format of models that are compatible to the standard as FMU. Simulation tools such as Dassault Systemes Dymola® (Dymola) or Simulink offer the option to generate FMUs from an existing model. For data-driven models, there are open-source tools available to export these models into the FMU format, such as the *pythonFMU* framework (Hatledal et al., 2020) and the *uniFMU* framework (Legaard et al., 2021).

1.1 Related Work

In energy systems modeling, different co-simulation frameworks have been created for the purpose of combining models created different simulation tools. For instance, several frameworks based on the FMI standard have been developed.The *Maestro* framework (Thule et al., 2019) implements a co-simulation orchestration engine for discrete-time and continuous-time co-simulation. The framework is implemented in Java, Scala and C and is based on the FMI standard. This framework supports Hardware-in-Loop (HiL) co-simulation. The *CyDER* (Nouidui et al., 2019) framework focuses on simulation for smart power grids. The framework is implemented in Python and suports HiL simulation. The *CyDER* framework offers the tool *Simulator2FMU*, which makes the interfaces of different power grid simulators compatible to the FMI standard. The main simulation is executed through the Python library *PyFMI* . In addition, smaller frameworks that focus on certain simulation tools have been developed. For the communication between Python and Dymola, several Python libraries have been developed. The Python library *buildingspy* (Wetter and USDOE, 2019) supports communication from Python to Dymola as well as to the Modelon Inc. *OPTIMICA* Compiler Toolkit. The Python package *dymat* (Rädler, 2013) supports reading and writing of Dymola output files. Based on existing Python libraries, different co-simulation frameworks have been developed. A Python-Modelica framework specialized for wind turbines called *MoWIT* was created in (Leimeister, 2019). This framework is based on the *buildingspy* library. Another framework called *PyMo* was created by (Febres et al., 2014).

1.2 Main Contribution

This work presents a workflow called *HybridCosim* that combines the creation of data-driven models with co-simulation. In this workflow, data-driven models are automatically created and then combined with physical models inside a co-simulation environment. The workflow is based on the FMI standard 2.0. The data-driven models are created in Python, converted into FMUs, and then simulated in Dymola as a part of an automatically generated testbench. The framework supports the creation of ML models of different architectures, as well as simulation in Dymola. The framework is demonstrated on a case study of a solar collector.

2 Methodology

The presented framework consists of four steps. Firstly, a data-driven model of an existing system is trained in Python. This model is then converted into an FMU. For the FMU, a Modelica testbench model is generated. Finally, the testbench is simulated in Dymola. An overview of the created workflow is given in Figure 2.

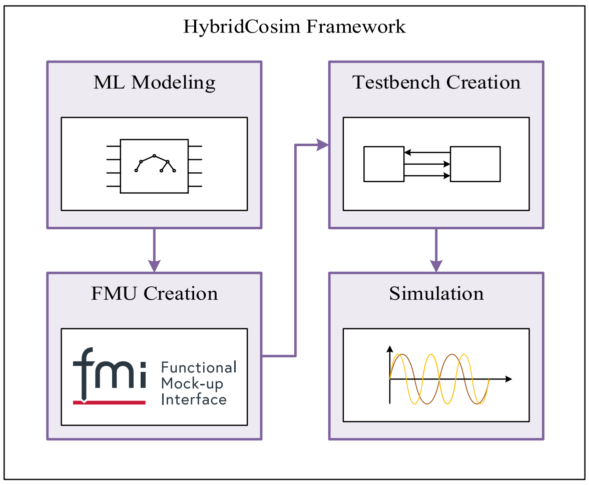


Figure 1. Co-Simulation Workflow

The first three steps of the workflow are executed purely in Python, the last step is executed through Python and Dymola. While the simulation itself is executed in Dymola, the orchestration and the result post-processing are done in Python. This framework is based on the research in (Falay et al., 2021) and (Wilfling et al.).

2.1 Model Training

To create data-driven models, we implemented a basic framework in Python to train models of different architectures, such as linear regression models, decision tree-based models or Support Vector Machine (SVM) models. These models could be created based on different datasets and feature configurations. The models are based on the research in (Schranz et al.) and the Python packages *scikit-learn* (Pedregosa et al., 2011) and *statsmodels* (Seabold and Perktold, 2010).

2.2 Interfacing - FMI

In our work, the FMI standard was used as an interface between models of different types. Therefore, the models had to be converted into the FMU format, for which the *uniFMU* framework (Legaard et al., 2021) and the *pythonFMU* framework (Hatledal et al., 2020) were evaluated. The *uniFMU* framework allows to export models from different programming languages such as Python, C#, Matlab or Java into an FMU. *uniFMU* supports the FMI standard 2.0 and contains a graphical user interface to generate and validate FMUs. The *pythonFMU* framework supports FMU generation from Python files.

FMU Creation

In our work, machine learning models implemented in Python can be translated into FMU format through the *pythonFMU* or *uniFMU* framework. While the *pythonFMU* framework supports the generation of a full FMU from a Python model, the *uniFMU* framework requires additional steps for creating the FMU. The FMU format contains a model description in Extensible Markup Language (XML), in which the model interface, consisting of the model inputs, outputs and parameters, and the basic model structure, which may include dependencies, is defined. To create an FMU through *uniFMU*, the model description must be adapted to the interface of the model.

For the FMU creation through *uniFMU*, a method to adapt the FMU model description automatically depending on the required inputs and outputs for the model was created. When using the framework, either of the two frameworks can be selected.

2.3 Automatic Testbench Creation

In our framework, Dymola was selected as the main simulation master, therefore our top-level model had to be implemented in Dymola. To automatically create a simple testbench for the FMU, a Python module was created. This module could generate a Modelica model based on input data, a specification of input and output features, and the FMU file. In addition, components created in Modelica could be imported and added to the model. The data-driven model was imported into Dymola and connected to Dymola-native modules or other FMUs. With this structure, it was possible to create fully-coupled systems, such as feedback control loops, or simpler systems with fewer components.

2.4 Simulation

For the generated top-level model including the FMU, a co-simulation was executed in the Dymola environment. This simulation was implmented using parts of the process created in (Wilfling et al.). In our implementation, the main control for the simulation is implemented in Python. The Python controller then sends commands to Dymola, which executes the simulation. The simulation commands are based on Modelica .mos scripts, which are automatically generated in Python. Figure 2 gives an overview of the implemented simulation method.

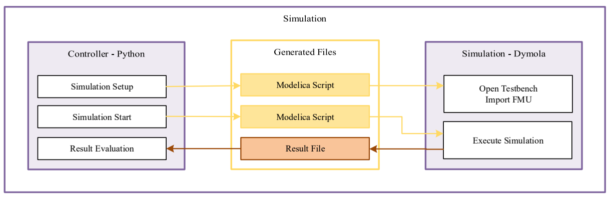


Figure 2. Python-Dymola Communication, c.f. (Wilfling et al.)

Alternatively, the testbench could be simulated directly through Dymola.

2.5 Framework Implementation

The framework was implemented mainly in Python. The framework is structured into four Python packages, each of which contains a step of the workflow. For each package, an example testscript is available to execute the operations of the step. In addition, all steps can be executed in combination as a full workflow run. In this case, the four steps are executed sequentially. During the execution of each workflow step, different files are created, which are then used by the next steps. The combined workflow requires two components as inputs: a dataset, and a configuration file containing definitions of the model inputs and outputs. Figure 3 depicts the full workflow structure with input and output files.

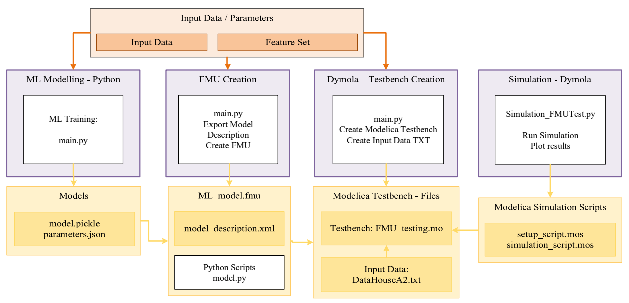


Figure 3. Workflow structure. Input files to the workflow are marked in red, automatically created output files are marked in yellow.

File Structure

The results of an experiment using the combined framework are stored inside a directory structure containing all automatically generated files including the models and the simulation results. An overview of this structure is given in Figure 4.

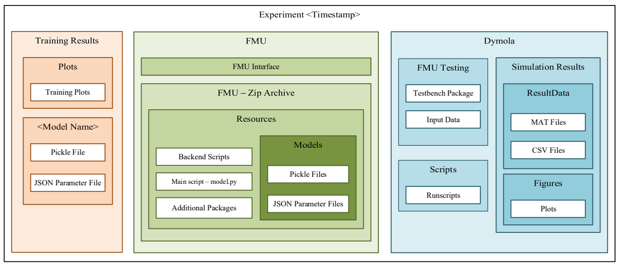


Figure 4. Directory Structure.

The structure is separated into three directories: one for the model training results, one for the FMU files, and one for the Dymola testbench and simulation results.

3 Case Study

To demonstrate the proposed framework, a case study on a use case from the energy domain was performed. For this purpose, a solar collector from a single-family house heating system was selected (Wilfling et al.). In a single-family house, the main heating demand is generated from the central heating for the rooms and the warm water consumption. In order to give options to optimize the heating energy consumption of such a house, the heating system should be modeled as accurately as possible. For this purpose, two different architectures for the data-driven model were evaluated.

3.1 Application - Solar Collector

The application of the case study was the supply temperature prediction for a flat-plate solar collector. This collector, which was already available as a physical model (Falay et al., 2021), should be modeled through a data-driven model. For the collector, the supply temperature T_S should be predicted based on the return temperature T_R, the mass flow through the collector V_d, the ambient temperature T_A and the solar radiation S_{Global}.

Underlying System

According to (Mahanta, 2020), the behavior of a flat-plate solar collector can be modeled through linear relations. The main factors affecting the solar collector supply temperature are the heat gain through the solar radiation and the heat loss to the ambient. While in the active state of the collector, the heat gain is affected by the mass flow through the collector. A simplified version of these relations can define the active behavior of the collector through Equation 1:

$$T_S = T_R + \frac{C_1 S_{Global}}{V_d} + \frac{C_2(T_S - T_A)}{V_d} \tag{1}$$

3.2 Data-driven Model

For the solar collector, a data-driven model was created through the model training part of the framework. To compare different model architectures, two models were created, one consisting of a linear regression model and one using Random Forest (RF) regression. The models were trained based on measurement data in a duration from 02/2019 to 10/2019, which was sampled with a timestep of 15 min. For the training, a train-test split of 0.8 was selected. The trained models were stored in the Pickle format.

3.3 FMU Creation and Testbench Generation

From the trained models, an FMU was created. Afterwards, a Dymola model to test the FMU was generated. This Dymola model was generated using the input measurement data and the description of the FMU inputs and outputs. Figure 5 depicts the generated Dymola model.

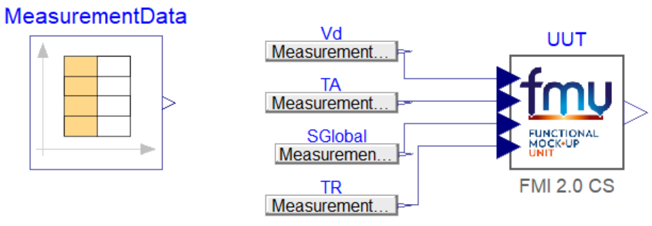


Figure 5. Graphical depiction of the generated Dymola model. The component placement was adapted manually for visualization.

The Dymola model contains the FMU and a Modelica *CombiTimeTable* containing the measurement data. For the *CombiTimeTable*, a text file was automatically generated from the input data to act as datasource.

3.4 Experimental Results

Finally, a simulation was executed for the generated Dymola model. The simulation duration was set to a time window of 30 days, with a timestep of 15 min. The results were post-processed in Python.

Performance Metrics

To evaluate the performance of the model, the metrics Coefficient of Determination (R^2), Coefficient of Variation of the Root Mean Square Error (CV-RMSE) and Mean Absolute Percentage Error (MAPE)(Falay et al., 2021) were selected. The performance metrics for the model are described in Table 1.

Table 1. Performance Metrics

Model	R^2	CV-RMSE	MAPE
Linear Regression	0.94	0.06	4.58%
Random Forest	0.98	0.04	2.21%

The performance metrics show the higher accuracy of the RF model. However, the linear regression model performs only slightly worse than the RF model despite its simple structure.

Timeseries Analysis

Figure 6 shows the timeseries analysis for the solar collector for a selected period of five days from the simulation duration. The timeseries analysis shows more accurate predictions of the data-driven model during daytime than during nighttime. This behavior was accredited to the characteristics of the solar collector, which is inactive during nighttime.

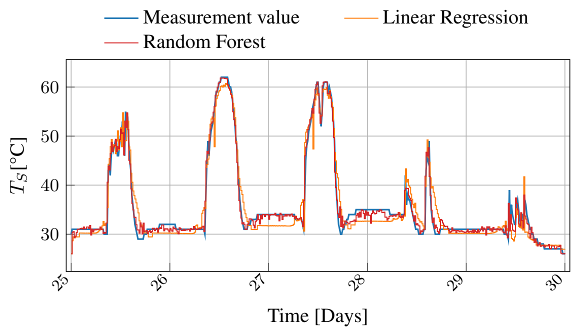


Figure 6. Timeseries Analysis for selected period from simulation.

The prediction error plots for the solar collector case study during the full simulation duration are depicted in Figure 7.

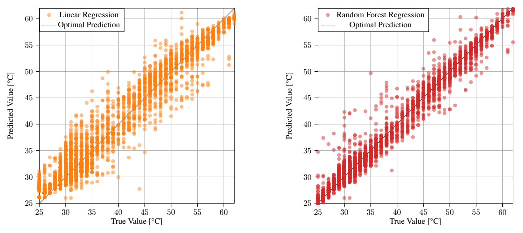


Figure 7. Predicted Values for T_S

From the prediction error plots, the higher accuracy of the RF regression model can be observed. The distribution of the residual error does not show significant anomalies.

4 Conclusion

We present a framework for data-driven model creation and co-simulation that allows the combination of different models. The framework is implemented in Python and Dymola and is based on the FMI standard. This framework allows automatic creation of data-driven models, translation into the FMU format, creation of a Dymola testbench model and simulation in Dymola. A case study performed on an application from the energy domain showed the performance of the created data-driven models.

4.1 Future Work

The current version of the framework gives many options for extensions. For instance, it is possible to extend the model training part of the framework to support additonal model types. The FMU creation part of the framework could be extended to support FMI 3.0, as well as include further extensions from FMI 2.0. Finally, the Dymola simulation part could be extended to support different simulation masters such as OpenModelica.

Acknowledgements

This work was funded through the project *"Next Generation Energy Services for Intelligent Buildings and Energy Systems" (NextGes)* as part of the Styrian *Zukunftsfonds Steiermark - "NEXT GREEN TECH"*.

References

K Arendt, M Jradi, H R Shaker, and C T Veje. Comparative Analysis of white-, gray- and black-box models for thermal simulation of indoor environment: Teaching Building Case Study. In *2018 Building Performance Modeling Conference and SimBuild Co-Organized by ASHRAE and IBPSA-USA Chicago*, page 8, 2018.

Basak Falay, Sandra Wilfling, Qamar Alfalouji, Johannes Exenberger, Thomas Schranz, Christian Møldrup Legaard, Ingo Leusbrock, and Gerald Schweiger. Coupling physical and machine learning models: Case study of a single-family house. *Modelica Conferences*, pages 335–341, September 2021. ISSN 1650-3740. doi:10.3384/ecp21181335.

Jesús Febres, Raymond Sterling, and Marcus Keane. A Python-Modelica Interface for Co-Simulation. *International Conference on Sustainability in Energy and Buildings*, page 11, 2014.

Ali Ghofrani, Seyyed Danial Nazemi, and Mohsen A. Jafari. Prediction of building indoor temperature response in variable air volume systems. *Journal of Building Performance Simulation*, 13(1):34–47, January 2020. ISSN 1940-1493, 1940-1507. doi:10.1080/19401493.2019.1688393.

Claudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: a survey. *ACM Computing Surveys (CSUR)*, 51(3):1–33, 2018.

Lars Ivar Hatledal, Houxiang Zhang, and Frederic Collonval. Enabling Python Driven Co-Simulation Models With PythonFMU. In *ECMS 2020 Proceedings Edited by Mike Steglich, Christian Mueller, Gaby Neumann, Mathias Walther*, pages 235–239. ECMS, June 2020. ISBN 978-3-937436-68-5. doi:10.7148/2020-0235.

IEEE. IEEE Std 1516'-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA),Framework and Rules. page 41, 2010.

Christian Møldrup Legaard, Daniella Tola, Thomas Schranz, Hugo Daniel Macedo, and Peter Gorm Larsen. A universal mechanism for implementing functional mock-up units.

In *11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, SIMULTECH 2021, page to appear, Virtual Event, 2021.

Mareike Leimeister. Python-Modelica Framework for Automated Simulation and Optimization. In *The 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, pages 51–58, February 2019. doi:10.3384/ecp1915751.

Henrik Lund, Poul Alberg Østergaard, David Connolly, and Brian Vad Mathiesen. Smart energy and smart energy systems. *Energy - The International Journal*, 137:556–565, October 2017. ISSN 03605442. doi:10.1016/j.energy.2017.05.123.

Deba Kumar Mahanta. Mathematical Modeling of Flat Plate Solar Collector. In *2020 IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES)*, pages 1–5, December 2020. doi:10.1109/PEDES49360.2020.9379669.

Modelica Association. Functional Mock-up Interface 2.0.2. 2020.

Thierry S. Nouidui, Jonathan Coignard, Christoph Gehbauer, Michael Wetter, Jhi-Young Joo, and Evangelos Vrettos. CyDER – an FMI-based co-simulation platform for distributed energy resources. *Journal of Building Performance Simulation*, 12(5):566–579, September 2019. ISSN 1940-1493. doi:10.1080/19401493.2018.1535623.

Palensky. Simulating Cyber-Physical Energy Systems: Challenges, Tools and Methods. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(3): 318–326, March 2014. ISSN 2168-2216, 2168-2232. doi:10.1109/TSMCC.2013.2265739.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Jörg Rädler. Dymat-hdf5 export and other modelica/python projects, 2013.

Thomas Schranz, Johannes Exenberger, Christian Møldrup Legaard, Jan Drgona, and Gerald Schweiger. Energy prediction under changed demand conditions:robust machine learning models and input feature combinations. In *Building Simulation 2021. International Building Performance Simulation Association*.

Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

Casper Thule, Kenneth Lausdahl, Cláudio Gomes, Gerd Meisl, and Peter Gorm Larsen. Maestro: The INTO-CPS co-simulation framework. *Simulation Modelling Practice and Theory*, 92:45–61, April 2019. ISSN 1569-190X. doi:10.1016/j.simpat.2018.12.005.

Jiangyu Wang, Shuai Li, Huanxin Chen, Yue Yuan, and Yao Huang. Data-driven model predictive control for building climate control: Three case studies on different buildings. *Building and Environment*, 160:106204, August 2019. ISSN 03601323. doi:10.1016/j.buildenv.2019.106204.

Michael Wetter and USDOE. Buildingspy, 4 2019. URL `https://www.osti.gov//servlets/purl/1569219`.

Sandra Wilfling, Basak Falay, Qamar Alfalouji, Johannes Exenberger, Thomas Schranz, Mina Basirat, and Gerald Schweiger. Smart Energy Systems Modeling - Component Exchange during Simulation. page 10.

Chengliang Xu, Huanxin Chen, Jiangyu Wang, Yabin Guo, and Yue Yuan. Improving prediction performance for indoor temperature in public buildings based on a novel deep learning method. *Building and Environment*, 148:128–135, January 2019. ISSN 03601323. doi:10.1016/j.buildenv.2018.10.062.

Towards Continuous Simulation Credibility Assessment

Maurizio Ahmann[1] Van Thanh Le[1,4] Frank Eichenseer[1,2] Frederik Steimann[1] Martin Benedikt[3]

[1]SETLabs Research GmbH, Germany, {Maurizio.Ahmann,VanThanh.Le,Frank.Eichenseer, Frederik.Steimann}@setlabs.de

[2]iCONDU GmbH, Germany, Frank.Eichenseer@icondu.de

[3]Virtual Vehicle Research GmbH, Austria, Martin.Benedikt@v2c2.at

[4]Free University of Bozen-Bolzano, Italy, vanthanh.le@stud-inf.unibz.it

Abstract

With the growing demand for virtual-informed decision-making in the development process of many engineering domains, the evidence in simulation results and thus simulation credibility becomes a critical aspect, in particular for releasing safety-relevant systems. However, simulation credibility is often interpreted to be of subjective nature. This paper summarizes basic assumptions for enabling the expression of credibility for building evidence in a more objective way. Based on these considerations, a concept is proposed that allows for an approximation of the credibility of simulations according to a discrete scale. The work is concluded by providing an implementation concept for a continuous simulation credibility assessment using a layered standard on top of the System Structure & Parameterization specification.

Keywords: *credibility, credibility assessment, verification, validation, traceability*

1 Introduction

With ever growing complexity of modern products across different industries and domains, the simulation of cyber-physical systems takes on an increasingly important role in the decision-making process. This can become critical for safety-relevant applications, like the simulation of automated driving systems (Knauss, 2017; Koopman, 2017), or simulation of medical devices (Rogers, 2019; FDA, 2021), where wrong decisions may have fatal consequences.

To mitigate the risk of making unreliable decisions based on insufficiently valid simulations, an added effort of verification and validation must be applied to models and simulations, to assure credibility in the simulation of complex systems.

1.1 Problem Statement and related work

Taking up complexity in state-of-the-art cyber-physical systems, it does not exclusively manifest through the technical complexity of the product itself, but also through the complexity of the product's underlying development process. More particularly, the product development in industries like the automotive industry typically has a strong distributed character, represented by complex supply chains, where simulation models are shared across organizational borders. This does not only go along with losing direct access to model sources, if models are provided as Black-Box models like Functional Mock-Up Units[1] (FMU), but also with a lack of knowledge about modeling assumptions, internal requirements, model design justification, or applied verification and validation techniques.

To keep this traceability information throughout the whole engineering process, the SET Level[2] project proposed a process framework for the execution of simulation-based engineering tasks (Heinkel and Steinkirchner, 2022) that supports for so called *credible* development of models and simulation, based on a detailed guideline focused on traceability, comprehensibility, and completeness of the documentation for modeling and simulation tasks.

However, to keep the framework generic and applicable to a wide range of engineering and simulation domains, this process framework is deliberately neither specifying the quality assurance any further nor does it define for distinguished methods to apply, dependent on the criticality of the simulation task.

The ITEA 3 project UPSIM[3] builds up its developments based on the SET Level result and smoothly extends this concept by introducing a formal quality assurance approach, targeting its integration into a collaborative, Continuous Integration (CI) environment for simulations and finally Digital Twins. In (Gall et al., 2021) the state-of-the-art and best practices in the development and management of credible Modelica models have already been identified within the UPSIM project, to be used as a basis for future improvements to work towards a well-documented, traceable development process for Modelica-based credible models.

The goal of the presented work is to introduce a concept for the continuous assessment of the credibility of simulations, using (among others) standards published by the Modelica Association and layered

[1]https://fmi-standard.org

[2]https://setlevel.de/projekt

[3]https://upsim-project.eu

standards on top of them. Furthermore, it will be shown that this concept can be applied domain-neutral and is extendable by design.

The remainder of this paper is organized as follows: First, an overview of the SET Level process framework for the realization of credible simulation tasks is given in Section 2, including its traceability concept. It is followed by a description of a credibility-based concept for quality assurance in Section 3, which will be applied to the recently proposed process framework that builds the foundation for distinguishing the applied degree of credibility. In Section 4, the generic implementation concept for credibility-based quality metrics will be sketched. These implementations will be finally used in a CI pipeline for the continuous assessment of the simulation's credibility.

2 Modeling and Simulation Process

The reliability and traceability of a decision-making process in engineering can be supported using reliable processes. If a simulation is involved in the decision-making process, an important requirement of a simulation process is to be embeddable into the overall development process frameworks. The SET Level *Credible Simulation Process* was – among other important assumptions, like taking into account the distributed character of the development and the necessity for traceability – built to fulfil this requirement. It therefore represents a lightweight and generic framework to be tailored to company specific handling of simulation.

2.1 Credible Simulation Process Framework

In (Heinkel and Steinkirchner, 2022) a complete framework is proposed to integrate a credible realization of simulation tasks into the overall product development process.

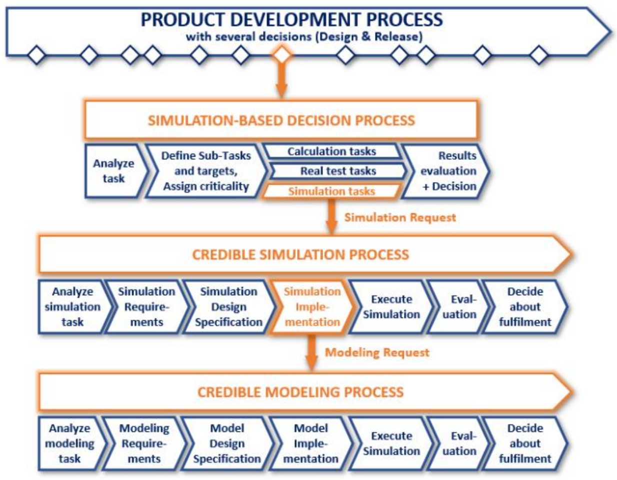

PRODUCT DEVELOPMENT PROCESS
with several decisions (Design & Release)
SIMULATION-BASED DECISION PROCESS
Analyze task
Define Sub-Tasks and targets, Assign criticality
Calculation tasks
Real test tasks
Simulation tasks
Results evaluation + Decision
Simulation Request
CREDIBLE SIMULATION PROCESS
Analyze simulation task
Simulation Require-ments
Simulation Design Specification
Simulation Imple-mentation
Execute Simulation
Eval-uation
Decide about fulfilment
Modeling Request
CREDIBLE MODELING PROCESS
Analyze modeling task
Modeling Require-ments
Model Design Specification
Model Imple-mentation
Execute Simulation
Eval-uation
Decide about fulfilment

Figure 1. Credible Simulation Process Framework

Figure 1 illustrates the relationship between the product development and its underlying processes. While there are several decisions to be made during the product development process, some decisions will incorporate simulation in the decision-making process, i.e., representing simulation-informed decisions.

2.1.1 Simulation-based Decision Process

These decisions will be made within so-called S*imulation-based Decision Processes* (SbDP) and are characterized by the fact that they contain simulation tasks, but may contain other tasks which do not use simulations. Each SbDP is assigned a decision consequence that shall be used as an input for approximating the criticality of an underlying simulation task that will be governing the actions for quality assurance.

For each of the underlying simulation tasks a *Simulation Request* is submitted.

2.1.2 Credible Simulation Process

The Simulation Request can be considered as an interface between the SbDP and the Credible Simulation Process (CSP). A Simulation Request transfers information from the SbDP to the CSP. Furthermore, requirements, specifications, and even implementations, if available, can be specified in advance.

The CSP has different phases needed to be executed, where the process is illustrated in form of a linear approach, but is typically applied and executed in an inherently iterative way, where steps are repeated several times. When it comes to model implementation, a *Modeling Request* is issued for the credible development of the models to be used for simulation.

2.1.3 Credible Modeling Process

Equivalent to a Simulation Request, a Modeling Request represents the interface from the CSP to the *Credible Modeling Process* (CMP). A Modeling Request contains all necessary information from the simulation process that is required to create a model for the dedicated simulation task, where distinctive requirements, specifications, and even implementations can be specified in advance.

The CMP will be processed equivalently to the CSP and is to be considered as an iterative process, as well.

2.2 Traceability

To allow for collecting relevant information to reconstruct how simulation results have been generated by execution of the CSP and CMP, this relevant information must be made available by means of metadata. For this purpose, a metadata specification – the *SSP Traceability Specification*[4] – has been drafted within the SET Level project as a layered standard in the

[4]https://pmsfit.github.io/SSPTraceability

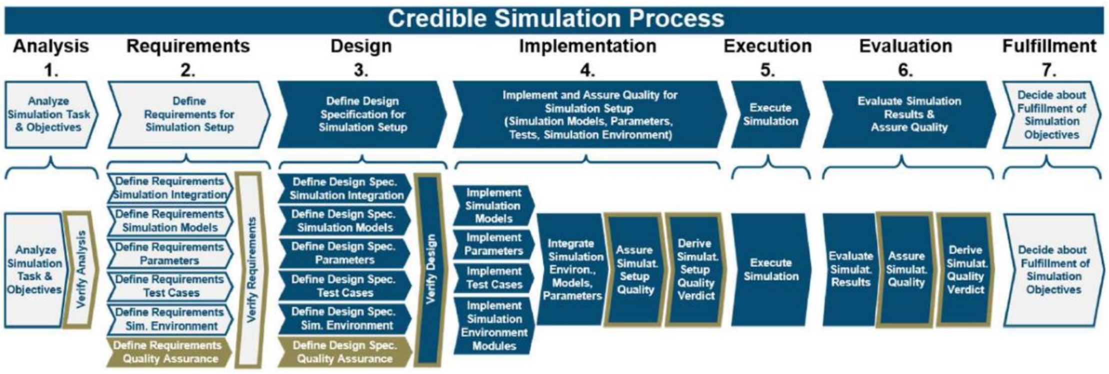

Credible Simulation Process
Analysis 1.
Requirements 2.
Design 3.
Implementation 4.
Execution 5.
Evaluation 6.
Fulfillment 7.
Analyze Simulation Task & Objectives
Define Requirements for Simulation Setup
Define Design Specification for Simulation Setup
Implement and Assure Quality for Simulation Setup (Simulation Models, Parameters, Tests, Simulation Environment)
Execute Simulation
Evaluate Simulation Results & Assure Quality
Decide about Fulfillment of Simulation Objectives
Verify Analysis
Define Requirements Simulation Integration
Define Requirements Simulation Models
Define Requirements Parameters
Define Requirements Test Cases
Define Requirements Sim. Environment
Define Requirements Quality Assurance
Verify Requirements
Define Design Spec. Simulation Integration
Define Design Spec. Simulation Models
Define Design Spec. Parameters
Define Design Spec. Test Cases
Define Design Spec. Sim. Environment
Define Design Spec. Quality Assurance
Verify Design
Implement Simulation Models
Implement Parameters
Implement Test Cases
Implement Simulation Environment Modules
Integrate Simulation Environ., Models, Parameters
Assure Simulat. Setup Quality
Derive Simulat. Setup Quality Verdict
Execute Simulation
Evaluate Simulat. Results
Assure Simulat. Quality
Derive Simulat. Quality Verdict
Decide about Fulfillment of Simulation Objectives

Figure 2. Phases and steps of the CSP (Heinkel and Steinkirchner, 2022)

SSP standard (Modelica Association, 2019) that can be used alongside the CSP and CMP. This specification defines the *Simulation Task Meta Data* (STMD) in form of an XML schema to store relevant meta data of process steps throughout the CSP and CMP. The specification provides for adding meta data regarding the processing scheme of each step in the CSP and CMP, namely for inputs, procedure, outputs, rationales, etc. Moreover, there is the possibility to add lifecycle information and linkage between certain resources of steps.

As the STMD will be used for the arrangement of the Continuous Integration pipeline in Section 4, the detailed application of this specification will be given within this further section.

3 Credibility Assessment

In order to be able to give an approximation about the simulation's credibility, it must be determined first how *credibility* can be specified and how to distinguish it from ordinary quality definitions. Further, a procedure must be derived on how to rank different degrees of credibility.

3.1 Distinguishing Credibility from Quality

For the term quality, there are existing many definitions that are widely accepted. From a generic point of view, *quality* can be defined as “the extent to which something has features which are good or bad, etc, especially features which are good” (Cambridge, 2022). From a technical point of view, quality is widely accepted to have two meanings (Vivek, 2005):

1. A characteristic of a product or service that bears on its ability to satisfy stated or implied needs.
2. A product or service free of deficiencies.

Following the above definitions, quality in simulation manifests itself through meeting its specified and unspecified requirements and being free from defects. To identify these target states, quality metrics and criterions are required. According to (Schütt, 2022), quality *metrics* are used to calculate metric results, based on data generated during test case execution, whereas a quality *criterion* is used to evaluate a metric result in relation to a threshold or evaluation scale.

The term *credibility* is interpreted more broadly, especially in the simulation domain. (Beisbart, 2019) notes that credibility may appear as something subjective since it can be reduced to being a property of a claim which deserves belief. He argues that however, the worthiness of belief is at least arguable that the degree to which a claim is credible in a certain context can be determined in an objective way.

Beisbart sharpens the term credibility by setting it in relation to the terms *truth* and *accuracy*: What users of simulation are interested in is simply the truth or, at least, that the outputs from their simulation come closest to the true values of the characteristics of interest. Nevertheless, the credibility of claims can only be established realistically based on the accuracy of the outputs. Therefore, credibility should be a function of the available evidence of a claim, in other words: The stronger the evidence of a claim, the more credible the claim.

(Oberkampf, 2019) supports the relation to truth and accuracy, as he states that simulation credibility deals with the assessment of the accuracy of certain system response quantities (SRQ) with respect to some true value or referent. He identifies three key issues on how to make credibility measurable:

1. How are the SRQ compared to the true values?
2. What is regarded as the true value?
3. What is the requirement for the simulation to be considered credible by the user or customer?

Whereas he carries out further that the first two issues are closely related to verification, validation, and uncertainty quantification, the third issue is rarely addressed in most simulation communities. He concludes that the requirement must be judged in relation to the accuracy of the simulation compared to the true value, even if the true value is also unknown or uncertain. He stresses further that the adequacy requirement should be set by the customer of the simulation.

(Gelfert, 2019) adds to this view that the assessment of model credibility needs always to be tentative and context-dependent – even for the rare case that a model may turn out to be successful and credible across a wide range of questions and applications.

3.2 Credibility Assessment concept

Based on the above statements, some basic assumptions for a credibility assessment can be formulated:

1. **Adequacy for purpose:** Credibility can only be formulated for a specific purpose of a simulation or a model and is never universally valid for a simulation or a model.
2. **Customer demand:** The required degree of credibility must be formulated in advance by the (external or internal) customer.
3. **Holistic approach:** Verification and Validation are crucial parts of a credibility assessment, but credibility should not be reduced to it, as a weak definition of requirements for example may be critical, if reference data is not available or limited.
4. **Collect Evidence:** To support credibility, evidence about the statement that is planned to be expressed with simulation must be collected. This evidence can be articulated with quality metrics and criterions.

We will follow a holistic approach, which means that for each relevant step of a process phase of the CSP or CMP (cf. Figure 2) evidence in form of evaluating quality metrics with quality criterions will be collected for assessing the credibility of the given objective of a simulation or model. The quality metrics will give supporting evidence about:

- How well founded and justified each development action is, for phases that will be carried out on the left side of the V-Model (VDI, 2021), namely the requirement definition and design specification phase; and
- how thoroughly the development actions are verified and validated (right side of the V-Model), namely for the implementation/integration and evaluation phase.

Another factor to be considered within the credibility assessment concept of this work is based on an insight from (Murray, 2015), gathered from the evaluation of several simulation case studies: For assessing the credibility of physically-based simulation models, a comprehensive view with respect to testing and validation procedures must be taken, as it is not enough to apply only few tests and validation methods, which leads to another principle of our concept, to distinguish the degree of credibility, based on:

- The collected amount of evidence; and
- the degree of formalization of the evidence.

This results in a discrete scale for the credibility assessment, consisting of three[5] *credibility levels* (CLs), where the lowest level provides for applying informal methods, usually based on expert opinion, whereas the highest level provides for applying metrics based on formal methods. The discrete scale is organized in a cumulative fashion: To reach the higher credibility level, the next lower credibility level needs to be accomplished before. This approach supports that the amount of evidence and the heterogeneity of applied methods rises with increasing credibility level.

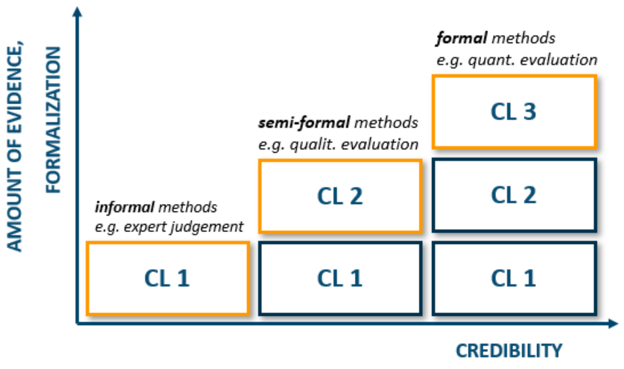

AMOUNT OF EVIDENCE, FORMALIZATION
CREDIBILITY
informal methods e.g. expert judgement
semi-formal methods e.g. qualit. evaluation
formal methods e.g. quant. evaluation
CL 1
CL 1
CL 1
CL 2
CL 2
CL 3

Figure 3. Discrete Credibility Level concept

The required credibility level for a specific simulation task will be determined by the customer in advance using a *Criticality Indicator*. This indicator is calculated in the course of a *Criticality Assessment* by evaluating:

1. The possible consequences in case of a wrong decision during the product development process;
2. the probability that a failure event happens at least once during the product lifetime that would lead to the described consequence; and
3. the influence of the simulation task on the decision of the associated engineering task.

This procedure is closely related to the M&S Criticality Assessment of the NASA Standard for Models and Simulations (NASA, 2016) and the criticality analysis of the Failure Mode and Effects Analysis (IEC, 2006).

For the formulation and application of quality metrics, many different quality metrics may exist that could be considered, depending on the applied simulation and model type and on the engineering domain. For this reason, the systematics are extended by a formulation concept for quality metrics (see Figure 4). On the first dimension, the applicability of the metrics will be differentiated. While there are metrics that can be applied to a wide range of simulation types (e.g., quality metrics that will give evidence about model

[5]The amount of three levels has been chosen in accordance with a process assessment that evaluates the degree to which a company has incorporated the CSP, similar to an A-SPICE assessment

convergency), others will be very specific and will remain subject to a certain engineering domain.

On another dimension, we will distinguish between abstract and concrete quality metrics. Abstract quality metrics will represent an implementation guideline and will be valid for a specific phase within the CSP.

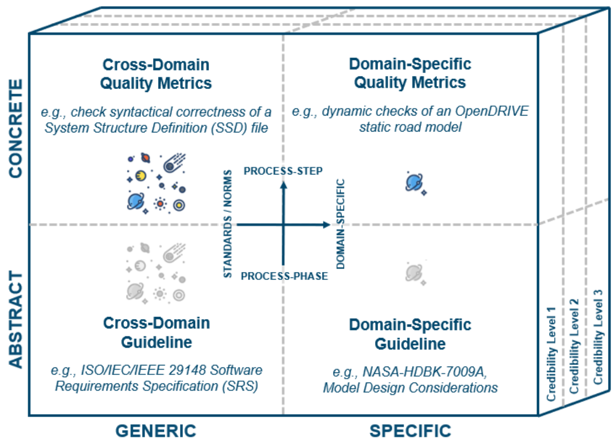

Cross-Domain Quality Metrics
e.g., check syntactical correctness of a System Structure Definition (SSD) file
Domain-Specific Quality Metrics
e.g., dynamic checks of an OpenDRIVE static road model
CONCRETE
ABSTRACT
PROCESS-STEP
PROCESS-PHASE
STANDARDS / NORMS
DOMAIN-SPECIFIC
Cross-Domain Guideline
e.g., ISO/IEC/IEEE 29148 Software Requirements Specification (SRS)
Domain-Specific Guideline
e.g., NASA-HDBK-7009A, Model Design Considerations
Credibility Level 1
Credibility Level 2
Credibility Level 3
GENERIC
SPECIFIC

Figure 4. Systematics for formulating Quality Metrics to assess simulation credibility

For example, an abstract metric for the specification of requirements, like the ISO 29148 standard (ISO, 2011) will be valid for the complete requirements phase of the CSP, whereas for the specific steps within the requirements phase the concrete metrics may adapt specifics with respect to the formulation of model requirements or test case requirements.

An abstract metric from a technical point of view can never be used for direct evaluation, as comparatively an abstract class in object-oriented programming can never be instantiated.

3.3 Abstract, Generic Quality Metrics

In the following, we will give a short description of abstract generic quality metrics for the phases that require quality assurance in the CSP (see Figure 2), following the concept given in Subsection 3.2. All following abstract metrics are equivalently representative for the CMP in the same way as for the CSP.

3.3.1 Requirements Phase

During the requirements phase of the CSP (Define requirements for simulation setup, see Figure 2) the requirements of the simulation task are broken down into the individual requirements for the simulation integration, models, parameters, test cases, and simulation environment. Essential within this phase is the clarification of general conditions, relevant assumptions, and requirements that the simulation must fulfil.

For the credibility of the simulation, it is important that requirements are formulated clearly and unambiguously in order to narrow down the interpretational space. Moreover, requirements shall be well founded to allow for traceability and should ideally be communicated using a standardized format to mitigate losing information due to incompatibility of requirement management tools.

This results in the following guideline for the different credibility levels, mainly derived from ISO/IEC/IEEE 29148 (ISO, 2011) and the INCOSE Systems Engineering Handbook (Walden, 2015):

1. **Semantic check**: All single requirements must be formulated according to semantic[6] criteria: A requirement must be necessary, unambiguous, complete, singular, achievable, and verifiable. Further, the collection of all requirements must be complete, consistent, affordable, and bounded.
2. **Check of traceability attributes**: All single requirements must contain traceability information to their source of the task analysis, to parent requirements (if child requirement), to peer requirements and to verification/validation results.
3. **Formal check**: Requirements must be provided using a standardized implementation like ReqIf (OMG, 2016) and must contain an agreed set of attributes.

3.3.2 Design Phase

In this phase of the CSP (Define design specification for simulation setup), consistent, coordinated specifications for all artifacts, models, tools, and parameters are elaborated.

The documentation of justifications for the selection of a specific design is essential for the credibility of the outcomes of this phase. This can be done on different levels of abstraction and detail, which should be aligned beforehand between customer and supplier.

The following guideline must be implemented for the credibility assessment of the design phase for the following credibility levels:

1. **Basic justification checks:** Basic justification of design specifications (e.g., the decision for a specific approach when modeling an effect, the source of parameter values, why specific test cases are used, etc.) must be documented, to check if the simulation has been built according to its given purpose and if the requirements have been respected. Must contain design assumptions and constraints, where necessary.
2. **Traceability check:** Check if the design specifications are supported formally, using linkage to other process phases. Especially, a decision must be justified with requirements and

[6]A detailed description of how to interpret the criteria can be found in the mentioned references

results of the task analysis. Moreover, links to verification results must be given to support proof of evidence. The traceability check may use results from meta-models of the simulation task, like a Goal Structuring Notation (GSN) (Spriggs, 2012).

3. **Constraints and Assumptions check:** Check if design constraints and assumptions are supported using linkage within the process phase to specifications of other steps (e.g., between test cases and parameters or models and environment, etc.). The traceability check may use results from meta-models of the simulation task, like a GSN.

3.3.3 Implementation Phase

In the implementation phase, the different elements of the simulation setup (models, parameters, test cases, simulation environment) will be implemented and integrated according to the information from the design specification phase. The verification of the functionality of the elements individually and in their interaction in the simulation setup will be carried out within this phase.

Verification is one of the most discussed topics in the simulation community. However, a comparable methodology on how to approach verification in modeling and simulation can be observed: Conducting code verification first, embracing software quality assurance and numerical algorithm verification, followed by solution verification, focusing on the estimation of the numerical accuracy of discrete solutions compared to their mathematical model; cf. (Roy 2005; Rider 2019).

In this phase, the transition from collecting evidence by means of foundation and justification to collecting evidence by thorough verification and validation is made. Therefore, the abstract metrics will focus on verification:

1. **Informal verification:** Basic code verification, beginning with Software Quality Assurance focusing on reliability and robustness from the perspective of software engineering, as for example described in (IEEE, 2014). Must be followed by static code checks and basic dynamic code checks.
2. **Formal, qualitative verification:** Verification must be carried out according to formal methods, and results will be evaluated according to qualitative acceptance criteria.
3. **Formal, quantitative verification:** Verification must be carried out according to formal methods and results will be evaluated according to quantitative acceptance criteria, using benchmarks as quality criterions that have been agreed on between customer and supplier.

3.3.4 Evaluation Phase

The final process phase of the CSP that requires for collecting evidence is the evaluation phase (Evaluate simulation results & assure quality). In this phase, simulation results are processed and evaluated. On the one hand, the simulation results are evaluated to make an assertion about the question the simulation task is trying to answer (e.g., “is the torque of the electrical motor sufficient to start the combustion engine?") and on the other hand, a confidence range of the given assertion must be approximated.

We propose the credibility-level-guideline for validation and uncertainty quantification as following:

1. **Informal validation:** Using informal validation techniques, as described in the taxonomy of (Balci, 1997), to assess if the simulation is a sufficient representation of the system. Evidence about the confidence of the given assertion must be given by approximating and propagating the worst-case configuration (in terms of uncertainties).
2. **Formal validation:** Formal techniques must be carried out for quantitative assessment of the validity of the simulation, using a validation benchmark as quality criterion that has been predefined and agreed upon. The validation domain must be predefined by a subject-matter expert (SME) to identify critical validation points. Evidence about the confidence of the assertion must be given by propagation of the upper and lower boundaries of the uncertainty range, that have been approximated by an SME.
3. **Uncertainty quantification:** The assertion must be supported by performing an uncertainty quantification, following the guideline proposed in (Roy and Oberkampf, 2010) for specific inputs that have been agreed upon.

These guidelines can be implemented specifically, on the one hand by different domains (e.g., the automotive, aerospace, or medical domain) and from another perspective with respect to different model types (i.e., for continuous models: Surrogate models, models based on algebraic equations, models based on ordinary differential equations, models based on partial differential equations, etc.).

Furthermore, the guidelines can also be implemented for system simulations, which will result in implementations that are widely based on evaluations using simulation and modeling standards.

3.4 Examples for Concrete, Generic Quality Metrics using Modelica Standards

The usage of standards can help to ease the implementation of the above proposed guidelines. In the following, some basic examples are provided on how to use standards of the Modelica Association to collect evidence. The following should be understood as examples on how to implement the abstract guidelines and do not have any claim to completeness of implementing all possible quality metrics, based on standards of the Modelica Association.

The System Structure Definition (SSD) is defined as part of the SSP specification and describes a nested hierarchy of interconnected (sub-)systems and atomic components (Modelica, 2019), which can be used for the implementation of system models.

To verify the implementation of the system structure, based on an SSD file, the following steps need to be carried out that can be considered as very basic CL1 quality metrics of the implementation phase for models (static code checks):

- **Syntax check:** Check if the SSD file implements the corresponding XSD [7], defined in the SSP specification.
- **Logic check:** Even if an SSD file implements the XSD correctly, it is not ensured that the proposed structure can be implemented. Therefore, logical checks need to be done: Check if all inputs are connected (if so required); check if connectors specified in the connections exist; check if the data types of wired connectors are consistent; check if connections are kept within their relevant subsystem.

To perform analogous static code checks for FMU model descriptions, the procedure for CL1 quality checks is similar:

- **Syntax check:** Check if the model description of the FMU implements the corresponding FMI (Functional Mock-up Interface) description schema.
- **Logic check:** Besides checking for a valid implementation of the XSD, the FMI specification defines some requirements and boundaries for the implementation (e.g., for definition of units or the allowed combination of attributes for specific variables). Therefore, some basic logic checks will be performed: Check if all units, used in the variable definitions are well defined with SI units; check if all types, used in the variable definitions are well defined; check if attribute combinations for variable definitions are valid.

When it comes to integration, it must be further ensured that the system structure and the underlying elements are compliant. Therefore, a basic integration check, based on static code analysis can be carried out.

- **Integration check:** Check if connectors, defined in the SSD, are consistent with variables/ports of the underlying component implementation (in the case of referenced FMUs this must match the name of the relevant variable in the referenced FMU).

[7]XML Schema Definition, https://www.w3.org/TR/xmlschema11-1

[8]https://github.com/virtual-vehicle/Credibility-Assessment-Framework/tree/main/Credibility-Development-Kit

The above-described quality metrics are basic examples of how to use implementation checks of Modelica Association standards within a credibility assessment, even if these checks can be considered state-of-the-art of many tools that implement these standards. For further examples, implementations and applications of concrete quality metrics, we refer to the repository of the so-called *Credibility Development Kit* that is outlined in Section 4.

4 Implementation Concept

The implementation of the concept for a credibility assessment proposed in Section 3 will intentionally be kept agnostic towards specific software applications and systems to enable broad usability. Within the UPSIM project, we are initiating implementations of the proposed concept that will result in a software development kit that provides quality metrics for each credibility level and each process phase of the CSP with the goal to provide reusable quality metrics for a credibility assessment in a transparent manner. It will be denoted as *Credibility Development Kit*[8] (CDK) in the following.

4.1 Credibility Development Kit

The core component of the CDK is a collection of Concrete Quality Metrics, mapped to process phases/steps and credibility levels that can be used to collect evidence for a credible statement of a simulation.

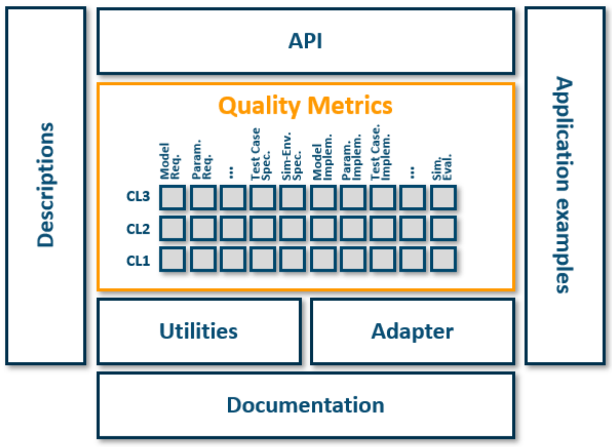


Figure 5. Components of the CDK

To support for the correct and unambiguous usage of Quality Metrics, further components are part of the CDK:

- **Descriptions, Documentation:** Descriptions of what Quality Metrics aim to measure and additional code documentation, using JSDoc[9].

[9]https://jsdoc.app

- **Utilities:** A collection of reusable helper functions that are used across different Quality Metrics with the purpose to have reproducible, traceable procedures, e.g., on how data is pre/post-processed.
- **Adapters:** A collection of functions that transform individual input data structures (may be standardized data structures like SSD or proprietary formats) into the data structures expected by the Quality Metrics implementations as an input (see Subsection 4.2).
- **API:** Facades as entry points to control pre-defined workflows, automatically using the correct adapters for individual Quality Metrics (cf. Figure 6).
- **Application examples:** Collection of best practices of proposed and former usage of Quality Metrics.

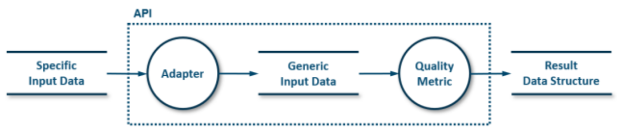


Figure 6. Basic data flow using Quality Metrics for credibility assessment. Notation: (DeMarco, 1979)

4.2 High-Level-Design

Taking into account the general considerations to enable broad applicability of Quality Metrics for a credibility assessment, the implementation of the CDK is carried out as a collection of Node.js[10] packages.

To support the applicability, especially to avoid insisting on too specific file formats, further considerations have been taken into account: A network-friendly data interchange format like JSON[11] is used and generic input data structures are defined that will be used as input to Quality Metrics. To allow for usage of different (standardized and proprietary) file formats, adapters can be provided that translate specific input data into the expected input data structure.

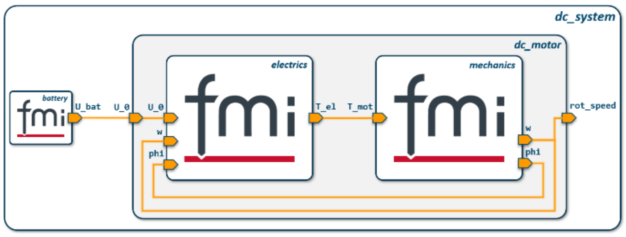
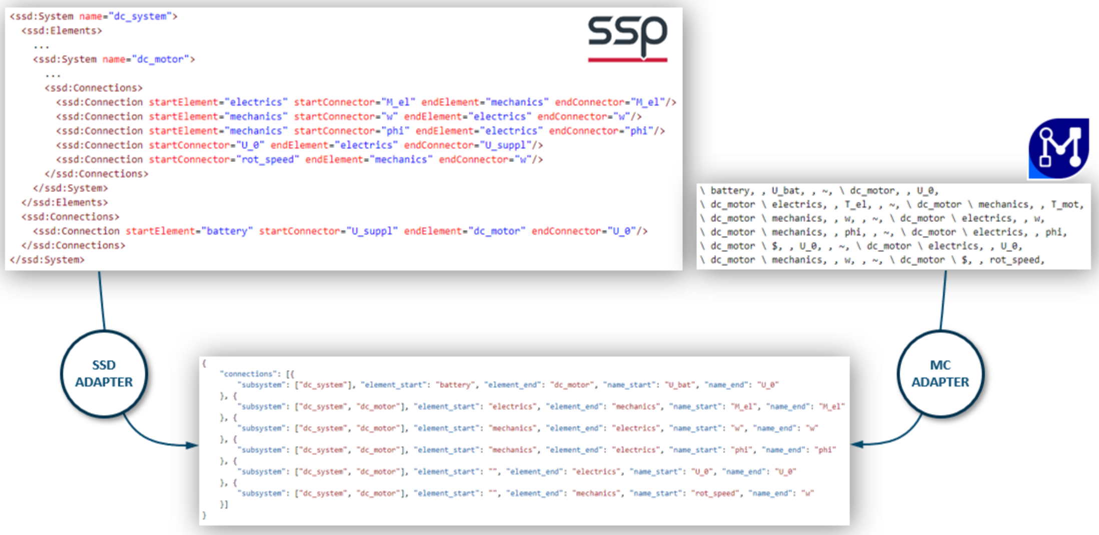


Figure 7. Simple example of a system model

As an example, for the simple system model in Figure 7, both the system and model connections are represented by the SSD standard and the proprietary format of the tool Model.CONNECT[12] will result in the same generic data structure (see Figure 8) that can be used as an input for a Quality Metric, once serialized.

As stressed in Subsection 3.2, some tests may require expert judgements. In this case the quality criterion will not directly be evaluated in the software function, but instead by an expert beforehand. To integrate this implicit evaluation, the statement of an expert (given for example as another serialized JSON structure) must be digitally signed by the expert, indicating the hash algorithm and signature encoding used (see Listing 1). This way the customer can verify if the judgement originates from an authorized expert, by checking against the individual public keys from expert's certificates that have been agreed upon before being accepted to carry out expert judgements.


```
<ssd:System name="dc_system">
  <ssd:Elements>
    ...
    <ssd:System name="dc_motor">
      ...
      <ssd:Connections>
        <ssd:Connection startElement="electrics" startConnector="M_el" endElement="mechanics" endConnector="M_el"/>
        <ssd:Connection startElement="mechanics" startConnector="w" endElement="electrics" endConnector="w"/>
        <ssd:Connection startElement="mechanics" startConnector="phi" endElement="electrics" endConnector="phi"/>
        <ssd:Connection startConnector="U_0" endElement="electrics" endConnector="U_suppl"/>
        <ssd:Connection startConnector="rot_speed" endElement="mechanics" endConnector="w"/>
      </ssd:Connections>
    </ssd:System>
  </ssd:Elements>
  <ssd:Connections>
    <ssd:Connection startElement="battery" startConnector="U_suppl" endElement="dc_motor" endConnector="U_0"/>
  </ssd:Connections>
</ssd:System>
```

```
\ battery, , U_bat, , ~, \ dc_motor, , U_0,
\ dc_motor \ electrics, , T_el, , ~, \ dc_motor \ mechanics, , T_mot,
\ dc_motor \ mechanics, , w, , ~, \ dc_motor \ electrics, , w,
\ dc_motor \ mechanics, , phi, , ~, \ dc_motor \ electrics, , phi,
\ dc_motor \ $, , U_0, , ~, \ dc_motor \ electrics, , U_0,
\ dc_motor \ mechanics, , w, , ~, \ dc_motor \ $, , rot_speed,
```

```
{
    "connections": [{
        "subsystem": ["dc_system"], "element_start": "battery", "element_end": "dc_motor", "name_start": "U_bat", "name_end": "U_0"
    }, {
        "subsystem": ["dc_system", "dc_motor"], "element_start": "electrics", "element_end": "mechanics", "name_start": "M_el", "name_end": "M_el"
    }, {
        "subsystem": ["dc_system", "dc_motor"], "element_start": "mechanics", "element_end": "electrics", "name_start": "w", "name_end": "w"
    }, {
        "subsystem": ["dc_system", "dc_motor"], "element_start": "mechanics", "element_end": "electrics", "name_start": "phi", "name_end": "phi"
    }, {
        "subsystem": ["dc_system", "dc_motor"], "element_start": "", "element_end": "electrics", "name_start": "U_0", "name_end": "U_0"
    }, {
        "subsystem": ["dc_system", "dc_motor"], "element_start": "", "element_end": "mechanics", "name_start": "rot_speed", "name_end": "w"
    }]
}
```

Figure 8. Adapter application example for a standard and proprietary format

[10]https://nodejs.org/en/about

[11]https://www.ecma-international.org/publications-and-standards/standards/ecma-404

[12]https://www.avl.com/-/model-connect-

Equivalently, an adapter is provided for signing expert judgements and transforming the judgement to the expected structure, as presented in Listing 1.

```
{
  "expert_judgement": "The func is fine",
  "signature": "9f066d7654fnk8hgc59f…",
  "hash_algorithm": "SHA256",
  "signature_encoding": "hex"
}
```

Listing 1. Example for a digitally signed expert judgement (shortened for better readability)

Similar to the unification of the input data that is passed to Quality Metric functions, outputs of Quality Metrics are (serialized) JSON structures and will always keep the following specific schema: The result, that indicates if the criterion of the Quality Metric has been matched or not, as well as logging information that adds valuable information to be used as feedback.

The outputs can be used for continuous assessment of the simulation's credibility (see Subsection 4.3).

4.3 Continuous Credibility Assessment

In the following, a proposal for the application of the outlined concept of this work for the continuous assessment of a system simulation will be sketched. In this application example, a multi-supplier scenario is presented that is using a Continuous Integration/ Deployment (CI/CD) pipeline for a continuous credibility assessment.

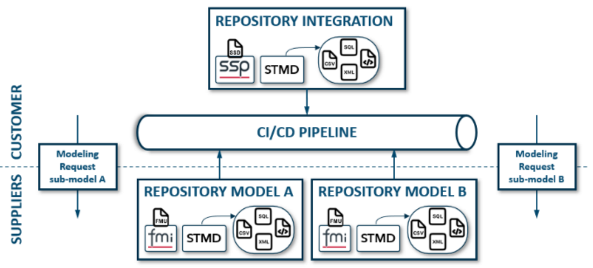


Figure 9. Example scenario

In the example application, we consider a distributed development of a system simulation, where a customer will integrate models from different parties – these parties will typically be suppliers that will provide black-box models (to keep the complexity of our example low, we consider only two sub-models).

The customer must have performed a criticality assessment (cf. Subsection 3.2) that will provide the required credibility level for the simulation task the customer plans to execute. In agreement with each supplier, the customer will select Quality Metrics (and quality criterions) from the CDK, according to the required credibility level that the corresponding sub-model needs to fulfill. This way, the customer and the supplier will have a bilateral, unified agreement on the interpretation of the credibility of this specific sub-model.

4.3.1 STMD for unique Artifact Identification

During the development of the sub-models, the suppliers must – in addition to the bare provision of the model – provide additional credibility documentation that can be used as inputs for the selected Quality Metrics of the CDK, especially for those process phases where data cannot be produced directly within the continuous integration pipeline (like for example simulation results, generated for verification and validation). To ensure the correct mapping of these credibility documentation artifacts to corresponding Quality Metrics, we propose to provide an STMD file (see Section 2) for unique identification.

For each step of the Credible Simulation and Credible Modeling Process the STMD schema enables for providing credibility documentation via the *Rationale* element. We propose to add the sources of all additional artifacts required for the credibility assessment within a Rationale element as *Resource* element and specify the mapping as a *MetaData* element of the Resource. The following elements and arguments shall be used with the subsequent conventions:

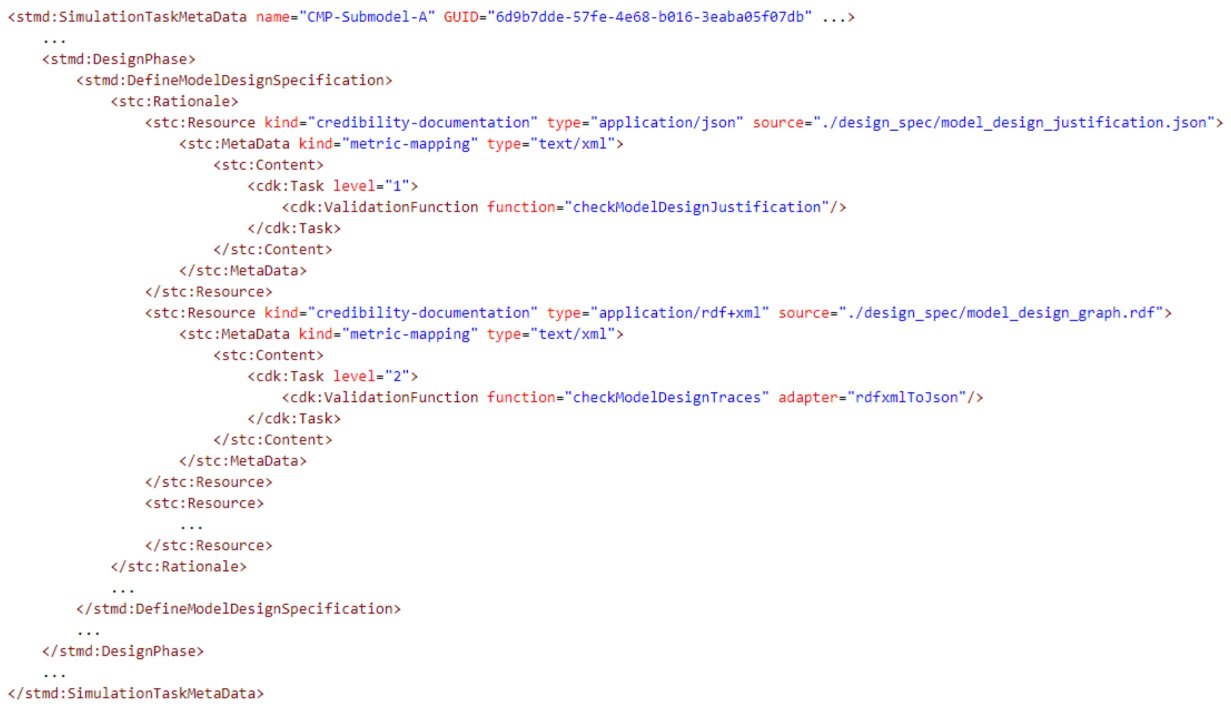
- **Resource element:** The argument *kind* must be specified as "credibility-documentation"; the argument *type* must indicate the MIME-Type of the resource, as required in the STMD specification; the argument *source* must indicate the URI[13] of the credibility documentation resource
- **Metadata element:** The argument *kind* must be specified as "metric-mapping"; the argument *type* must be specified as "text/xml"
- **Content element:** This element is allowed to contain user-defined elements. We propose to use an element called *cdk:Task*
- **cdk:Task element**: The argument *level* must be provided to indicate the corresponding credibility level
- **cdk:ValidationFunction element:** The argument *function* must be provided to indicate the target Quality Evaluation function. The argument *adapter* must be specified if the resource must be transformed to the expected file format and data structure; it must indicate the name of the function of the adapter to use

Figure 10 is presenting an excerpt of the proposed unique resource identification for a Quality Metric of the Design Specification phase that requires the provision of an expert judgement for CL1 and a graph for CL2.

4.3.2 Continuous Assessment and Deployment

In this manner, the credibility assessment must be performed for each phase and step of the process. The

[13]https://www.w3.org/TR/uri-clarification



```xml
<stmd:SimulationTaskMetaData name="CMP-Submodel-A" GUID="6d9b7dde-57fe-4e68-b016-3eaba05f07db" ...>
    ...
    <stmd:DesignPhase>
        <stmd:DefineModelDesignSpecification>
            <stc:Rationale>
                <stc:Resource kind="credibility-documentation" type="application/json" source="./design_spec/model_design_justification.json">
                    <stc:MetaData kind="metric-mapping" type="text/xml">
                        <stc:Content>
                            <cdk:Task level="1">
                                <cdk:ValidationFunction function="checkModelDesignJustification"/>
                            </cdk:Task>
                        </stc:Content>
                    </stc:MetaData>
                </stc:Resource>
                <stc:Resource kind="credibility-documentation" type="application/rdf+xml" source="./design_spec/model_design_graph.rdf">
                    <stc:MetaData kind="metric-mapping" type="text/xml">
                        <stc:Content>
                            <cdk:Task level="2">
                                <cdk:ValidationFunction function="checkModelDesignTraces" adapter="rdfxmlToJson"/>
                            </cdk:Task>
                        </stc:Content>
                    </stc:MetaData>
                </stc:Resource>
                <stc:Resource>
                    ...
                </stc:Resource>
            </stc:Rationale>
            ...
        </stmd:DefineModelDesignSpecification>
        ...
    </stmd:DesignPhase>
    ...
</stmd:SimulationTaskMetaData>
```

Figure 10. Example of using the Rationale element within a STMD file to ensure unique identification of Quality Metric inputs

concrete Quality Metrics that have been agreed upon between customer and suppliers will be evaluated to aggregate the results for being able to derive a condensed credibility level for the individual sub-models.

By providing the logging information, next to the results of the Quality Metric evaluation (see Subsection 4.2), developers are able to get feedback and can iterate on developing towards the required credibility level. For each iteration the credibility level will be determined, based on the changes done; thereby the credibility is assessed continuously.

It is important to point out that the interpretation of the overall credibility of a simulation or a model depending on the achieved atomic credibility levels of each process step is the responsibility of the applying parties. As we see the risk of error propagation in a simulation process, we propose to use a minimum rule, which means that the overall credibility level is equal to the lowest credibility level of a single process step. This does not apply only for the sub-models by execution of the CMP on the supplier side, but for the overall simulation by execution of the CSP on the customer side, as well.

The final step of the CI/CD pipeline is the automatic deployment of an SSP package. This package is having a unique identifier with the purpose to enable an unambiguous mapping of the deployed SSP package to the assessed credibility level. Again, it must be emphasized that the credibility level connected to this package is only valid for the given purpose of the simulation and does not represent a globally valid certification for the simulation model.

5 Conclusions and Outlook

In this paper, a proposal is presented on how to continuously assess the credibility of models and simulations. As the term *credibility* is interpreted differently in the community, we clarified some basic assumptions that our concept for credibility assessment is built on.

A central aspect of these assumptions is that credibility increases with the amount of evidence given about the statement that is planned to be expressed with simulation. However, even if we recommend distinguishing the degree of credibility by using a discrete level scale that is separated according to the formal degree of the applied methods and aims at increasing the amount of collected evidence with increasing credibility level, it must be stressed that this classification can only represent an approximation of the credibility. Simulation tasks will differ in their modeling approach, complexity, and prior knowledge. Therefore, we emphasize taking these conditions into account when applying the presented concept, especially when it comes to selection of Quality Metrics.

From implementational point of view we presented an approach on how to document and reference to additional information that will be required for assessing the credibility. In our concept, we adapted to the concept of STMD that references additional traceability and credibility documentation by adding it as a layered standard on top of the specification of Modelica standards. Still, there's a trend to add credibility information directly to models (cf. Gall et al., 2021), which is also subject of investigation within the UPSIM project. These developments can be considered to be used in further developments of the continuous credibility assessment in this work.

Acknowledgements

This work has been partly supported by the European ITEA3 Call6 project UPSIM[14] – Unleash Potentials in Simulation (number 19006). The work was funded by the German Federal Ministry of Education and Research (BMBF, grant numbers 01IS20072H and 01IS20072G).

References

Osman Balci (1997). Verification, Validation and Accreditation of Simulation Models. In: *Proceedings of the 1997 Winter Simulation Conference*. Blacksburg, Virginia, USA. DOI: 10.1145/268437.268462

Claus Beisbart (2019). What is Validation of Computer Simulations? Toward a Clarification of the Concept of Validation and of Related Notions. In: *Beisbart, C., Saam, N. (eds) Computer Simulation Validation. Simulation Foundations, Methods and Applications*. Springer. DOI: 10.1007/978-3-319-70766-2_2

Cambridge University Press (2022). *Cambridge Dictionary*. Accessed July 20, 2022 [Online]. URL: https://dictionary.cambridge.org

Tom DeMarco (1979). Structured Analysis and System Specification. Yourdon Press, New York. P. 352. ISBN: 978-0138543808

FDA (2021). Assessing the Credibility of Computational Modeling and Simulation in Medical Device Submissions, Draft Guidance for Industry and Food and Drug Administration Staff. U.S. Department of Health and Human Services Food and Drug Administration

Leo Gall et al. (2021). Continuous Development and Management of Credible Modelica Models. In: *Proceedings of the 14th Modelica Conference*. Linköping, Sweden, pp. 359–374. DOI: 10.3384/ecp21181359

Axel Gelfert (2019). Assessing the Credibility of Conceptual Models. In: *Beisbart, C., Saam, N. (eds) Computer Simulation Validation. Simulation Foundations, Methods and Applications*. Springer. DOI: 10.1007/978-3-319-70766-2_10

Hans-Martin Heinkel and Kim Steinkirchner (2022). Credible Simulation Process Framework. Tech. rep. Robert Bosch GmbH and PROSTEP AG. URL: https://gitlab.setlevel.de/open/processes_and_traceability/credible_simulation_process_framework

IEC (2006). Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA), Second Edition. International Electrotechnical Commission. International Standard IEC 60812:2006.

IEEE (2014). Software Quality Assurance Processes. Institute of Electrical and Electronics Engineers. International Standard IEEE 730. DOI: 10.1109/IEEESTD.1998.88284

ISO/IEC/IEEE (2011). Systems and software engineering - Requirements engineering. Geneva, Switzerland: International Organization for Standardization (ISO)/International Electrotechnical Commission/Institute of Electrical and Electronics Engineers (IEEE), (IEC). ISO/IEC/IEEE 29148

Alessia Knauss et al. (2017). Paving the roadway for safety of automated vehicles: An empirical study on testing challenges. In: *Proc. IEEE Intelligent Vehicles Symposium (IV)*. Los Angeles, USA. pp. 1873–1880. DOI: 10.48550/arXiv.1708.06988

Philip Koopman and Michael Wagner (2017). Autonomous vehicle safety: An interdisciplinary challenge. In: *IEEE Intelligent Transportation Systems Magazine*, Volume 9, No. 1, pp. 90–96. DOI: 10.1109/MITS.2016.2583491

Modelica Association (2019). System Structure and Parametrization. Report 1.0. URL: https://ssp-standard.org/publications/SSP10/SystemStructureAndParameterization10.pdf

David Murray-Smith (2015). Testing and Validation of Computer Simulation Models. In: *Simulation Foundations, Methods and Applications*. Springer. pp. 233-343. DOI: 10.1007/978-3-319-15099-4_13

Vivek Nanda (2005). Quality Management System Handbook for Product Development Companies (1st ed.). CRC Press, pp. 1-3. DOI: 10.1201/9781420025309

NASA (2016). Standard for Models and Simulations. NASA-STD-7009A, Rev. A, Change 1. pp. 50-54. National Aeronautics and Space Administration. URL: https://standards.nasa.gov/sites/default/files/standards/NASA/w/CHANGE-1/1/nasa_std_7009a_change_1.pdf

William L. Oberkampf (2019). Simulation Accuracy, Uncertainty and Predictive Capability: A Physical Sciences Perspective. In: *Beisbart, C., Saam, N. (eds) Computer Simulation Validation. Simulation Foundations, Methods and Applications*. Springer. DOI: 10.1007/978-3-319-70766-2_3

Object Management Group (OMG) (2016). Requirements Interchange Format (ReqIF). Standard Specification 1.2. URL: http://www.omg.org/spec/ReqIF/1.2

William J. Rider (2019). The Foundations of Verification in Modeling and Simulation. In: *Simulation Foundations, Methods and Applications*. Springer. pp. 271-293. DOI: 10.1007/978-3-319-70766-2_11

Erica N. Rogers (2019). Risk Assessment for Medical Devices. In: *Shayne C. Gad. Integrated Safety and Risk Assessment for Medical Devices and Combination Products.* Springer. DOI: 10.1007/978-3-030-35241-7

Christopher John Roy (2005). Review of code and solution verification procedures for computational simulation. In: *Journal of Computational Physics*, Volume 205, Issue 1. pp. 131-156. DOI: 10.1016/j.jcp.2004.10.036

Christopher John Roy, William L. Oberkampf (2010). A Complete Framework for Verification, Validation, and Uncertainty Quantification in Scientific Computing (Invited). In: 4*8th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition.* Orlando, USA. DOI: 10.2514/6.2010-124

Barbara Schütt et al. (2022). A Taxonomy for Quality in Simulation-based Development and Testing of Automated

[14]https://itea3.org/project/upsim.html

Driving Systems. In: *IEEE Access 10*, pp. 18631-18644 DOI: 10.1109/access.2022.3149542

John Spriggs (2012). GSN – The Goal Structuring Notation: A Structured Approach to Presenting Arguments. Springer London. DOI: 10.1007/978-1-4471-2312-5

VDI (ed) (2021). Development of mechatronic and cyber-physical systems. Technical guideline. Verein Deutscher Ingenieure. VDI/VDE 2206

David Walden, Garry Roedler and Kevin Forsberg (2015). Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities. 4. Hoboken, NJ: Wiley. ISBN: 978-1-118-99940-0