THE AMERICAN MODELICA 2024 CONFERENCE

STORRS
OCTOBER 14-16

PROCEEDINGS OF THE AMERICAN MODELICA CONFERENCE 2024
Storrs, Connecticut, USA, October 14-16, 2024

EDITORS
Michael Tiller, Hubertus Tummescheit, Luigi Vanfretti, Christopher Laughman, and Michael Wetter

PUBLISHED BY
Modelica Association and Linköping University Electronic Press

Series info: Linköping Electronic Conference Proceedings Nr. 207
ISBN: 978-91-8075-568-9
ISSN: 1650-3686
eISSN: 1650-3740
DOI: 10.3384/ecp207

ORGANIZED BY
North America Modelica Users' Group
na.modelica-users.org

in co-operation with:
Modelica Association
c/o PELAB, Linköpings Univ.
SE-581 83 Linköping
Sweden

CONFERENCE LOCATION
Innovation Partnership Building
University of Connecticut Tech Park
159 Discovery Drive
Storrs, CT 06269
USA

SWAMINATHAN GOPALSWAMY

Research Professor, Director
Connected Autonomous Safe Technologies (CAST) Lab
Department of Mechanical Engineering, Texas A&M

BIO

Swami Gopalswamy is a Research Professor in the department of mechanical engineering at Texas A&M University, and the director of the Connected Autonomous Safe (and Sustainable) Technologies (CAST) Lab. His research at A&M is focused on control-enabled technologies that can lead to safer and more sustainable transportation solutions. The research at CAST spans development of fuel-efficient powertrains to new vehicle platooning concepts to techno-business paradigms such as "Infrastructure Enabled Autonomy". He is also working on developing algorithms and software to enable coordinated movement of a fleet of heterogeneous air and ground vehicles autonomously, on off-road terrains. He has introduced Modelica as part of classes that he has taught at Texas A&M.

Previously, Dr. Gopalswamy was the CEO of Emmeskay, Inc., a company that provided modeling, simulation and other advanced technology solutions to the automotive industry. As part of Emmeskay, Dr. Gopalswamy and his colleagues were engaged in advanced modeling and control projects based on Modelica. Subsequently, he was LMS-VP at LMS International and Siemens PL, responsible for managing a global MBSE engineering services team, as also leading the development of Control-design software products. Prior to Emmeskay, Dr. Gopalswamy was a Staff Research Engineer at General Motors R&D Center.

At GM, he was engaged in solving powertrain control challenges (such as smooth clutch-to-clutch shifting) and developing powertrain concepts (such as magnetorheological fluid devices and hybrid electric powertrains).

He holds a doctorate and master's degree in controls from the department of mechanical engineering at the University of California, Berkeley. He has a bachelor's degree in Mechanical Engineering from the Indian Institute of Technology, Madras.

ABSTRACT

Modeling, Simulation, and Autonomous Vehicles: Challenges and Opportunities

Modeling is abstraction of reality. As an engineering practice and skill set, modeling has evolved and been evolving over the decades aligned with two fundamental priorities – form and function. Geometric modeling to capture form has made tremendous progress, and spawned form-driven functional (primarily structural) modeling through finite element decompositions and analysis. However, the interest of the Modelica community is on functional modeling that abstracts away form and focuses on dynamic behavior of systems.

I will discuss some of my experiences in functional modeling, along with the evolution in techniques over the decades. This experience spans aero and automotive systems. The growing maturity of the field is observed in the distillation and separation of critical techniques to enable specialization and cross-leverage of expertise. The explosive growth of software content in functionality over the last decades has been a catalyst for the use of modeling for algorithm development and verification and validation of embedded software.

With the advent of machine learning, artificial intelligence, and growth of autonomous driving systems, new challenges and opportunities have emerged for modeling to be useful. There is morphing of the fundamental priorities between form and function, as photorealistic rendering of environment becomes integral to functional modeling driven by the presence of sensors such as cameras and LIDARs that observe the environment and influence dynamic behavior. The sheer growth in complexity and scale of these systems forces the exploration of distributed computations and simulations, bringing with it new questions to be answered. Uncertainty modeling and stochasticity are part and parcel of learning systems, providing yet another avenue for evolution of modeling.

Finally, continued evolution of abstraction leads to the next layer, where machines are interacting with humans. I will finish with some thoughts on the opportunities for modeling in this context.

CLAS A. JACOBSON

Senior Fellow, Systems Engineering
Carrier Global Corporation

BIO

Clas Jacobson is a Carrier Senior Fellow with over 25 years experience in Systems Engineering and Controls.

Jacobson has focused his efforts on “Model Based Development” and has contributed to several areas to develop and deploy computational methods and tools for the effective use of model based development across Carrier.

Jacobson served as Chief Scientist for the United Technologies Systems & Controls Engineering (UTSCE) organization across United Technologies Corporation and, before that, in several UTRC management and technical positions. In his Chief Scientist position, he led the creation of the UTSCE organization with a mission of driving product and product development transformation enabled by systems and controls engineering technologies.

Jacobson was a (tenured) Associate Professor at Northeastern University before joining Carrier.

ABSTRACT

Energy Urgency, Computation and Role of “Systems” Methods & Tools

Today, energy considerations are critical and policy issues matter more than in the recent past, so that there is a newfound urgency to address cost, climate and security. These kinds of decisions require very large capital investments; how do we address the risks involved and the choices that are to be made? Several things are needed that are described in this talk. First, well-crafted design flows are needed to determine how overall capabilities will be considered, designed, implemented, and maintained. Second, it is essential that we understand what we can compute today and what we cannot compute, as well as the reliability of these computations and their accessibility to a wide set of groups. A few examples of energy districts and data centers will be used to illuminate what we can do and what we cannot. What should the audience thus take away from this presentation? Computing is much more than simulation. Computation --- in reliable and trusted ways - is a key element to decision making today, but not all the pieces are in place. In particular, the workforce in “computational engineering” and their training in academia needs to be used and designed to scale.

CONFERENCE CO-CHAIRS

Dr. Michael Tiller, JuliaHub

Dr. Hubertus Tummescheit, Modelon

PROGRAM CO-CHAIRS

Prof. Luigi Vanfretti, RPI

Dr. Michael Wetter,
Lawrence Berkeley National Laboratory

CONFERENCE EXECUTIVE COORDINATOR

Dr. Christopher Laughman,
Mitsubishi Electric Research Laboratories

LOCAL CONFERENCE CHAIR

Prof. George Bollas, University of Connecticut

LOCAL CONFERENCE COORDINATOR

Janesa Mackin, University of Connecticut

Dr. Behnam Afsharpoya,
Dassault Systèmes

PROGRAM COMMITTEE

Bernhard Bachmann, Fachhochschule Bielefeld

John Batteh, Modelon

Christian Bertsch, Robert Bosch GmbH

Torsten Blochwitz, ESI ITI GmbH

George Bollas, University of Connecticut

Francesco Casella, Politecnico di Milano

Yan Chen, Pacific Northwest National Lab

Atiyah Elsheikh, Mathemodica.com

Olaf Enge-Rosenblatt, Fraunhofer

Yutaka Hirano, Woven Planet Holdings, Inc.

Jianjun Hu,
Lawrence Berkeley National Laboratory

Yaoyu Li, University of Texas at Dallas

Alexandra Mehlhase, TU Berlin

Thierry S Nouidui,
The United African University of Tanzania

Kaustubh Phalak, Ingersoll Rand

Adrian Pop, Linköping University

Johan Rhodin, ModSimTech, LLC

Clemens Schlegel, Schlegel Simulation GmbH

Michael Sielemann, Modelon Deutschland GmbH

Giorgio Simonini, EDF

Martin Sjölund, Linköping University

Wilhelm Tegethoff, TLK-Thermo GmbH

Alfonso Urquia,
Universidad Nacional de Educación a Distancia (UNED)

Stefan Wischhusen, XRG Simulation GmbH

Dirk Zimmer, DLR

CONTENTS

Thermofluid Systems 2

Electromechanical Systems

Language/Tools 3

Thermofluid Systems 3

PAPERS

Model-Based Design and Characterization of an Actuator with Low-Boiling Liquid

Christoph Steinmann[1] Johannes Herold[1] Jens Schirmer[1]

[1]Institute of Electromechanical and Electronic Design, TUD Dresden University of Technology, Germany
christoph.steinmann@tu-dresden.de

Abstract

Visually impaired people rely on special equipment for access to graphic representations in digital form. The available devices are very large and expensive. A simple and cost-effective alternative to the existing concepts for haptic displays is therefore desirable. This paper evaluates the concept of a lifting actuator based on a fluid with a low boiling point for this purpose. A functional prototype is constructed and its behavior is characterized. A corresponding model is built and validated to simulate the actuator and to analyze its operation. It provides detailed information about the actuator that can be used to further develop the design and to make decisions on the usability of the new actuator in the product design process. Following test runs and investigations on the model, the actuator concept proved to be suitable for haptic display devices under certain assumptions. Therefore the newly developed model presents a good starting point for future revisions of the concept.

Keywords: haptic display, multi-domain model, liquid-to-gas phase change actuator, low-boiling liquid

1 INTRODUCTION

Tactile displays make it possible for visually impaired people to interact with graphical representations of information. Text-to-speech or classical braille lines can not fulfill this functionality to the same degree (Baldwin et al. 2017). It is therefore desirable to further improve this type of device.

Information can be made palpable primarily by thermal, electrical or mechanical stimulation. Consequently a variety of actuators can be used to generate these effects. For dot based graphical output the most common method is to feel mechanically elevated surfaces (Vidal-Verdú and Hafez 2007). These can be actuated by electric motors (Wagner, Lederman, and Howe 2002; Sarakoglou, Tsagarakis, and Caldwell 2005), shape memory alloy (Howe, Kontarinis, and Peine 1995; Velazquez et al. 2005), light (Mirvakili et al. 2021), pneumatic devices (Caldwell, Tsagarakis, and Giesler 1999; Wilhelm 2015) or piezoelectric actuators, as often used in commercial products (Tieman and Zeehuisen 1988; Matschulat 2024; Metec AG 2024).

Thermopneumatic actuators are rarely used. They are based on the expansion of fluids due to heat input. When liquids with low boiling points are utilized even more mechanical expansion is obtainable during evaporation. This type of actuator is called a phase change actuator (PCA). A very simple actuator consisting of a closed volume of fluid with the ability to expand directionally can be constructed (Rai-Choudhury 1997; Matsuoka and Suzumori 2014).

This actuator principle could have many advantages because of its very simple structure and the miniaturization potential it offers. The low number of internal components might also lead to a more cost effective design.

There are multiple publications utilizing this principle to generate mechanical force in soft robotics and other fields (Han et al. 2019; Matsuoka, Kanda, et al. 2016; Niiyama, Rus, and Kim 2014; Boyvat, Daniel M. Vogt, and Robert J. Wood 2019; Sanchez et al. 2020; Uramune et al. 2022). Work has also been done on low-boiling liquids in the context of single braille actuators (Vidal-Verdú, Madueno, and Navas 2005) and even on the use of micro electro-mechanical system technology (Kwon, S. W. Lee, and S. S. Lee 2009).

A common shortcoming of existing publications using thermopneumatic actuators for braille displays is the lack of a satisfactory theoretical model to describe the entire system. Available models typically focus only on specific properties or are based on simplified assumptions regarding the thermodynamics of the system. An example of this is the use of the Clausius-Clapeyron equation in (Vidal-Verdú, Madueno, and Navas 2005), which describes the resulting vapor pressure as a function of the temperature. It does not provide any information on the system state and therefore its own validity; nor does it provide any information on transport properties like heat capacity, which is needed to calculate the energy demand. The exact force generated is often characterized by taking measurements on prototypes subsequently.

The goal of this work is to build a useful model of the complete tactile system with standard engineering tools and to make similar models applicable to other designs based on the same principle. For this purpose a thermopneumatic actuator is built based on the braille standard size with a corresponding model describing its behavior. The prototype is then characterized with measurements to validate the model.

2 METHODS

First a basic design is decided upon to then derive a model of the relevant physical effects and the model's theoretical behavior. A demonstrator is constructed on the same basis. Finally, the two are compared by means of experimental validation in order to derive findings for further decisions in the product development process.

2.1 Actuator Design

The actuator meets the requirements of the braille standard, in particular the footprint of 2.5 mm (braille dot size), and represents a single tactile point. The prototype was designed considering the available tools, scalability and general ability to build up a matrix display based on it. This is a more segmented approach comprising discrete components as compared to other more integrated approaches like in Kwon, S. W. Lee, and S. S. Lee (2009). The system comprises only a few components to facilitate modeling (Figure 1 A).

A wide variety of materials with different state transitions (solid to liquid, liquid to gas) can be used to implement the basic principle (Wilhelm, Richter, and Rapp 2018). We decided to use Novec 7000 (also known as HFE 7000 or RE347mcc) because of its low boiling point, good environmental performance and its successful use in other studies by Nakahara et al. (2017), Hiraki et al. (2020), and Narumi et al. (2020).

The working fluid (F) is held in a copper chamber sealed by adhesive copper foil (Figure 1 component "C"). While using this much copper increases the thermal mass and therefore slows down the actuator, it accelerates heat entry and speeds up the process. The latter was the dominant effect with the first prototypes. Thermopneumatic actuators are often designed with thin, flexible membranes primarily made from latex. Tests showed that this is not an option for this prototype using Novec 7000 as a working fluid: The hygroscopic properties of alkoxy perfluoroalkanes (Koenigsegg 2021) might have led to water from the surrounding air migrating through the thin latex layers. Therefore in our design a polyester foil with aluminum coating is used (M). The metal layer makes this seal gastight. The actuator can only expand by curving the foil instead of mechanically stretching it like an elastic membrane. This solution reaches its lower size limit with the design footprint of 2.5 mm but produces a better seal than a latex membrane. The prototype is completed by a small push rod (P) to transmit the force and motion to the touch surface and a spring (S) to assist with restoring the initial position. The actuator itself does not include a heat generating device for the demonstrator. An external Peltier element with temperature control is used to provide the thermal energy needed (H).

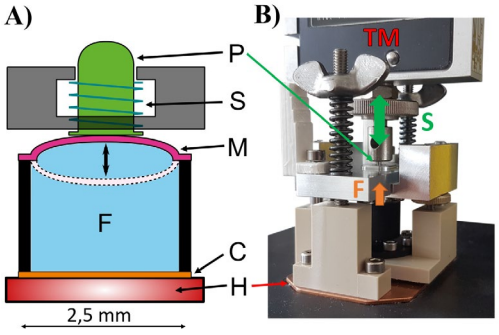


Figure 1. A) Components of actuator prototype from bottom to top: heatplate H, copper foil C, fluid F, membrane/foil M, spring S, pin P
B) Experiment with tensile testing machine TM, stroke S and force F

2.2 Model of the Actuator

Bardaweel et al. (2009) built a detailed model of a PCA by dividing it into discrete elements that are each described by algebraic differential equations. Combining these results in a model describing all relevant internal and external properties of the actuator. This technique of combining differential equations across multiple physical domains in a single system simulation is well suited for thermopneumatic actuators. The structure of our chosen design is shown schematically in Figure 2 A. It illustrates that many variables and various physical effects interact with each other in this actuator.

To make a similar approach for our actuator more generally accessible we opted to use the Modelica (Mattsson and Elmqvist 1997) based commercial simulation tool *SimulationX* (*SimulationX* 2024). With this approach a lot of object-oriented basic elements are predefined and easily accessible. A second general advantage of the approach is that the parameters of the idealized individual components can be well estimated in the concept phase and still be easily adjusted later. Hence, models created this way offer a high degree of flexibility. The most important constructive parameters of the prototype are shown in Table 1. These values change when altering the actuator design. Parameters which are constant in this context like material properties are not listed in the table.

In Figure 2 B the network of thermal components in the system is built from the bottom up. It consists of contact resistances between the components and their respective heat capacities. Constructive parameters from Table 1 in this context are contact areas and the mass of the components to calculate heat capacity. The thermal conductivity at the contact points with thermal grease can be determined from the literature (Lienig and Bruemmer 2017). For the heat transfer into the fluid, boiling and condensation processes would theoretically have to be considered, which affect the resistance value depending on the fluid state. Since available models do not properly reflect a wide chamber, like it is used in our design, and since this variable primarily affects system dynamics, the effect is not being modeled more precisely for the time being. It could be integrated in the model in future iterations.

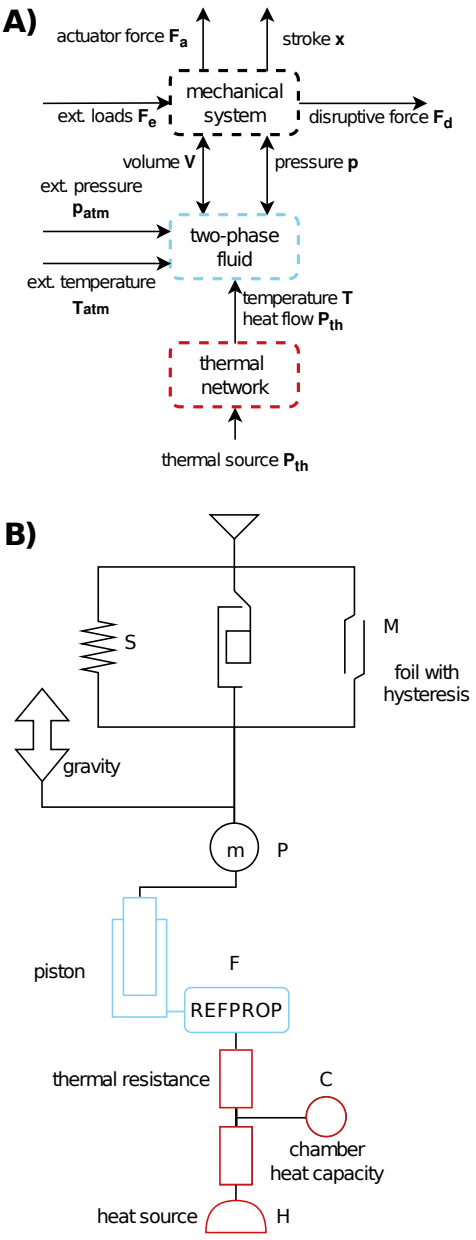


Figure 2. A) Schematic of the different physical domains involved in a PCA and their connections
B) Simplified version of the system model realized with Modelica

The chamber with the working fluid is connected to the thermal network. The thermodynamic properties of the fluid are hard to determine. In principle, the pressure in the fluid can be represented as a function of temperature using the Antoine equation or an empirical function according to the data sheet. With the characteristic points given in the data sheets, the Clausius-Clapeyron equation (Gerlach and Dotzel 2008) describing the vapor pressure can often be evaluated:

$$p(T) = p_k \cdot e^{\frac{L_0}{R}\left(\frac{1}{T_k}-\frac{1}{T}\right)} \quad (1)$$

Here the critical point tuple (T_k, p_k) and heat of evaporation L_0 can be taken from the manufacturer's data sheet. The universal gas constant R is also well documented. The state of the fluid includes additional variables like the ratio of vapor to liquid phase as well as dependent transport variables, such as thermal conductivity and thermal capacity. In the case of very extensively tested fluids (e.g. water), the corresponding relationships can be found in empirical formulae or tables. But generally such thoroughly verified data is not available for every material.

Table 1. Most important constructive model parameters for nominal (n) and tuned (t) model

component	*parameter*	*value*
H,C	radius contact area heater	5.25 mm
H,C	thickness copper bottom	0.05 mm
C	mass fluid chamber	2.1 g
C	radius fluid chamber	1.25 mm
C	volume fluid chamber	23 mm^3
M	movement force foil	36.8 mN
M	stiffness foil (n)	137 N/m
M	*stiffness foil* (t)	*500 N/m*
S	spring constant	585 N/m
P	mass pin	0.5 g
P	maximum stroke	0.6 mm
F	elasticity fluid volume (n)	0.7 mm^3/bar
F	*elasticity fluid volume* (t)	*0.5 mm^3/bar*

Thermodynamic reference models such as CoolProp (Bell et al. 2014), TREND (Span et al. 2020) or REFPROP (Lemmon et al. 2018) offer a solution. Based on the Helmholtz equation of state and appropriate laboratory data, these tools provide both state and transport quantities. They also determine properties that would be unavailable with the above method. REFPROP, which was developed by the National Institute of Standards and Technology (NIST), was used in this work. In order to include the data in the Modelica model, either suitable lookup tables can be generated beforehand and integrated into the corresponding simulation programs or, as in our case, a program interface can be used, which feeds parameters from the current simulation step directly into REFPROP and receives the results. This means that states outside the phase transition region can also be simulated in the fluid section of the model (Figure 2 blue) and energy aspects can be considered by means of the variable material parameters.

The model is completed by the actuator's mechanical components. These consist of basic elements such as springs and masses as well as travel stoppers. Foil behavior (Figure 1 component "M" and Figure 2 B "M") is modeled by a force hysteresis over the deformation path. Hysteresis parameters are obtained from previously performed tests on the physical setup with an external pressure source instead of the fluid. Two parameters in Table 1

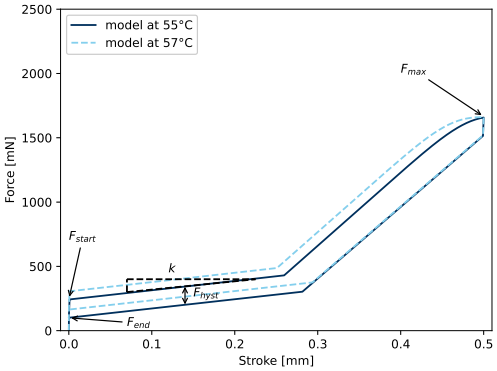


Figure 3. Simulation results with characteristic points on the actuator curve

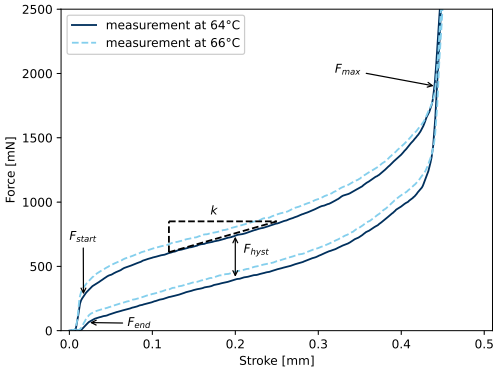


Figure 4. Test results with characteristic points on the actuator curve

are used to represent the hysteresis in the model: the stiffness of the foil, which determines the increase in force, and a constant force offset, which determines the pressure at which the foil begins to change its direction of movement. The latter was adjusted in a tuned model in section 3. For the purpose of this study, only the observed behavior is used as a basis for modeling. One approach for a more precise modeling would be the phenomenological description of the process using two principles: The stiffness is mainly caused by friction of the foil against itself during flipping. The constant force is comparable to the limit load that leads to plate buckling. However, since the foil represents a design trade off to make the chamber gas tight, a more detailed modeling is not considered to be necessary. Further iterations of the actuator design would need a different solution for this component nonetheless.

2.3 Experimental Validation

The experimental procedure can now be virtually reproduced in the actuator model and the measurable variables can be compared with each other. In addition, the model also provides insight into the system states.

First, the actuator is preheated to a constant temperature and completely extended. In a real-world scenario, a person would now apply a tactile force from above when feeling the actuator. In order to simulate this process and to characterize the actuator as appropriately as possible, the measurement is performed identically: The extended actuator is compressed to the lower end stop by a tensile testing machine (Zwick 1120) and then released. Meanwhile, the force is measured as a function of the actuator stroke. The test cycles are very slow with 40 s for 0.5 mm stroke to enable the readjustment of the temperature, which is assumed to be constant. Each measurement is carried out at least twice on the real prototype. The tests are repeated at different temperatures. A characteristic curve based on the model is plotted in Figure 3.

The plot shows that the force at the beginning (F_{start}) is greater by the amount of the hysteresis force F_{hyst} when moving down than when moving out at the end (F_{end}). As long as steam is still present in the system, the pressure is constant according to theory, since the temperature is kept constant. The increase k visible in Figure 3 is due to the mechanical properties of the spring and foil.

When leaving the two-phase region, there is a significant increase of up to F_{max} in the force required because the vapor pressure equation loses its validity at this point and the tensile tester is now acting against a non-compressible fluid with no vapor content. This response can only be depicted by using the thermodynamic reference model.

Curves similar to those yielded by the model are also expected in the trials and can be compared based on the characteristic points. Exemplary test results are shown in Figure 4 and variables corresponding to those in the model are plotted for comparison purposes.

3 Results

The experiments show that the fully assembled prototype fulfills the basic function of an actuator for an active braille display. In addition to the functional tests, the test sequence described above was performed 13 times. Figure 4 shows two of these test runs at 64°C and 66°C, respectively.

There is a constant temperature offset of 9 K between the model values and the measurements. This will be discussed later.

3.1 General Behavior

Some basic actuator characteristics were determined prior to the force measurements. It has been shown that the prototype does not start to extend until a temperature of 62°C is reached. The extension and retraction periods of 31 s and 60 s, respectively, are very long with the heating and cooling unit used. The reason for this is that the overall structure is stiffer then it needs to be, which is mainly due to the foil being used to make the system gas tight. The minimum force of 50 mN - required for probing the point (Vidal-Verdú and Hafez 2007) - is greatly exceeded as well. However, these drawbacks are all a direct consequence of the concrete design implementation. With the

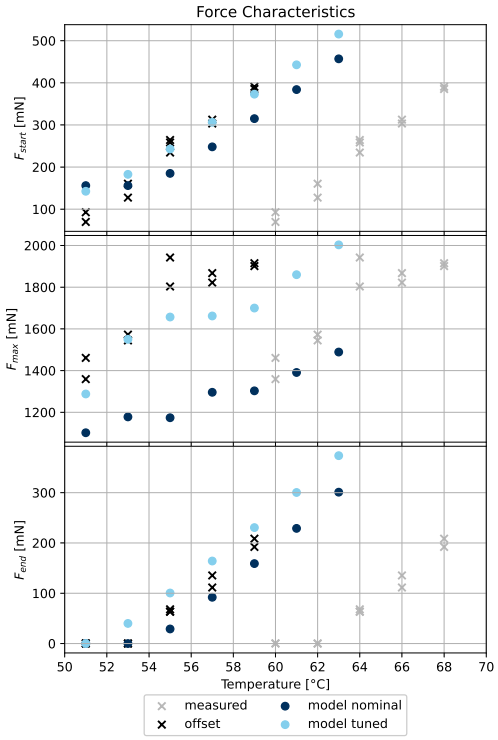


Figure 5. Forces at different temperatures. From top to bottom: force at first contact, maximum force and force after releasing the load

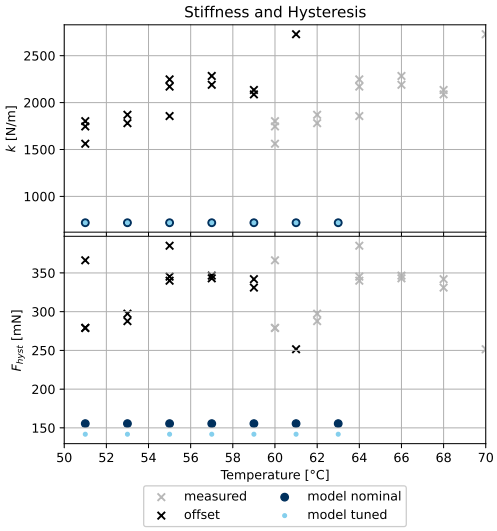


Figure 6. Stiffness (top) and hysteresis (bottom) of the actuator at different temperatures

help of the model, it is possible to simulate a smaller actuator with lower heat capacity and less stiff mechanical components that reaches cycle times of 0.1 s with still sufficiently large tactile force.

Although the energy consumption could not be measured in the demonstrator, the results from the model allow conclusions to be drawn in this regard as well. The selected design boundary conditions, such as overall size, mechanical losses and stiffness, are clearly too large. This results in very high energy consumption. In practical terms, this would mean that approximately 1 W of power would be required per actuator in the extreme case of a display with a 10 Hz refresh rate. This is not realistic for applications with several thousand actuators like high resolution tactile displays. The design of the actuator concept would have to be adapted so that it could realistically be operated at lower power levels.

The curve in Figure 4 can be compared with the modeled sequence in Figure 3 and all characteristic values are identifiable for further examination. The plot also shows that the demonstrator does not quite achieve the targeted maximum stroke of 0.5 mm due to mechanical tolerances. Other differences between the curves can be better analyzed using the characteristic points in the next section.

3.2 Actuator Characteristics

The characteristic points in the force-displacement curve are obtained from the experimental procedure described above. This data is used to compare the model and the experiment. In addition to the model with nominal values from the design process, a tuned model was developed, as a result of deviating system parameters (Table 1). In certain areas this new model better corresponds to the measurement. Figure 5 shows the three force points. The force at the beginning of the cycle F_{start} is well reproduced by both models, taking into account the constant temperature offset. The same applies to the force after releasing the load (F_{end}).

The maximum force F_{max} applied during the experiment is significantly less accurate. The parameter for the stiffness of the fluid chamber was changed in the tuned model to achieve a slightly better match. However, this is still significantly less accurate compared to the other two forces. There are two reasons for this: First, the tensile tester compresses the test setup to a defined maximum force, since the stroke could not be used in a control capacity. As a result, the very high peak forces do not originate from the actuator itself but from the mechanical stop between the actuator and the testing equipment. The exact value is therefore difficult to determine from the test data. Secondly, there are also inaccuracies in the model, which compresses a fluid inside a stiff mechanical structure after leaving the two-phase region. This parameter change causes numerical inaccuracies at peak values, which could not be fine-tuned for every test series.

The parameters of the actuator stiffness k and hysteresis width F_{hyst} in Figure 6 can also not be fitted very well and both show a fundamental unsolved problem in our studies: In the demonstrator the actuator stiffness increases with temperature. This behavior is not implemented in the model, since the exact cause of the effect is still unknown. Accordingly, there are strong deviations between model and measurement here.

4 Discussion

In the simulation results (Figure 3), a clear distinction can be made between the two-phase region with steam inside the fluid chamber and the single-phase region with progressive compression. The sharp bend separating the two regions is assumed to occur at a stroke height of around 0.3 mm, however in the test setup it is not nearly as apparent as in the model. It should be noted, that the rightmost part of the curve in Figure 4 results from the actual test setup, in which the actuator is fully compressed and released again. This represents a well repeatable test case, but does not correspond to the practical application. When the elevated surfaces are sensed by touch, the actuator would only be loaded in the left part of the characteristic curve. The relevant characteristic values F_{start} and F_{end} are again well modeled there.

The continuous transition between the two regions in the trial could explain why the stiffness k of the test actuator is higher than in the model. A temperature dependency can be at least partially explained: In the two-phase region, the stiffness depends only on the mechanical components of the system, since the pressure in the actuator remains constant. However, this does not apply to further compression with purely liquid fluid. Changes can occur here since the mechanical properties of the fluid change in relation to temperature. Whether these effects are a sufficiently accurate quantitative explanation for the differences needs further investigation.

4.1 Temperature Offset

The temperature during the experiment was only set on the heater and actively regulated there. It could not be measured inside the actuator itself due to its compact design. It is therefore conceivable that the temperature inside the actuator was slightly lower than inside the heating plate because the device was air cooled. In addition, it was shown earlier that the complete actuator is overall stiffer than the model predicts. This means that the real actuator needs a higher temperature to exert the desired force. Both effects combined can explain the temperature difference between model and experiment.

4.2 Model Inaccuracies

Two simplifications were made in the design of the model. On one hand, the expansion within an ideal cylindrical piston is assumed. In this case, the calculated volume does not exactly match that of the actuator because the foil bulges slightly and does not form perfect edges. On the other hand, the boiling and condensation processes are not accurately modeled. The heat transfer between chamber and fluid is thus only a simplification, which impacts actuator dynamics. However, the impact of both effects on the system should be minimal and does not explain the increase in actuator stiffness as a function of temperature. Enhancing these parts of the model therefore improves the overall model behavior only slightly.

4.3 Future Work

This work shows how a Modelica based simulation model can be used to evaluate an idea during the product development process. Apart from this, the actually implemented structure of the actuator prototype still has room for further improvement that should be addressed in future work. The modeling inaccuracies that occur are mainly related to these components.

The foil used as a membrane was selected to replace latex membranes, which repeatedly showed leaks. The foil for the 2.5 mm diameter prototype actuator is on the limit of usability because of its stiffness. This caused the very high operating temperature and large actuator forces, which are significantly greater than required. A better solution with good sealing properties when used in combination with fluorine-based fluids is needed for future setups.

In addition, the setup should be equipped with more sensors for pressure and temperature monitoring inside the actuator. Even if this means that the targeted size of 2.5 mm is probably not feasible for an experimental setup, this disadvantage is outweighed by the significantly better verifiability of the model.

Furthermore it would be worth considering whether a different fluid could be utilized. The coolant used has a high evaporation enthalpy, due to its intended use for absorbing heat, which is also responsible for the high energy consumption. It might be more advantageous to choose a fluid with a higher boiling point but a lower heat of evaporation. The temperature of the actuator must then be kept close to the boiling point by means of insulation. Unfortunately, substances with similarly low boiling points and at the same time low enthalpy of vaporization, such as isopentane, are often toxic (Chiba and Oshida 1991).

5 Conclusions

It was shown that the system simulation approach in combination with thermodynamic reference models can be used to simulate PCAs for tactile applications. The structure is intuitive due to the existing object-oriented libraries and can be adapted well to further work. They therefore form a useful basis for product development processes when evaluating solution concepts.

A model of an actuator was successfully built and experimentally validated. This makes transient simulations possible. It can be easily adapted to modified designs. The challenges that still exist are largely due to the specific foil used in the actuator.

The model based on Modelica with libraries from *SimulationX* can also be extended to include the previously neglected effects. To do this, it would make sense to first carry out further tests with more measurement options and thus clarify the still unexplained effects in regard to the actuator stiffness. However, due to the energy consumption observed on the model, the actuator concept was not pursued further for the specific planned application of a haptic braille display.

Acknowledgements

We would like to thank the NIST and Dr. Eric Lemmon for providing the fluid model file used with REFPROP as a part of the system model.

References

Baldwin, Mark S. et al. (2017). "The Tangible Desktop". In: *ACM Transactions on Accessible Computing* 10.3, pp. 1–28. DOI: 10.1145/3075222.

Bardaweel, H. K. et al. (2009). "Characterization and modeling of the dynamic behavior of a liquid–vapor phase change actuator". In: *Sensors and Actuators A: Physical* 149.2, pp. 284–291. DOI: 10.1016/j.sna.2008.11.020.

Bell, Ian H. et al. (2014). "Pure and pseudo-pure fluid thermophysical property evaluation and the open-source thermophysical property library CoolProp". In: *Industrial & engineering chemistry research* 53.6, pp. 2498–2508. DOI: 10.1021/ie4033999.

Boyvat, Mustafa, Daniel M. Vogt, and Robert J. Wood (2019). "Ultrastrong and High-Stroke Wireless Soft Actuators through Liquid–Gas Phase Change". In: *Advanced Materials Technologies* 4.2, p. 1800381. DOI: 10.1002/admt.201800381.

Caldwell, D. G., N. Tsagarakis, and C. Giesler (1999). "An integrated tactile/shear feedback array for stimulation of finger mechanoreceptor". In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. Vol. 1, 287–292 vol.1. DOI: 10.1109/ROBOT.1999.769991.

Chiba, S. and S. Oshida (1991). "Metabolism and toxicity of n-pentane and isopentane". In: *Nihon Hoigaku Zasshi - The Japanese Journal of Legal Medicine* 45.2, pp. 128–137. PMID: 1920919.

Gerlach, Gerald and Wolfram Dotzel (2008). *Introduction to microsystem technology: a guide for students*. John Wiley & Sons. Chap. 7.2 Transducers for Sensors and Actuators. ISBN: 9780470058619.

Han, Jie et al. (2019). "Untethered Soft Actuators by Liquid–Vapor Phase Transition: Remote and Programmable Actuation". In: *Advanced Intelligent Systems* 1.8, p. 1900109. DOI: 10.1002/aisy.201900109.

Hiraki, Takefumi et al. (2020). "Laser Pouch Motors: Selective and Wireless Activation of Soft Actuators by Laser-Powered Liquid-to-Gas Phase Change". In: *IEEE Robotics and Automation Letters* 5.3, pp. 4180–4187. DOI: 10.1109/LRA.2020.2982864.

Howe, R. D., D. A. Kontarinis, and W. J. Peine (1995). "Shape memory alloy actuator controller design for tactile displays". In: *Proceedings of 1995 34th IEEE Conference on Decision and Control*. Vol. 4, pp. 3540–3544. DOI: 10.1109/CDC.1995.479133.

Koenigsegg, Christian (2021-10-13). "Liquid heat transfer mixture and use thereof". European pat. EP3757190B1. Alpraaz AB. URL: http://v3.espacenet.com/textdoc?IDX=EP3757190.

Kwon, Hyuk-Jun, Seok Woo Lee, and Seung S. Lee (2009). "Braille dot display module with a PDMS membrane driven by a thermopneumatic actuator". In: *Sensors and Actuators A: Physical* 154.2, pp. 238–246. DOI: 10.1016/j.sna.2008.10.002.

Lemmon, Eric W. et al. (2018). "NIST standard reference database 23: reference fluid thermodynamic and transport properties-REFPROP, Version 10.0, National Institute of Standards and Technology". In: *Standard Reference Data Program, Gaithersburg*.

Lienig, Jens and Hans Bruemmer (2017). *Fundamentals of electronic systems design*. Springer. Chap. 5.4.2 Thermal Interface Materials. ISBN: 9783319558400.

Matschulat, Gunnar (2024). *ABTIM*. URL: http://www.abtim.com/ (visited on 2024-04-26).

Matsuoka, Hiroki, Takefumi Kanda, et al. (2016). "Development of a rubber soft actuator driven with gas/liquid phase change". In: *International Journal of Automation Technology* 10.4, pp. 517–524. DOI: 10.20965/ijat.2016.p0517.

Matsuoka, Hiroki and Koichi Suzumori (2014). "Gas/liquid phase change actuator for use in extreme temperature environments". In: *International Journal of Automation Technology* 8.2, pp. 140–146. DOI: 10.20965/ijat.2014.p0140.

Mattsson, Sven Erik and Hilding Elmqvist (1997). "Modelica - An International Effort to Design the Next Generation Modeling Language". In: *IFAC Proceedings Volumes* 30.4, pp. 151–155. DOI: https://doi.org/10.1016/S1474-6670(17)43628-7.

Metec AG (2024). *Hyperflat*. URL: https://www.metec-ag.de/downloads/hyperflat-flyer-komplett.pdf (visited on 2024-04-26).

Mirvakili, Seyed M. et al. (2021). "Solar-Driven Soft Robots". In: *Advanced Science* 8.8, p. 2004235. DOI: 10.1002/advs.202004235.

Nakahara, Kenichi et al. (2017). "Electric phase-change actuator with inkjet printed flexible circuit for printable and integrated robot prototyping". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1856–1863. DOI: 10.1109/ICRA.2017.7989217.

Narumi, Koya et al. (2020). "Liquid Pouch Motors: Printable Planar Actuators Driven by Liquid-to-Gas Phase Change for Shape-Changing Interfaces". In: *IEEE Robotics and Automation Letters* 5.3, pp. 3915–3922. DOI: 10.1109/LRA.2020.2983681.

Niiyama, Ryuma, Daniela Rus, and Sangbae Kim (2014). "Pouch Motors: Printable/inflatable soft actuators for robotics". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6332–6337. DOI: 10.1109/ICRA.2014.6907793.

Rai-Choudhury, Prosenjit (1997). *Handbook of microlithography, micromachining, and microfabrication: microlithography*. Vol. 39. SPIE press. ISBN: 0-8194-2378-5.

Sanchez, Vanessa et al. (2020). "Smart thermally actuating textiles". In: *Advanced Materials Technologies* 5.8, p. 2000383. DOI: 10.1002/admt.202000383.

Sarakoglou, I., N. Tsagarakis, and D. G. Caldwell (2005). "A portable fingertip tactile feedback array - transmission system reliability and modelling". In: *First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics Conference*, pp. 547–548. DOI: 10.1109/WHC.2005.17.

SimulationX (2024). URL: https://www.esi-group.com/products/simulationx (visited on 2024-04-26).

Span, R. et al. (2020). *TREND. Thermodynamic Reference and Engineering Data 5.0*.

Tieman, Frans J. and Kees Zeehuisen (1988-07-19). "Tactile relief display device and method for manufacture it". U.S. pat. 4758165.

Uramune, Ryusei et al. (2022). “HaPouch: A Miniaturized, Soft, and Wearable Haptic Display Device Using a Liquid-to-Gas Phase Change Actuator”. In: *IEEE Access* 10, pp. 16830–16842. DOI: 10.1109/ACCESS.2022.3141385.

Velazquez, Ramiro et al. (2005). “A low-cost highly-portable tactile display based on shape memory alloy micro-actuators”. In: *IEEE Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2005.* IEEE, 6–pp. DOI: 10.1109/VECIMS.2005.1567577.

Vidal-Verdú, Fernando and Moustapha Hafez (2007). “Graphical tactile displays for visually-impaired people”. In: *IEEE transactions on neural systems and rehabilitation engineering : a publication of the IEEE Engineering in Medicine and Biology Society* 15. DOI: 10.1109/TNSRE.2007.891375.

Vidal-Verdú, Fernando, Manuel J. Madueno, and Rafael Navas (2005). “Thermopneumatic actuator for tactile displays and smart actuation circuitry”. In: *Smart Sensors, Actuators, and MEMS II*. Ed. by Carles Cane, Jung-Chih Chiao, and Fernando Vidal Verdu. SPIE Proceedings. SPIE, pp. 484–492. DOI: 10.1117/12.607603.

Wagner, C. R., S. J. Lederman, and R. D. Howe (2002). “A tactile shape display using RC servomotors”. In: *Proceedings 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. HAPTICS 2002*, pp. 354–355. DOI: 10.1109/HAPTIC.2002.998981.

Wilhelm, E. (2015). “Entwicklung eines mikrofluidischen Brailledisplays”. PhD thesis. Karlsruhe. ISBN: 978-3-7315-0385-9.

Wilhelm, E., C. Richter, and B. E. Rapp (2018). “Phase change materials in microactuators: Basics, applications and perspectives”. In: *Sensors and Actuators A: Physical* 271, pp. 303–347. DOI: 10.1016/j.sna.2018.01.043.

Dynamic Modeling Methodology for Near Isothermal Compressor

Haopeng Liu[1] Vikrant Aute[1] Yunho Hwang[1] Cheng-Yi LEE[1]
Jan MUEHLBAUER[1] Lei Gao[1]

[1]Center for Environmental Energy Engineering, Department of Mechanical Engineering, University of Maryland, College Park, USA,
{hliu1220, vikrant, yhhwang, muehlie, cylee, leigao}@umd.edu

Abstract

Compressors are the vital component of the vapor compression systems and account for the majority of energy consumption. Developing appropriate controllers or optimizing compressor design can significantly reduce the carbon emissions. The isothermal compressor combines the compressor chamber and gas cooler, using the liquid piston to compress the working fluid for near-isothermal compression. This methodology can reach up to 30% energy saving compared to the traditional isentropic compression work. This paper leverages the CEEE Modelica Library (CML) to demonstrate a detailed isothermal compressor model that captures the near-isothermal compression process of transcritical carbon dioxide (CO_2) cycle. The model uses the real experimental data as the boundary conditions, and the relevant component-level experimental validation was carried out by using a prototype with 1-ton nominal capacity. The results proved the accuracy of the dynamic model (7.5% relative error for chamber pressure and 0.74 K deviation for chamber temperature), and provide a guideline for designing the isothermal compressor chamber. Finally, the modeling for the isothermal compression cycle is ongoing and the filed is still in its infancy.

Keywords: Isothermal Compressor, Transcritical CO_2 Cycle, Dynamic Modeling

1 Introduction

Vapor compression system (VCS) are extensively utilized in heating, ventilation, and air conditioning (HVAC) area, which together account for more than 30% of electricity generated in the U.S. (EIA, 2022). The compressor, responsible for circulating refrigerant and transferring heat, consumes the majority of this electricity. Therefore, measures to improve the system energy efficiency, especially the innovative design of compressor, can significantly reduce the carbon footprint and boost the resilient energy economy.

The ideal thermodynamic cycle (Carnot cycle) defines the upper limit on the efficiency of refrigeration system in creating a temperature difference through the application of work to the system. In reality, it's not possible to build such thermodynamically reversible engine and the real engines that even operate along the Carnot cycle style (isothermal expansion / isentropic expansion / isothermal compression / isentropic compression) are rare. However, the isothermal compression can bring the system close to the Carnot cycle efficiency, and the related technology have achieved breakthrough progress recently, especially with the widespread adoption of compressed air energy storage (CAES) that driven by the increasing penetration of renewable energy sources (Kim et al., 2022). Given that one of the biggest problems come with CAES is low energy efficiency, i.e., the traditional CAES systems lose energy due to heat generated during the compression, which cannot be fully recovered. A considerable number of studies have explored achieving isothermal compression in CAES applications, including water injection (Patil et al., 2020, Odukomaiya et al., 2016), chamber shape optimization (Zhang et al, 2016) and chamber packing with inserted material (Yan et al, 2015, Saadat et al, 2012).

On the other hand, CAES systems emit greenhouse gases, which pose challenges to the goals of reducing greenhouse gas emissions. In view of long-term environmental safety, one potential substitute refrigerant is carbon dioxide (CO_2), a natural refrigerant that has negligible impact on climate change, which is environmentally benign, non-toxic, and non-explosive. In addition, recent advancements in system design and manufacturing improvements make it possible to achieve high pressure required for CO_2 transcritical operation, and the CO_2 can deliver much higher heat rejection through sensible cooling to regain efficiency when it's compressed beyond critical point (31.1 °C, 7.38 MPa). The CO_2 transcritical cycles have many different system configurations for different applications (Sarkar et al., 2004, Fernandez et al., 2010), with ongoing tightening environmental regulations, further theoretical research is needed to explore the potential for enhancing the energy efficiency of transcritical CO_2 systems.

The CEEE Modelica Library (CML) is a comprehensive Modelica library developed by the Center for Environmental Energy Engineering (CEEE), University of Maryland, College park. It is designed for transient simulation of extensive thermal system configurations and HVAC applications. The CEEE can assist in gaining a deeper understanding of thermodynamic systems. The CML is scalable, allowing users to virtually assess and optimize the VCS's performance. Two features in our implementation are tailored for modeling of isothermal compressor model in CML: (1) Using the liquid piston to compress the working fluid (CO_2) within the heat exchanger-based chamber to enhance the heat transfer. (2) The compression chambers can realize the double-acting mechanism, allowing compression and suction processes to occur simultaneously. The modeling details are elaborated in Section 2, while Section 3 covers the information about our experimental setup and the relevant experimental validation. Finally, the conclusions and future work are summarized in Section 4.

2 Modeling Methodology

In our test unit, a two-chamber isothermal compressor design with shared liquid pump is adopted. Each compression chamber is based on the plate heat exchanger (PHX), with the secondary fluid (water) provided by the air-cooled radiator to cool down the compressed refrigerant. Additionally, both the residual gas cooler and suction line heat exchanger are PHXs as well, utilized to ensure the rated cooling capacity of 1 ton. The piston accumulator is used as the storage component to regulate the system charge level and mitigate pressure fluctuation. The electronic expansion valve (EXV) controls the downstream pressure and mass flow rate, while the electric heater acts as the evaporator to control the refrigerant state at the suction side of the isothermal compressor by regulating its heat load.

The schematic diagram of the test facility with sensors installed is shown in Figure 1. Sensors are in place to measure key operation variables such as pressures, temperatures and mass flow rates, etc. In Figure 1, letters 'T' and 'P' represent the temperature and pressure measurement via thermocouples and pressure transducers, respectively, which can provide us with information about the refrigerant state for different components. The hydraulic directional control valve is installed to facilitates the switching of flow direction when the isothermal chamber completes the compression/ suction stroke, and the oil level reaches the upper/ lower oil level sensor. The check valves were utilized to minimize the dead volume in each compression cycle and the mass flow meter is uilized to measure the suction side mass flow rate of the isothermal compressor.

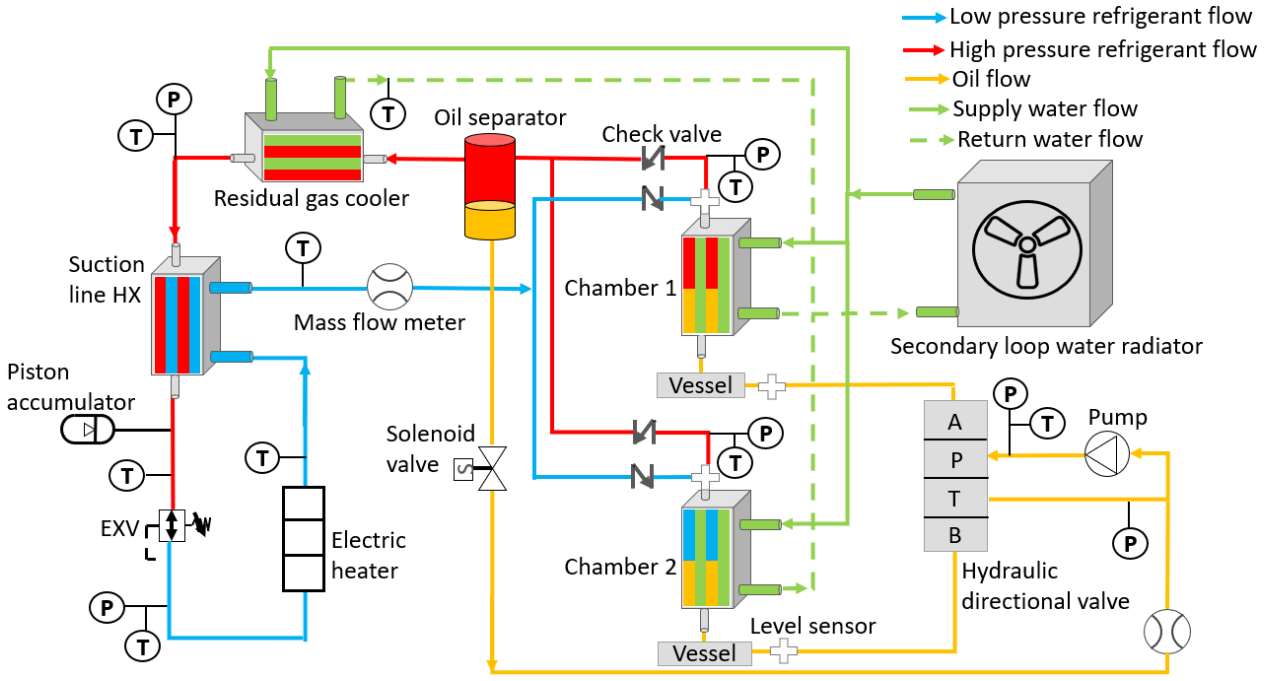


Figure 1. Sensor instrumentation diagram for the test unit.

The Modelica interface of the PHX-based isothermal compressor is depicted in Figure 2 and following assumptions are made for the modeling:

1. The refrigerant side boundary conditions (e.g., pressure and enthalpy) of the isothermal compressor were provided based on the experimental data, while the water-side boundary

conditions were assumed to be constant for simplification.

2. The liquid piston model is simplified as the input to the isothermal compressor, which compresses or suck in the refrigerant within the chamber under the given volumetric flow rate and time period.
3. For each channel of the PHX-based isothermal compressor, the geometric details and the flow conditions (e.g., mass flow rate and temperature) for both primary and secondary fluid were assumed to be the same. Therefore, the individual channel was selected for the modeling, and the corresponding results were multiplied by half the total number of plates to derive the overall component results.
4. The process of solubility / degassing of CO_2 in oil during the compression/ suction is too complex and can affect the charge estimation within the isothermal compressor. Therefore, for simplification, the oil is assumed to be mineral oil with no solubility. Additionally, the model does not consider the heat transfer between the oil and CO_2.

The geometric details of the PHX are shown in Table 1.

Table 1. Geometric details of the PHX.

Parameter	*Value*
Port to port length (mm)	329
Width (mm)	119.5
Corrugation depth (mm)	1.55
Area enlargement factor	1.24
Total number of plate	50
Diamter of port (mm)	23.5

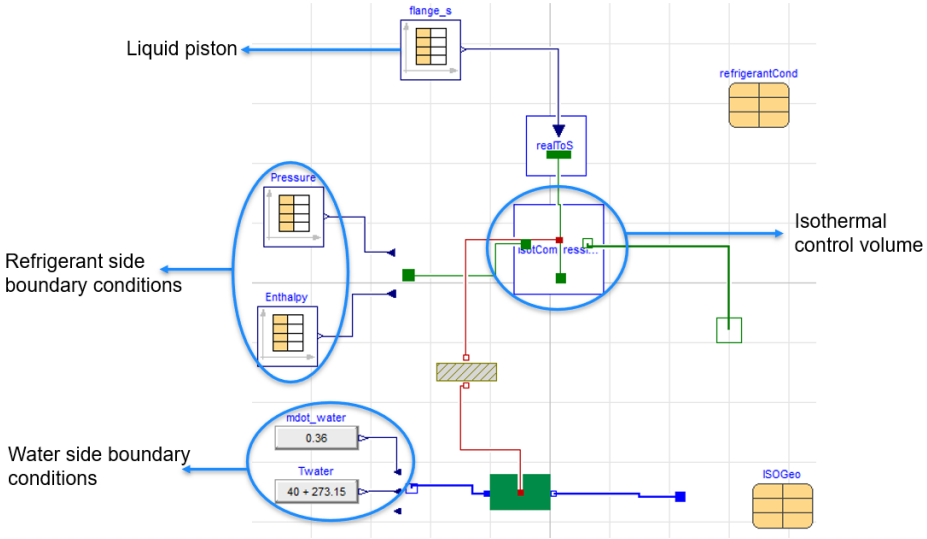


Figure 2. Modelica interface of the isothermal compressor model.

The conservation differential equations for the refrigerant energy, mass and tube wall energy for the isothermal compressor are given in Equation 1 to Equation 3.

$$\dot{U} = \dot{m}_{in}h_{in} - \dot{m}_{out}h_{out} - \alpha_r A\left(T_r - T_w\right) \tag{1}$$

$$\dot{m}_e = \dot{m}_{in} - \dot{m}_{out} \tag{2}$$

$$\dot{E} = C_{th,w}\dot{T}_w = \alpha_r A\left(T_r - T_w\right) - \alpha_{water} A\left(T_w - T_{water}\right) \tag{3}$$

where U is the refrigerant internal energy; $\dot{m}_{in}$ and $\dot{m}_{out}$ represent the inlet and outlet refrigerant mass flow rates of the chamber, respectively; h_{in} and h_{out} represent the inlet and outlet refrigerant enthalpy of the chamber, respectively; T_r, T_w and T_{water} are the temperature of refrigerant, plate wall and water (secondary fluid); $\dot{m}_e$ is the time derivative of refrigerant mass held in the chamber; E is the plate wall energy and $C_{th,w}$ denotes its thermal capacitance; A is the heat transfer area; α_r and α_{water} are the refrigerant side and water side heat transfer coefficient (HTC), respectively;

Based on the relationship between the refrigerant internal energy and enthalpy

$$u = h - \frac{P}{\rho} \tag{4}$$

where u is the refrigerant specific internal energy and ρ is the refrigerant density, the time derivative of the refrigerant internal energy U in Equation 1 can be decomposed into terms of time derivatives of the pressure

and enthalpy using the chain rule:

$$\dot{U} = \left(\frac{\partial \rho}{\partial P} Vh - V\right)\dot{P} + \left(\frac{\partial \rho}{\partial h} Vh + \rho V\right)\dot{h} + (\rho h - P)\dot{V} \quad (5)$$

where V is the chamber volume of isothermal compressor. Similarly, the mass balance equation of Equation 2 can be rewritten as

$$\dot{m}_e = V\left(\frac{\partial \rho}{\partial P}\dot{P} + \frac{\partial \rho}{\partial h}\dot{h}\right) + \rho\dot{V} \quad (6)$$

Equation 5 and 6 allow the reformulation of the governing equations using the pressure P and enthalpy h as the state variables. The resultant state-space governing equations for each channel of PHX-based isothermal compressor are formulated as follows

$$\begin{bmatrix} V_i\frac{\partial \rho_i}{\partial P_i}h_i - V_i & V_i\frac{\partial \rho_i}{\partial h_i}h_i + V_i\rho_i & 0 & \rho_i h_i - P_i \\ V_i\frac{\partial \rho_i}{\partial P_i} & V_i\frac{\partial \rho_i}{\partial h_i} & 0 & \rho_i \\ 0 & 0 & C_{th,w} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{P}_i \\ \dot{h}_i \\ \dot{T}_{w,i} \\ \dot{V}_i \end{bmatrix} = \begin{bmatrix} \frac{\dot{m}_{in}}{N}h_{in} - \frac{\dot{m}_{out}}{N}h_i - \alpha_{r,i}A_i\left(T_{r,i} - T_{w,i}\right) \\ \frac{\dot{m}_{in}}{N} - \frac{\dot{m}_{out}}{N} \\ \alpha_{r,i}A_i\left(T_{r,i} - T_{w,i}\right) - \alpha_{water,i}A_i\left(T_{w,i} - T_{water,i}\right) \\ -\frac{V_m}{N} \end{bmatrix} \quad (7)$$

where i denotes the i^{th} channel; V_m is volumetric flow rate of the liquid piston and N is half of the total number of plates.

As shown in Figure 1, a flow meter was installed at the hydraulic part to measure the volumetric flow rate of oil, and the experimental results were shown in the left part of Figure 3. For modeling purposes, simplifications were thereby implemented: in each process (compression/ suction), the volumetric flow rate was initially kept constant for the first 6 seconds, and then linearly declined throughout the remaining period. Note that the volumetric flow rate profile for each chamber takes mirror relationship.

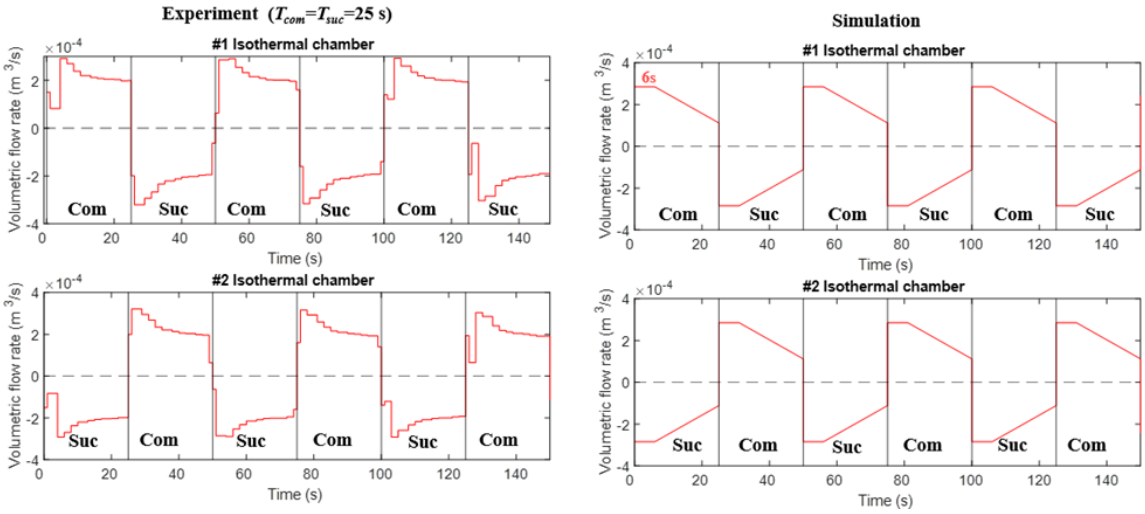


Figure 3. Volumetric flow rate of liquid piston for each compression chamber ("Com": compression, "Suc": suction).

3 Experimental Validation

Figure 4 shows the test rig of isothermal compressor system with the rated cooling capacity of 1 ton. As forementioned, the two PHXs-based compression chambers were utilized to implement the double-acting mechanism. This design ensures each of the compression chamber undergoes either the suction or compression process, with both processes having the same duration by using the level sensor to monitor the oil level. The secondary loop of PHX is managed by the water radiator with fan to release the heat into ambient. The oil separator is introduced to separate the oil dissolved in the refrigerant during the compression process and return it to the liquid

pump. This pump provides high-pressure liquid to the compression chamber and is designed to achieve high volumetric efficiency.

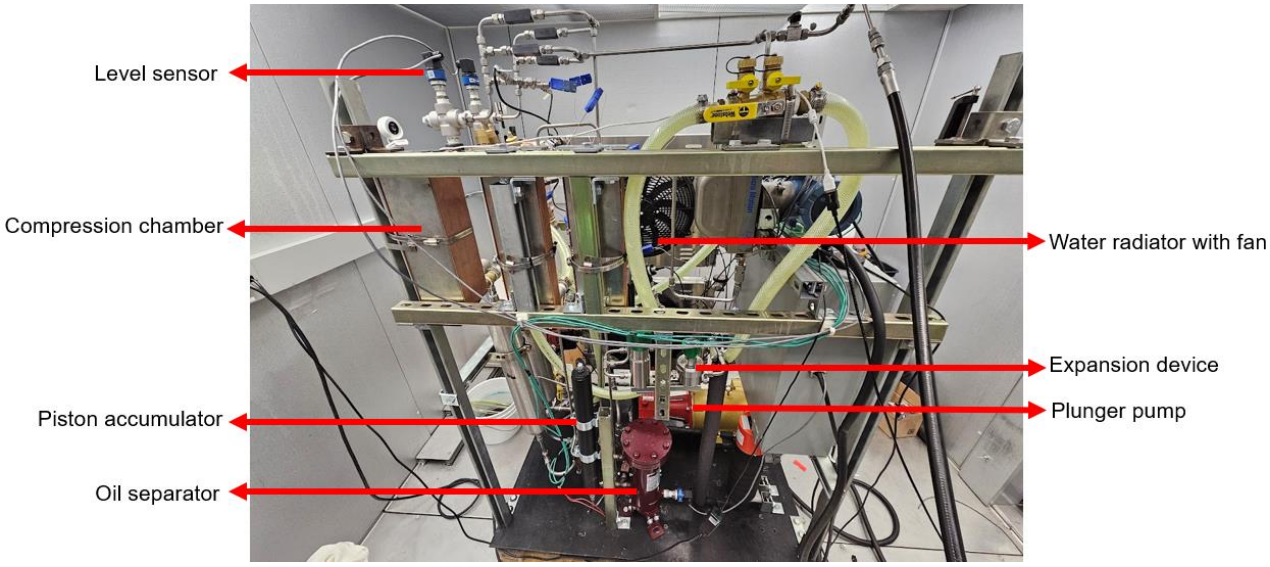


Figure 4. Experimental facility of isothermal compressor cycle.

The experimental validation results for the isothermal compression system model were depicted in Figure 5. Overall, the established model can accurately capture the pressure and temperature behavior of test rig, for both dynamic and steady state characteristics, with 7.5% relative error for the chamber pressure and 0.74 K deviation for the chamber temperature. However, the pressure comparison revealed that the simulation curve appears "steeper" than the experimental counterpart at the end of the compression process (delivery of CO_2), which probably due to the liquid piston model didn't accurately reflect the real volumetric flow rate at that stage. Therefore, one of the future tasks should be the calibration of liquid piston model. Furthermore, during the suction period, there is noticeable deviation between the simulation and experiment. This is mainly because the model didn't account for the degassing process, where quite amount of CO_2 is released from the oil due to the pressure drop.

As for the temperature comparison, the experimental results show a sudden drop at the beginning of the suction process. This drop is due to the expansion of the remaining CO_2 in the isothermal chamber. However, the model did not capture this behavior, likely because the sensor is attached to the surface of the compression chamber, and the model did not account for the thermal mass of the chamber wall.

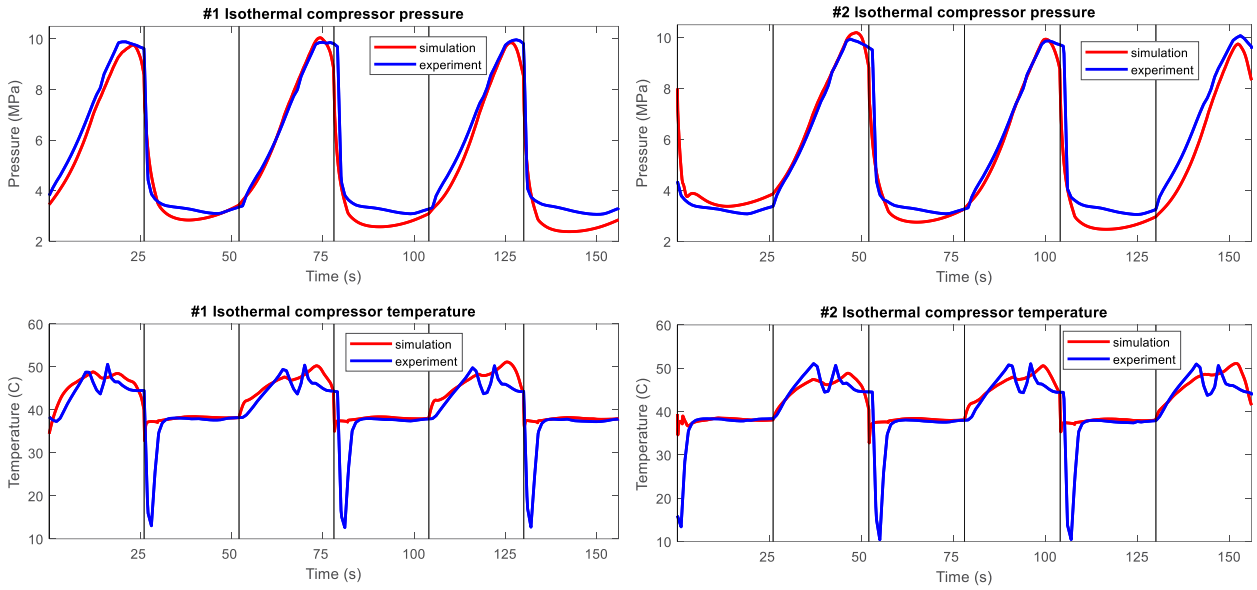


Figure 5. Comparisons of pressure and temperature between experiment and simulation for each compression chamber.

4 Conclusions

In this paper, a dynamic model for the double-acting isothermal compressor based on the CML is established. The isothermal compressor is coupled with liquid piston to serve the dual purpose as a heat exchanger to cool down the refrigerant and achieve the isothermal compression within the chamber. Unlike the conventional compressor model which evolve on much faster time scales than the heat exchanger dynamics and typically established as quasi-steady state model, the compression process for the isothermal compressor model requires much longer time to dissipate the sufficient heat. To demonstrate the accuracy, the experimental tests were carried out based on the 1-ton refrigeration system, where the boundary conditions of experimental data were fed into the model. The results validated that the established model can accurately capture both the steady-state and dynamic behaviors of test rig (7.5% relative error for chamber pressure and 0.74 K deviation for chamber temperature).

To further improve the model accuracy, future work include the calibration of liquid piston model and the incorporation of the CO_2 solubility correlation in the oil. The developed model will be used to guide the future design (e.g., shape optimization of chamber) and prototyping.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Building Technologies Office Award Number DE-EE0008674.

References

EIA (2022). Electricity data. 2022. https://www.eia.gov/energyexplained/electricity/use-of-electricity.php.

Kim, T., Lee, C. Y., Hwang, Y., & Radermacher, R. (2022). A review on nearly isothermal compression technology. International Journal of Refrigeration, 144, 145-162.

Patil, V. C., Acharya, P., & Ro, P. I. (2020). Experimental investigation of water spray injection in liquid piston for near-isothermal compression. Applied energy, 259, 114182.

Odukomaiya, A., Abu-Heiba, A., Gluesenkamp, K. R., Abdelaziz, O., Jackson, R. K., Daniel, C., ... & Momen, A. M. (2016). Thermal analysis of near-isothermal compressed gas energy storage system. Applied energy, 179, 948-960.

Zhang, C., Li, P. Y., Van de Ven, J. D., & Simon, T. W. (2016). Design analysis of a liquid-piston compression chamber with application to compressed air energy storage. Applied thermal engineering, 101, 704-709.

Yan, B., Wieberdink, J., Shirazi, F., Li, P. Y., Simon, T. W., & Van de Ven, J. D. (2015). Experimental study of heat transfer enhancement in a liquid piston compressor/expander using porous media inserts. Applied energy, 154, 40-50.

Saadat, M., Li, P. Y., & Simon, T. W. (2012, June). Optimal trajectories for a liquid piston compressor/expander in a compressed air energy storage system with consideration of heat transfer and friction. In 2012 American Control Conference (ACC) (pp. 1800-1805). IEEE.

Sarkar, J., Bhattacharyya, S., & Gopal, M. R. (2004). Optimization of a transcritical CO2 heat pump cycle for simultaneous cooling and heating applications. International Journal of Refrigeration, 27(8), 830-838.

Fernandez, N., Hwang, Y., & Radermacher, R. (2010). Comparison of CO2 heat pump water heater performance with baseline cycle and two high COP cycles. International Journal of Refrigeration, 33(3), 635-644.

Fluid Property Functions in Polar and Parabolic Coordinates

Scott A. Bortoff, Christopher R. Laughman, Vedang Deshpande and Hongtao Qiao [1]

[1]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA {bortoff, laughman, deshpande, qiao}@merl.com

Abstract

This paper presents two methods for reallizing fluid property functions in Modelica simulation models. Each makes use of a coordinate transformation that aligns one coordinate with the saturation curve. This provides for a precise representation of the fluid property function at the saturation curve, and for connected domains of interest including the liquid, vapor, supercritical and two-phase regions. Both approaches make use of spline function approximation in the aligned coordinates, and are numerically efficient, well conditioned, and allow for efficient calculation of derivatives up to any desired order that are precise up to processor numerical tolerance.

Keywords: thermofluid models, fluid properties

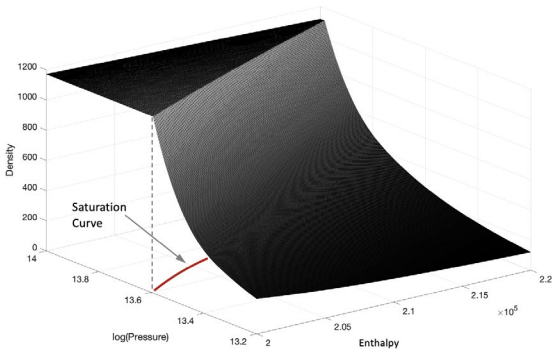


Figure 1. Density of R410A as function of h and $\log(P)$, showing the saturation curve (red).

1 Introduction

Fluid property functions relate fluid property variables such as temperature, pressure, density, etc., to one another. For a fluid of fixed composition in thermodynamic equilibrium, all fluid property variables can be calculated as a function of two independent variables: a mixture variable and a second variable. Pressure P and specific enthalpy h are often chosen for vapor compression models, but other combinations are also possible (Bejan, 1993). Fluid property functions are critical for thermofluid model simulation, and must be implemented in an accurate, computationally efficient manner. For some applications, fluid property function evaluations consume more than 70% of simulation time (Aute and Radermacher, 2014).

Mathematically, the domain of a fluid property function is the span of the two independent fluid property variables. For many thermofluid systems, such as vapor compression cycles, the domain includes values of the two independent fluid property variables that correspond to more than one of the fluid's states, such as the vapor state, the supercritical state, or the two-phase state. The boundary between the liquid region and two-phase region in the domain is the liquid saturation curve, and the boundary between the vapor region and two-phase region in the domain is the vapor saturation curve. These curves intersect smoothly at the critical point of the fluid, and their union is referred to as the saturation curve.

The saturation curve is distinguished because its image under a fluid property function is not smooth. The fluid property function is continuous (C_0), but not continuously differentiable (C_1), for all points on the saturation curve. For all other points in the domain, the fluid property function is a smooth (C_n) function of the two fluid property variables, for some $n \geq 1$, as shown in Figure 1, which plots density ρ as a function of specific enthalpy h and pressure P for R410A. Modelica models of thermofluid systems often make use of derivatives of a fluid property function with respect to the two fluid property variables, and it is important to compute these accurately, especially near the saturation curve.

Several approaches for computing fluid properties may be found in the literature. Some are based upon using the Helmholtz (or Gibbs) free energy equation (Span, 2000). Any fluid property of interest may be numerically computed by solving these equations using iterative methods, typically Newton's method of root finding. These methods are realized in available software such as REFPROP (Lemmon et al., 2018) and CoolProp (Bell et al., 2014), and also realized in the HelmholtzMedia Modelica library (Thorade and Saadat, 2012, 2013). While these methods are general and accurate, they tend to be computationally expensive for use in simulation models, especially for large models with long simulation times. Furthermore, iterative algorithms include a stopping criteria, and therefore small errors are introduced into the computed fluid property values. If these values are numerically differentiated, which may be done by a simulation tool to compute a Jacobian, for example, then small errors can be amplified to the point of being unacceptably large, especially in the region near the saturation curve. Moreover, these iterative methods can fail to converge for certain values of the two independent fluid property variables.

Other approaches for calculating fluid property functions for use in simulation include Taylor's series approximations or splines. Aute, et al describe a method using Chebyshev polynomials that is built from data obtained from REFPROP (Aute and Radermacher, 2014). This method demonstrates a significant speedup over standard iterative methods, but does not enforce consistency between the properties and their derivatives and cannot represent the behavior of the fluid close to the critical point. Kunick et al. describe a method using quadratic splines to represent the fluid properties of water and steam for the International Association for the Properties of Water and Steam (IAPWS) (Kunick, 2018). This method divides a domain of interest into three distinct regions of fluid state. But by splitting up the domain into non-overlapping sets, the method introduces inconsistencies at the saturation curve between these sets, resulting in errors in the property derivatives near the saturation curve.

US Patent Application 2020/0050158 (Xu, 2020) describes a thermodynamic property calculation method using a linear approximation of the properties, but this does not capture the nonlinearities that are pominant near the saturation curve. US Patent 7,676,352 B1 (Van Peursem and Xu, 2010) describes a method for calculating thermodynamic properties and their derivatives using local approximations of fluid property functions, but it is an iterative algorithm and fails to describe nonlinear fluid behavior on a large domain of interest, and does not provide accurate derivatives near the saturation curve.

Generally, previously published methods that use function approximations such as splines, or commonly used iterative methods based on the Helmholtz free energy function, for example, suffer from two fundamental problems. First, the coordinates used for numerical calculation are not aligned with the saturation curve. In other words, the saturation curve is not represented as a contour of one of the two independent coordinates. Therefore, the discontinuity in derivative across the saturation curve is not accurately represented. Secondly, the coordinates typically used can result in an ill-posed numerical calculation at or near the critical point. This is because one of the coordinates achieves a maximum when expressed as an explicit function of the other coordinate at this point. Iterative methods especially fail near the critical point, and may employ special curve-fit approximations near it. As such, many available fluid property libraries are limited to the sub-critical region. However, both of these problems are purely a consequence of poorly chosen coordinates: The saturation curve itself is smooth everywhere, and the property function itself is smooth everywhere except across the saturation curve.

In this paper, we introduce two coordinate systems for representation of fluid property functions that are well-defined for all regions of practical interest, including the two-phase, vapor, liquid, and super-critical regions. Both coordinate systems are defined to be *aligned* with the saturation curve, so that the discontinuity in derivative is represented in terms of only one of the coordinates, which is defined to be constant along the saturation curve (Laughman et al., 2023). In both coordinate systems, the critical point is no different from any other on the saturation curve, so that super-critical problems are no different than purely subcritical ones.

First, we show how a normalized polar coordinate system may be used to define fluid property functions. In these coordinates, the interior of the unit disk represents the two-phase region, and the liquid, supercritical and vapor regions are represented in the exterior of the unit disk. The saturation curve is an arc of the unit circle. Fluid property functions are represented as B-spline functions (de Boor, 1978; Piegl and Tiller, 1995), arranged such that the transition across the saturation curve is C_0 but not C_1. B-spline coefficients are computed off-line by solving a constrained least squares problem using data generated by a reference calculator such as REFPROP. The implementation is computationally and memory efficient, accurate, numerically well-conditioned and allows for evaluation of derivatives of the fluid property function of any desired order.

We derive a second implementation using normalized parabolic coordinates, which may be less familiar to the reader but for this application have a certain elegance. In normalized parabolic coordinates, the saturation curve is represented as a unit parabola in one of the two coordinates, which is naturally similar in shape to the saturation curve expressed in conventional (h, P) variables. The resulting fluid property functions are computationally efficient and well-conditioned, but some of the peculiarities of parabolic coordinates require additional attention.

Both are realized as a set of C language functions, with interface to Modelica though the external function interface. This makes the coordinates entirely invisible to the user. However, the result begs a question: Is there a benefit to expressing the fluid dynamics equations explicitly in these variables, instead of using conventional physical variables? This might be possible if the coordinate transformations were defined natively in the Modelica language. Addressing this question is left to future research.

This paper is organized as follows. In Sections 2 and 3 we derive the polar and parabolic coordinate transformations and property functions realizations, respectively. We discuss some of the implications in Section 4, and draw some conclusions in Section 5.

2 Polar Coordinates

Consider density ρ (kg/m^3) as a representative fluid property, to be computed as a function of independent fluid property variables pressure P_e (Pa) and specific enthalpy h_e (J/kg), where the subscript e denotes that the variables are in engineering units. Consider a domain of interest Ω in the $h_e - P_e$ plane, on which an approximation $\hat{\rho}$ of ρ is defined. Ω may include the two-phase region, the liquid and vapor regions, and the super-critical region, and

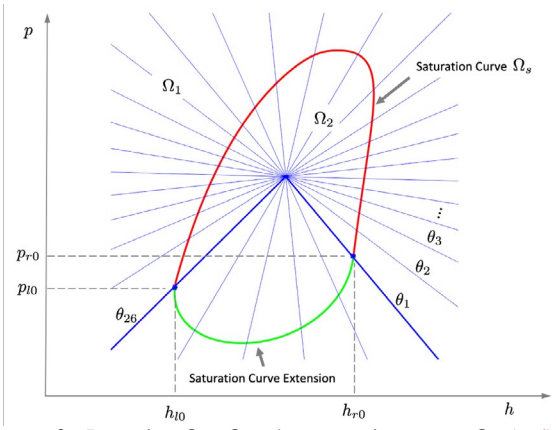


Figure 2. Domains Ω_1, Ω_2, the saturation curve Ω_s (red) and the saturation curve extension (green).

will be defined below. $\hat{\rho}$ is computed in a normalized polar coordinate system defined by the composition of three coordinate transformations $T = T_3 \circ T_2 \circ T_1$.

Scaling Coordinate Transformation T_1

Choose an origin $(h_{e0}, P_{e0}) \in \Omega$, inside the two-phase region, and values for two scaling factors, p_s (dimensionless) and h_s (J/kg), to define the scaling coordinate transformation $T_1 : \mathbb{R}^2 \to \mathbb{R}^2 : (h_e, P_e) \mapsto (h, p)$ as

$$h = (h_e - h_{e0})/h_s \tag{1a}$$
$$p = p_s \cdot \log(P_e/P_{e0}). \tag{1b}$$

The scaling factors are chosen such that the dimensionless p and h are $O(1)$ over Ω. The inverse scaling coordinate transformation, $T_1^{-1} : \mathbb{R}^2 \to \mathbb{R}^2 : (h, p) \mapsto (h_e, P_e)$, is

$$h_e = h_s \cdot h + h_{e0} \tag{2a}$$
$$P_e = P_{e0} \cdot \exp(p/p_s). \tag{2b}$$

Polar Coordinate Transformation T_2

In the scaled (h, p) coordinates, define the polar coordinate transformation $T_2 : \mathbb{R}^2 \to \mathbb{R}^2 : (p, h) \mapsto (r, \theta)$ as

$$r = \sqrt{h^2 + p^2} \tag{3a}$$
$$\theta = \text{atan}(p, h), \tag{3b}$$

where $\text{atan}(\cdot, \cdot)$ is the two-argument, four quadrant inverse tangent function. The inverse polar coordinate transformation $T_2^{-1} : \mathbb{R}^2 \to \mathbb{R}^2 : (r, \theta) \mapsto (h, p)$ is

$$h = r \cdot \cos(\theta) \tag{4a}$$
$$p = r \cdot \sin(\theta). \tag{4b}$$

Saturation Curve Radial Distance Normalization T_3

Figure 2 shows a domain Ω on the (h, p) - plane, divided into three regions: Ω_2 is the two-phase region; Ω_1 is outside the two phase region, and may include the liquid, vapor and super-critical regions; and Ω_s is the saturation curve, which is the boundary between Ω_1 and Ω_2. Define p_{r0} to be a small value of p on the vapor side of the saturation curve Ω_s at or near the lower boundary of Ω. Consider a small value p_{l0} of p on the liquid side of the saturation curve, at or near the lower boundary of Ω. A precise value for p_{l0} will be computed from p_{r0} and the choice of spline knots in the θ-direction below. Define h_{r0} and h_{l0} to be the scaled enthalpies corresponding to p_{r0} and p_{l0}, respectively, on Ω_s. This defines the points (h_{r0}, p_{r0}) and (h_{l0}, p_{l0}) on Ω_s. Express these points in polar coordinates as

$$(r_1, \theta_1) = T_2(h_{r0}, p_{r0}) \tag{5}$$

and

$$(r_{j^*}, \theta_{j^*}) = T_2(h_{l0}, p_{l0}), \tag{6}$$

where j^* is defined below. Then the saturation curve between (h_{r0}, p_{r0}) and (h_{l0}, p_{l0}), including the critical point (h_c, p_c), may be represented on the (h, p) plane in polar coordinates as the image of $(h_{\text{sat}}, p_{\text{sat}}) = T_2^{-1}(r_{\text{sat}}, \theta)$, where $f_{\text{sat}} : \mathbb{R} \to \mathbb{R} : \theta \mapsto r$, is

$$r_{\text{sat}} = f_{\text{sat}}(\theta) \quad \text{for} \quad \theta \in [\theta_1, \theta_{j^*}]. \tag{7}$$

As shown in Figure 2, choose an extension of f_{sat} on the open interval $(\theta_{j^*}, \theta_1 + 2\pi)$ so that the extended f_{sat} is periodic in θ and C^{n-1} (continuous up to $(n-1)^{th}$ derivative) for all $\theta \in \mathbb{R}$, for some value of $n > 0$. (A value for n is defined below as the degree of a spline.) Essentially, this defines a closed curve (a loop) to be the image of the extended f_{sat} that is the saturation curve for scaled pressures larger than p_{r0} and p_{l0}, and connects (h_{l0}, p_{l0}) and (h_{r0}, p_{r0}) through the two-phase region.

The extended function $f_{\text{sat}}(\theta)$ is approximated with a periodic B-spline denoted $\hat{f}_{\text{sat}}(\theta)$, which is fit to data on Ω_s that is generated by a thermofluid property calculator such as REFPROP. Other functional representations, such as NURBS, Fourier series or Chebychev polynomials might also be used. Define

$$\Theta_s = \{\theta_1, \ldots, \theta_{j^*}, \ldots, \theta_N\} \tag{8}$$

to be a set of (periodic) knots in the θ-direction, and denote the corresponding i^{th} degree-n periodic B-spline basis function as $B_{i,n}(\theta)$, $1 \le i \le N$ (de Boor, 1978; Piegl and Tiller, 1995). Then

$$\hat{f}_{\text{sat}}(\theta) = \sum_{i=1}^{N} c_{si} B_{i,n}(\theta). \tag{9}$$

The coefficients c_{si} $1 \le i \le N$ are computed by solving a least squares curve fit to data, as follows. First compute a number N_{ds} of pairs of values of (h, p) along the liquid and vapor sides of the saturation curve from (h_{r0}, p_{r0}) and (h_{l0}, p_{l0}), respectively, up to but not including the critical point (h_c, p_c), using a thermofluid property calculator and the transformations T_1. For many fluids the values of P_e

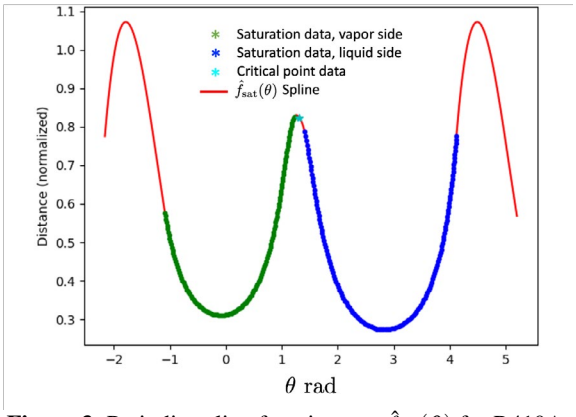


Figure 3. Periodic spline function $r = \hat{f}_{sat}(\theta)$ for R410A.

and h_e on the saturation curve near the critical point is difficult to compute and may be inaccurate, but the value of P_e and h_e at the critical point may be computed accurately. Add the calculated value of (h_c, p_c) to the set of data from the vapor and liquid saturation curves, giving $N_{ds}+1$ data pairs. This set is transformed into polar coordinates using T_2 giving a set of data points (r_k, θ_k) for $1 \le k \le N_{ds}+1$, and this set is used to solve a least squares curve fit problem to compute c_{si}, $1 \le i \le N$.

Note that the set Θ_s need not be uniform, and we may set $j^* = N$, so that the saturation curve extension is defined by a single spline interval. If the data set is accurate, then Θ_s may be defined by the values of θ_k in the data, so that the spline function is interpolating between the data points.

The third coordinate transformation $T_3 : \mathbb{R}^2 \to \mathbb{R}^2 : (r,\theta) \mapsto (\bar{r},\bar{\theta})$, which normalizes the distance between the origin and Ω_s to a constant value of one, is defined as

$$\bar{r} = r / \hat{f}_{\text{sat}}(\theta) \tag{10a}$$
$$\bar{\theta} = \theta, \tag{10b}$$

with inverse $T_3^{-1} : \mathbb{R}^2 \to \mathbb{R}^2 : (\bar{r},\bar{\theta}) \mapsto (r,\theta)$

$$r = \bar{r} \cdot \hat{f}_{\text{sat}}(\bar{\theta}) \tag{11a}$$
$$\theta = \bar{\theta}. \tag{11b}$$

Polar Splines

The fluid property function ρ is approximated by a two-dimensional spline function $\hat{\rho}$ of degree n defined in the $(\bar{r},\bar{\theta})$-coordinates. Spline functions in dimensions higher than one are conventionally constructed for Cartesian coordinates, and the presence of the origin, where conventional polar coordinates exhibit a singularity, requires some care.

Knot Points

Referring to Figure 2, a set of knots Θ_ρ is defined in the $\bar{\theta}$-direction around the full circle, such that the first knot $\bar{\theta}_1$ is coincident with the point (h_{ro}, p_{ro}) on the vapor side of Ω_s, knots are spaced in a counter-clockwise (positive) direction, and the set includes θ_{j^*}. Note that Θ_ρ need not be the same as Θ_s (8), used to represent $\hat{f}_{\text{sat}}$. Computations are simplified if an even number of knots is used such that that both $\bar{\theta}_i$ and $\bar{\theta}_i + \pi$ are in Θ_ρ, simplifying consideration of negative $\bar{r}$. The multiplicity of the knots depends on the region and is discussed in the next section.

In the $\bar{r}$-direction, a set of knots

$$\Phi_\rho = \{-r_n, -r_{n-1}, \ldots, -r_1, 0, r_1, r_2, \ldots, r_{\max}\} \tag{12}$$

is defined such that 0 and 1 are elements, and n negative values are included to create some overlap at the origin. The multiplicity of the knots at $\bar{r} = 1$, corresponding Ω_s, is n so along Ω_s in the $\bar{r}$-direction, the spline function is C_0 but not C_1. All other knots have multiplicity 1 so that the spline function is C^n between any of the knots, C^{n-1} at any of the knots not on Ω_s.

Indexing

Indexing of B-spline functions in polar coordinates is more complex than for Cartesian coordinates. For the $\bar{r}$-direction, denote the set of integers that index the spline basis as

$$\mathscr{I} = \{i \in \mathbb{I} : 1 \le i \le i_{\max}\}, \tag{13}$$

where $i_{\max}$ is the number of spline basis functions. Let $i_{sp} \in \mathscr{I}$ denote the index for $\bar{r} = 1$, and decompose $\mathscr{I}$ into

$$\mathscr{I}_s = \{i_{\text{sp}}\} \tag{14a}$$
$$\mathscr{I}_1 = \{i \in \mathscr{I} : i > i_{\text{sp}}\} \tag{14b}$$
$$\mathscr{I}_2 = \{i \in \mathscr{I} : i < i_{\text{sp}}\}, \tag{14c}$$

so that $\mathscr{I}_s$ contains the basis indices in the $\bar{r}$-direction on Ω_s, $\mathscr{I}_1$ contains the basis indices in the $\bar{r}$-direction outside of Ω_s (region Ω_1), and $\mathscr{I}_2$ contains the basis indices in the $\bar{r}$-direction inside of Ω_s (region Ω_2).

In the $\bar{\theta}$-direction, the number of basis functions depends on the fluid state region, shown in Figure 4, making the B-spline indexing dependent on the region. In the two-phase region Ω_2, the B-spline basis functions in the $\bar{\theta}$ direction are periodic, defined for all values of $\bar{\theta}$, and all of the knots are multiplicity one. Then the set of integers that index the spline basis in the $\bar{\theta}$-direction in region Ω_2 is

$$\mathscr{J} = \{j \in \mathbb{I} : 1 \le i \le j_{\max}\}. \tag{15}$$

where $j_{\max}$ is the number of elements of Θ_ρ.

On the saturation curve, the density $\hat{\rho}$ is smooth as a function of $\bar{\theta}$ except at the points $\bar{\theta}_1$ and $\bar{\theta}_{j^*}$ where there is a transition between the actual saturation curve and the extended saturation curve, $\hat{\rho}$ is C_0 but not C_1 in the $\bar{\theta}$ direction, so the multiplicity of knots at $\hat{\theta}_1$ and $\hat{\theta}_{j^*}$ is n. This leads to a different number of B-spline basis functions in the $\bar{\theta}$ direction for Ω_s compared to Ω_2, requiring a different indexing. The set of integers that index the B-spline basis in the $\bar{\theta}$-direction in region Ω_s is

$$\mathscr{J}_s = \{j \in \mathbb{I} : 1 \le i \le j_{\max} + 2(n-1)\}. \tag{16}$$

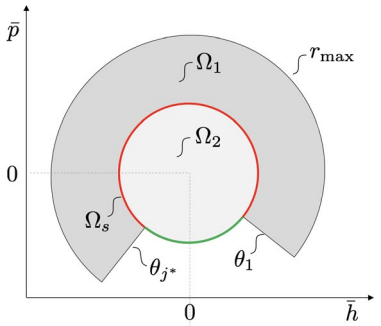


Figure 4. Domain Ω in normalized polar coordinates, showing the saturation curve Ω_s (red) the saturation curve extension (green), the two-phase region Ω_2, and single-phase region Ω_1.

As illustrated in Figure 4, $\hat{\rho}$ in the region Ω_1 is the partial annular set $(1, r_{\max}] \times [\bar{\theta}_1, \bar{\theta}_{j^*}]$. For many thermofluid systems of interest, the fluid property ρ for values of P_e and h_e corresponding to the region below the extended saturation curve, between the limits $\bar{\theta}_{j^*}$ and $\bar{\theta}_1$, is outside the region of interest and is therefore excluded from Ω.

Since the region Ω_1 is only partially annular, the B-spline functions in the $\bar{\theta}$ direction are Cartesian and not periodic in $\bar{\theta}$. The set of integers that index the spline basis functions in the $\bar{\theta}$-direction in region Ω_1 is

$$\mathscr{J}_1 = \{1 \leq j \leq j^* - 1 + n\} \tag{17}$$

Normalized Polar Spline Functions

The normalized polar spline function $\hat{\rho}$ is

$$\begin{aligned}\hat{\rho}(\bar{r},\bar{\theta}) = &\underbrace{\sum_{i\in\mathscr{I}_2}\sum_{j\in\mathscr{J}} c_{ij}B_{i,n}(\bar{r})B_{j,n}(\bar{\theta})}_{\Omega_2} \\ &+ \underbrace{\sum_{i\in\mathscr{I}_s}\sum_{j\in\mathscr{J}_s} c_{ij}B_{i,n}(\bar{r})B_{j,n}(\bar{\theta})}_{\Omega_s} \\ &+ \underbrace{\sum_{i\in\mathscr{I}_1}\sum_{j\in\mathscr{J}_1} c_{ij}B_{i,n}(\bar{r})B_{j,n}(\bar{\theta})}_{\Omega_1}\end{aligned} \tag{18}$$

where $B_{i,n}(\bar{r})$ and $B_{j,n}(\bar{\theta})$ are n-degree B-spline basis functions defined by knot sets Φ_ρ and Θ_ρ, respectively, and c_{ij} are spline coefficients that are computed by solving a constrained least squares or equivalent curve fitting algorithm. Note that although the knot sets are identical for each region, the multiplicities differ, so the $B_{jn}(\cdot)$ are different in each region.

Coefficient Calculation

Values for the coefficients c_{ij} in (18) are computed by solving a constrained least squares problem using a reference property calculator such as REFPROP. First, note that for values of $(\bar{r},\bar{\theta}) \in \Omega_s$,

$$\hat{\rho}(\bar{r},\bar{\theta}) = \sum_{j\in\mathscr{J}_s} c_{i_{\text{sp}}j}B_{i,n}(\bar{\theta}). \tag{19}$$

This is because all of the B-spline basis functions in the $\bar{r}$-direction vanish on Ω_s, except for those corresponding to index i_{sp}, which is identically 1 for $\bar{r} = 1$. This makes the contributions from the Ω_2 and Ω_1 terms in (18) to be zero for $(\bar{r},\bar{\theta}) \in \Omega_s$.

The coefficients c_{ij} for the Ω_s term in equation (18) are computed first using equation (19). For each value of $\bar{\theta}_j$ from a data set $\mathscr{D}_s = \{\bar{\theta}_j : 1 \leq j \leq N_s\}$, where N_s is any integer greater or equal to the number of coefficients in (19) and $\bar{\theta}_j$ suitable sample Ω_s, ρ_j is computed on the extended saturation curve using a thermofluid property calculator such as REFPROP. Then equation (19) may be solved for $c_{i_{\text{sp}}}$ by solving a least squares or similar curve fit problem.

Once the coefficients c_{ij} are computed for the saturation curve Ω_s, then equation (18) decomposes into two decoupled equations

$$\hat{\rho}(\bar{r},\bar{\theta}) - \underbrace{\sum_{i\in\mathscr{I}_s}\sum_{j\in\mathscr{J}_s} c_{ij}b_i^n(\bar{r})b_j^n(\bar{\theta})}_{\Omega_s} = \underbrace{\sum_{i\in\mathscr{I}_2}\sum_{j\in\mathscr{J}} c_{ij}b_i^n(\bar{r})b_j^n(\bar{\theta})}_{\Omega_2} \tag{20}$$

for the two-phase region Ω_2, and

$$\hat{\rho}(\bar{r},\bar{\theta}) - \underbrace{\sum_{i\in\mathscr{I}_s}\sum_{j\in\mathscr{J}_s} c_{ij}b_i^n(\bar{r})b_j^n(\bar{\theta})}_{\Omega_s} = \underbrace{\sum_{i\in\mathscr{I}_1}\sum_{j\in\mathscr{J}_1} c_{ij}b_i^n(\bar{r})b_j^n(\bar{\theta})}_{\Omega_1} \tag{21}$$

for the region Ω_1. Note that the terms on the left-hand sides of (20) and (21) labeled Ω_s may be assigned a numerical value given a value for $(\bar{r},\bar{\theta})$. For each element of a set of data $\mathscr{D}_2 = \{(\bar{r}_i,\bar{\theta}_j) : 1 \leq i \leq N_2,\ 1 \leq j \leq M_2\}$ over the region Ω_2, where $\bar{r}_i$ and $\bar{\theta}_j$ suitably sample Ω_2, and N_2 and M_2 are sufficiently large, values of ρ_{ij} are computed using a thermofluid property calculator such as REFPROP. These values are substituted for $\hat{\rho}$ in equations (20), defining an constrained least squares problem, which is solved for the unknown coefficients c_{ij}. The constraint arises because for coefficients near the origin, c_{ij} for positive $\bar{r}_i$ and $\bar{\theta}_j$ is identical to the coefficient c_{ij} for negative $\bar{r}_i$ and $\bar{\theta}_i + \pi$. This is repeated for a set of data $\mathscr{D}_1 = \{(\bar{r}_i,\bar{\theta}_j) : 1 \leq i \leq N_1\ 1 \leq j \leq M_1\}$ over the region Ω_1, where $\bar{r}_i$ and $\bar{\theta}_j$ suitably sample Ω_1.

Derivative Evaluation

Derivatives of $\hat{\rho}$ with respect to the $(\bar{r},\bar{\theta})$ variables may computed using efficient algorithms (de Boor, 1978; Piegl and Tiller, 1995), and add marginal overhead to the computational cost of evaluation of the B-spline function $\hat{\rho}$ at a given $(\bar{r},\bar{\theta})$. These derivative calculations are exact; there is no numerical differentiation. The derivatives of $\hat{\rho}$ with respect to the two input fluid property variables h_e and P_e are computed with the Jacobian of T, denoted DT,

$$\begin{bmatrix} \frac{d\hat{\rho}}{dh_e} \\ \frac{d\hat{\rho}}{dP_e} \end{bmatrix} = DT \cdot \begin{bmatrix} \frac{d\hat{\rho}}{d\bar{r}} \\ \frac{d\hat{\rho}}{d\bar{\theta}} \end{bmatrix} = DT_3 \cdot DT_2 \cdot DT_1 \cdot \begin{bmatrix} \frac{d\hat{\rho}}{d\bar{r}} \\ \frac{d\hat{\rho}}{d\bar{\theta}} \end{bmatrix}, \tag{22}$$

where DT_1, DT_2 and DT_3 are the Jacobians of T_1, T_2 and T_3, respectively.

At the origin, derivatives of $\hat{\rho}$ with respect to the engineering coordinates h_e and P_e are well defined and are evaluated by computing derivatives of $\hat{\rho}$ with respect to $\bar{r}$ evaluated at $(\bar{r},\bar{\theta}) = (0,0)$ and $(\bar{r},\bar{\theta}) = (0,\pi/2)$, respectively, and then by using elements of DT_1, DT_2 and DT_3 to transform back to engineering units:

$$\frac{\partial\hat{\rho}}{\partial h_e}\Big|_{\substack{P_e=P_{e0}\\ h_e=h_{e0}}} = \frac{\partial\hat{\rho}}{\partial\bar{r}}\Big|_{\substack{\bar{r}=0\\ \bar{\theta}=0}} \cdot \frac{1}{\hat{f}(0)} \cdot \frac{1}{h_s} \tag{23a}$$

$$\frac{\partial\hat{\rho}}{\partial P_e}\Big|_{\substack{P_e=P_{e0}\\ h_e=h_{e0}}} = \frac{\partial\hat{\rho}}{\partial\bar{r}}\Big|_{\substack{\bar{r}=0\\ \bar{\theta}=\pi/2}} \cdot \frac{1}{\hat{f}(\pi/2)} \cdot \frac{p_s}{P_{e0}}. \tag{23b}$$

This calculation is well-defined because the domain of the spline in the $\bar{r}$ direction was extended to include negative values of $\bar{r}$, and also because of the structures of T_1, T_2 and T_3 make some of the off-diagonal terms in the Jacobians zero. Higher derivatives with respect to h_e and P_e are computed similarly.

3 Parabolic Coordinates

Normalized parabolic coordinates are similar to normalized polar coordinates, defined by the composition of three coordinate transformations, $T = T_3 \circ T_2 \circ T_1$, reusing notation from Section 2. T_1 is the same, but here T_2 defines parabolic instead of polar coordinates.

Parabolic Coordinate Transformation T_2

In the scaled (h,p) coordinates, define the parabolic coordinate transformation $T_2 : \mathbb{R}^2 \to \mathbb{R}^2 : (p,h) \mapsto (\sigma,\tau)$ as

$$\sigma = \text{sign}(h) \cdot \sqrt{\sqrt{h^2+p^2} - p} \tag{24a}$$

$$\tau = \sqrt{\sqrt{h^2+p^2} + p}, \tag{24b}$$

with inverse $T_2^{-1} : \mathbb{R}^2 \to \mathbb{R}^2 : (\sigma,\tau) \mapsto (h,p)$

$$h = \sigma \cdot \tau \tag{25a}$$

$$p = (\tau^2 - \sigma^2)/2. \tag{25b}$$

Figure 5 shows constant contours of σ and τ on the (h,p)-plane, along with the saturation curve for R410A for reference.

Saturation Curve τ-Distance Normalization T_3

The third coordinate transformation $T_3 : \mathbb{R}^2 \to \mathbb{R}^2 : (\sigma,\tau) \mapsto (\bar{\sigma},\bar{\tau})$ normalizes the saturation curve to be the locus $\bar{\tau} = 1$, and is defined as

$$\bar{\sigma} = \sigma \tag{26a}$$

$$\bar{\tau} = \tau / \hat{f}_{\text{sat}}(\sigma), \tag{26b}$$

with inverse $T_3^{-1} : \mathbb{R}^2 \to \mathbb{R}^2 : (\bar{\sigma},\bar{\tau}) \mapsto (\sigma,\tau)$

$$\sigma = \bar{\sigma} \tag{27a}$$

$$\tau = \bar{\tau} \cdot \hat{f}_{\text{sat}}(\bar{\sigma}), \tag{27b}$$

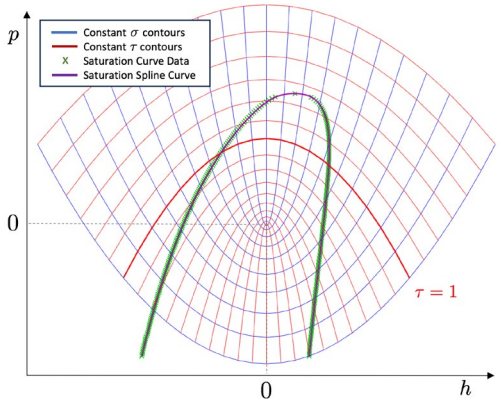


Figure 5. Constant contours of σ (blue) and τ (red), on the (h,p)-plane. Also shown is the saturation curve represented as a spline function (purple), fit to saturation curve data (*).

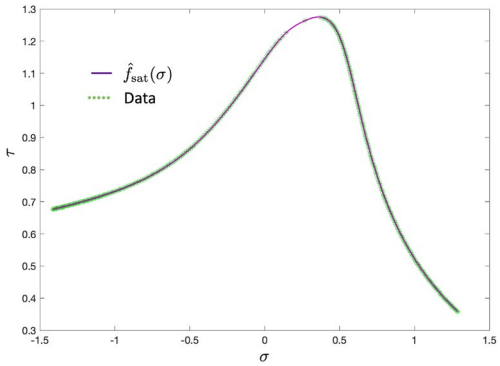


Figure 6. Saturation curve defined as the spline function $\tau = \hat{f}_{\text{sat}}(\sigma)$ for R410A. Note the gap in data around the critical point.

where $\tau = \hat{f}_{\text{sat}}(\sigma)$ denotes an approximation to $\tau = f_{\text{sat}}(\sigma)$, which defines the saturation curve in τ as a function of σ. Just as for polar coordinates, we use a property calculator to compute pairs of values for τ and σ on the saturation curve, using T_1 and T_2, and then fit a spline to the data to give the approximation $\tau = \hat{f}_{\text{sat}}(\sigma)$, as shown in Figure 6 for R410A.

Normalized Parabolic Spline Functions

The density function $\hat{\rho}$ is realized as a 2-dimensional n-degree B-spline function in $(\bar{\sigma},\bar{\tau})$ coordinates,

$$\hat{\rho}(\bar{\sigma},\bar{\tau}) = \sum_{i=1}^{M}\sum_{j=1}^{N} c_{ij} B_{i,n}(\bar{\sigma}) B_{j,n}(\bar{\tau}), \tag{28}$$

defined on the rectangular domain $[-\bar{\sigma}_{\max},\bar{\sigma}_{\max}] \times [0,\bar{\tau}_{\max}]$, which defines $\Omega = \Omega_1 \bigcup \Omega_2 \bigcup \Omega_s$. Figure 7 shows the domain in the $(\bar{\sigma},\bar{\tau})$ coordinates, and its pull back into the (h,p) coordinates, for R410A.

Parabolic coordinates exhibit two characteristics that at first seem to be obstacles but with some thought present no problems. First, there is an apparent singularity along the p-axis ($h = 0$), where $\tau = 0$ along the negative p axis, and where $\sigma = 0$ along the positive p axis. Along a constant σ contour, the sign of σ changes discontinuously from positive in the right half plane to negative in the left half plane

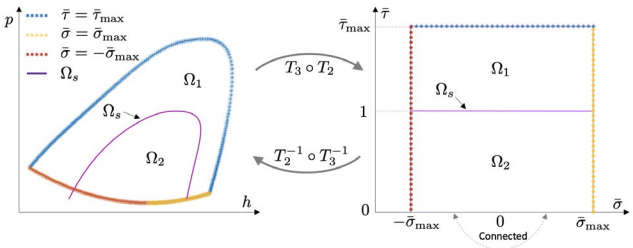


Figure 7. Ω_1, Ω_2, Ω_s and their boundaries in (h, p) coordinates (left) and $(\bar{\sigma}, \bar{\tau})$ coordinates (right) for R410A.

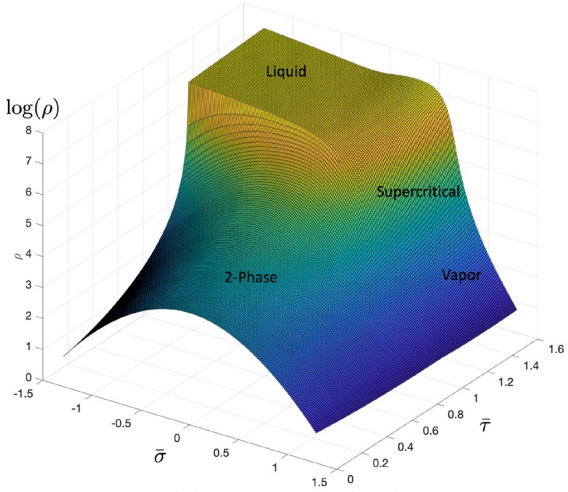


Figure 8. $\log(\hat{\rho})$ for R410A in $(\bar{\sigma}, \bar{\tau})$-coordinates.

at the p axis. This is caused by the sign(h) used to define σ in (24). Second, the boundary of Ω seems to have a different number of edges when represented in $(\bar{\sigma}, \bar{\tau})$ coordinates, compared to (h, p) coordinates. Indeed, the rectangular region in (σ, τ), with four boundary edges, maps to a region in (h, p) coordinates that is bounded by two parabolas, as shown in Figures 5 and 7.

Fortunately these characteristics do not present any obsticles. Figure 7 shows how the rectangular domain Ω in $(\bar{\sigma}, \bar{\tau})$ coordinates maps back to (h, p) coordinates, with the boundaries shown in color. The two vertical boundaries along $-\bar{\sigma}_{\max}$ and $\bar{\sigma}_{\max}$ map to the lower boundary of Ω in (h, p), while the lower boundary $\bar{\tau} = 0$ is in fact not a boundary at all. Points on the positive $\bar{\sigma}$-axis are connected to points on the negative $\bar{\sigma}$-axis, so that $\hat{\rho}(\bar{\sigma}, 0)$ is equivalent to $\hat{\rho}(-\bar{\sigma}, 0)$, for $0 < \bar{\sigma} \leq \sigma_{\max}$. The $\bar{\tau} = 0$ axis is equivalent to the negative p axis, which is inside Ω. Therefore, when defining a spline function $\hat{\rho}$ on Ω, we simply need to ensure that coefficients are constrained so that the spline function is connected across the $\tau = 0$ axis. This ensures that the spline $\hat{\rho}$ and its first $n-1$ derivatives are continuous across $\tau = 0$, and are well defined for all points in $\Omega_1 \bigcup \Omega_2$. This is precisely how the spline coefficients were computed for polar coordinates around the origin (by extending $\bar{r}$ to be negative, and then constraining coefficients for positive and negative $\bar{r}$ to ensure continuity at 0) except for parabolic coordinates, it must be done across the entire $\bar{\tau}$ axis.

Knot indexing is simplified compared to polar coordinates. In the $\bar{\tau}$-direction, knots are spaced from 0 to $\tau_{\max}$, with a knot of multiplicity n placed at 1, which is the saturation curve in $(\bar{\sigma}, \bar{\tau})$ coordinates. All other knots are multiplicity 1. In the $\bar{\sigma}$-direction, knots are spaced from $-\bar{\sigma}_{\max}$ to $\bar{\sigma}_{\max}$, all of multiplicity 1. This defines the degree-n B-spline basis functions $B_{i,n}$ and $B_{j,n}$ in the $\bar{\sigma}$ and $\bar{\tau}$ directions, respectively.

4 Discussion

Both methods are computationally efficient. The calculation of T in polar coordinates requires 11 floating point operations, compared to 14 for the equivalent calculation in parabolic coordinates. Only one floating point division is needed, but if $\hat{f}_{\text{sat}}^{-1}$ is approximated by a spline instead of $\hat{f}_{\text{sat}}$, then that division becomes a multiplication. Additionally, both methods require one 1-d spline function evaluation of $\hat{f}_{\text{sat}}$, plus evaluation of the 2-d spline function $\hat{\rho}$. Spline derivatives are computed essentially for free and pulled back into the (P_e, h_e) coordinates using DT.

Polar coordinates have the advantage of familiarity and simplicity in terms of domain boundaries. Computing derivatives at the origin is not ill-posed because the spline is defined for some range of negative $\bar{r}$, and the derivatives are computed in the (P_e, h_e) coordinates using elements of DT that are all well defined at the origin.

There are some disadvantages to polar coordinates. Indexing is complex. The extended saturation curve is clumsy, and the spline function $\hat{\rho}$ near the point (h_{l0}, p_{l0}) can fit data poorly because of the large change in derivative near this point in the $\bar{\theta}$ direction. Despite their unfamiliarity, parabolic coordinates seem more natural once their peculiarities are mastered, as these issues are avoided entirely. One issue with parabolic coordinates is that the domain Ω is "warped" by T_3. In particular, the vapor region to the right of the saturation curve may be insufficiently covered using a rectangular domain in $(\bar{\sigma}, \bar{\tau})$ coordinates. This is apparent in Figure 7. One solution is to make use of a non-rectangular domain, extending $\bar{\tau}$ for positive values of $\bar{\sigma}$.

It should be emphasized that the spline functions $\hat{f}_{\text{sat}}$ and $\hat{\rho}$ (and its derivatives) should be used consistently and exclusively in any simulation model. These in effect become the definitions of the saturation curve and fluid property function, even though they are spline approximations of a data set, which is in turn was computed from a Helmholtz energy function, which itself is defined to be the reference standard, but is in reality a curve fit to experimental data. It is important not to mix different representations of the saturation curve or fluid property functions in the same model, because even slight differences, especially near the saturation curve, can result in significant differences (or even failures) in simulation results.

For some properties, especially density ρ, we have noticed that a 2-dimensional spline fit to $\log \rho$, (or, more rigorously, $\log(\rho/\rho_0)$ for some constant density ρ_0) instead

of ρ, reduces approximation error, especially near the saturation curve, at the expense of one additional exponential computation during evaluation. A rendering of $\log(\hat{\rho})$ on Ω in $(\bar{\sigma}, \bar{\tau})$-coordinates for R410A is shown in Figure 8. The log reduces the magnitude of the first and second derivatives near the saturation curve, reducing approximation error. We have observed similar behavior for polar coordinates. However we offer no formal proof of this statement.

In practice, significant attention must be paid to data cleaning. Values for ρ, P and h computed by REFPROP, for example, may exhibit small errors that can adversely affect the curve fitting computation. This is especially true for mixtures such as R454C. Errors are caused by finite, nonzero iteration termination conditions in REFPROP's code, or sometimes by failures to converge, and are especially apparent around the saturation curve at high pressures, although other regions can also exhibit errors. A full discussion may be found in (Laughman et al., 2024).

There is little performance gain to be had by implementing these functions directly in Modelica, since the Modelica compiler will translate it into C anyway, and that code is unlikely to be more efficient than the relatively simple, hand coded function. However, one potential performance improvement is largely unexploited: derivative evaluation can be done largely for free when evaluating $\hat{\rho}$. Unfortunately the Modelica compiler does not know that, so it may evaluate the function multiple times to compute $\hat{\rho}$ and its derivatives. Perhaps there is a way to prevent this behavior in Modelica?

Finally, we speculate that the fluid property coefficients, and the 2-dimensional spline function evaluations, could be implemented in integer arithmetic. Although this is not likely to improve numerical efficiency by a large margin in a modern superscalar architecture, it would reduce memory storage requirements. This in turn could reduce simulation time because of improved cache efficiency.

5 Conclusion

This paper presents two methods for computing fluid property functions in simulation models. Both make use of coordinate transformations that align one coordinate with the saturation curve. This provides for precise representation of the fluid property function at the saturation curve, and for connected domains of interest including the liquid, vapor, supercritical and two-phase regions. Both approaches make use of spline function approximation in these special coordinates, and are numerically efficient, well conditioned, and allow for efficient calculation of derivatives up to any desired order that are precise up to processor numerical tolerance.

References

V. Aute and R. Radermacher. Standardized polynomials for fast evaluation of refrigerant thermophysical properties. In *International Refrigeration and Air-Conditioning Conference at Purdue*, 2014.

Adrian Bejan. *Advanced Engineering Thermodynamics*. Wiley, 1993.

Ian H. Bell, Jorrit Wronski, Sylvain Quoilin, and Vincent Lemort. Pure and pseudo-pure fluid thermophysical property evaluation and the open-source thermophysical property library coolprop. *Industrial & Engineering Chemistry Research*, 53(6):2498–2508, 2014. doi:10.1021/ie4033999. URL `http://pubs.acs.org/doi/abs/10.1021/ie4033999`.

Carl de Boor. *A Practical Guide to Splines*. Springer, 1978.

M. Kunick. *Fast Calculation of Thermophysical Properties in Extensive Process Simulations with the Spline-Based Table Look-Up Method (SBTL)*. Number 618. Fortschritt - Berichte VDI, 2018. ISBN 978-3-18-361806-4.

C. Laughman, H. Qiao, and S. A. Bortoff. System and method for calculation of thermofluid properties using saturation curve-aligned coordinates. U.S. Patent 11,739,996, Aug. 29 2023.

Christopher R. Laughman, Vedang Deshpande, Ankush Chakrabarty, and Hongtao Qiao. Enhancing thermodynamic data quality for refrigerant mixtures: Domain-informed anomaly detection and removal. In *Procedings of the* 20th *International Refrigeration and Air Conditioning Conference at Purdue*, 2024.

E. W. Lemmon, I.H. Bell, M. L. Huber, and M. O. McLinden. NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties-REFPROP, Version 10.0, National Institute of Standards and Technology, 2018. URL `https://www.nist.gov/srd/refprop`.

L. Piegl and W. Tiller. *The NURBS Book*. Springer, 2 edition, 1995.

R. Span. *Multiparameter Equations of State*. Springer-Verlag, 2000.

Matthis Thorade and Ali Saadat. HelmholtzMedia - a fluids properties library. In *Proceedings of the 9th International Modelica Conference*, pages 63–70, 2012.

Matthis Thorade and Ali Saadat. Partial derivatives of thermodynamic state properties for dynamic simulation. *Environmental Earth Sciences*, 2013.

D.J. Van Peursem and G. Xu. System and method for efficient computation of simulated thermodynamic property and phase equilibrium characteristics using comprehensive local property models, 2010.

Gang Xu. Super-linear approximation of dynamic property values in a process control environment, 2020.

Objectively Defined Intended Uses, a Prerequisite to Efficient MBSE

Erik Rosenlund[1,2] Robert Hällqvist[1,2] Robert Braun[2] Petter Krus[2]

[1]Saab Aeronautics, Linköping, Sweden
[2]Fluid and Mechatronic Systems, Linköping University, Sweden

Abstract

This article proposes a method for improved model verification within Large-Scale Simulators (LSS). The approach relies on machine-interoperable traceability of model verification information, such as model Operational Domains (ODs). This enables automated evaluation of model relevance and facilitates the combination of models for a broader evaluation of credible simulation results. The paper introduces a proof-of-concept testbed for verification of black-box models against model requirements. Furthermore, the results also include a proposal for a machine-readable format to capture model requirement Verification & Validation (V&V) results, along with the resulting model and updated model OD information.
Keywords: Verification and Validation, Operational Domain, Large-Scale Simulators (LSS), Machine-Interoperable Traceability, Model Reuse, Model Exploration, Simulation Credibility, FMI, SSP, SSP Traceability

1 Introduction

In the area of modeling and simulation, the primary challenges no longer concern whether something can be simulated or not but rather if the results are credible and can be utilized as intended. Many different factors influence whether the results can be used for a specific purpose or not, several of them exemplified by *NASA STD-7009* (2008):

- Are models validated for the scenario simulated?
- Does the model fidelity correspond to what is required by the intended use of the results?
- Are aggregation effects between interacting models sufficiently accounted for?

For a user to make credible decisions based on simulation results, all questions above, and many more, must be answered to the rigor required purpose. When scaling from a scenario where the model designer is making decisions based on simulation results, to a use-case where the simulator end-user has minimal knowledge of the simulated models, additional information needs to be transferred along with the model for the user to be able to draw conclusions regarding the credibility of the test results. Increased simulator complexity or model complexity increases the need for this knowledge transfer. Simulator complexity is in this context seen to be dictated by factors that increase the number of support functions needed for a simulation to execute; such as functions that enable the mixing of Software-in-the-loop (SIL) and Hardware-in-the-loop (HIL) or distributing computations over multiple computers.

How this additional information is to be transferred is situation-dependent, for smaller projects or Modeling & Simulation (M&S) activities it could be enough to discuss model requirements between the user and model designer. For *LSS* (Andersson 2012; Steinkellner 2011), the number of models and support systems makes the amount of knowledge needed to evaluate if results are credible almost impossible for a single user to manually ingest in a reasonable time frame. Thus solutions to automate this workflow are required (Hällqvist 2023). This creates additional demands on the model designer and many model requirements that previously were implicit now need to be explicit and verified in a traceable way. Examples of model characteristics that typically fall into this category are related to runtime performance, numerical errors, or end-use not captured in the original model specification. The gain of explicitly expressing these requirements, and imposing a standard for how the information is relayed, is seen as an enabler in further scaling of LSS.

In short, LSS imposes an increased need for standardized knowledge transfer and evaluation of model intended uses. Enabling this effectively is stipulated to minimize the need for users to possess detailed knowledge of all constituent models and their respective implementations. This would allow extended simulator utilization and increase the credibility of decisions based on simulator results. The goal of this work is to propose a machine-interoperable way of providing model requirement verification information. This goal is expressed in the following research questions,

> RQ: How can information regarding model verification activities be communicated in a tool-independent way to enable model evaluation in a LSS environment?

1.1 Contributions

The work includes the development, evaluation, and demonstration of a proof of concept testbed that provides a structured way of verifying black-box models against requirements. This testbed enables the verification of legacy models where knowledge of the model's intended use and model requirements are documented according to the traditional document-centric paradigm; any improvements on this topic are highly desirable in organizations with a large knowledge capital expressed in legacy models. The testbed verification results are documented in a machine-readable format, based on Extensible Markup Language (XML). The proposed format provides a container for mediating information regarding model verification results that enable automated reasoning on how models can be combined without implicit compromises of credibility. Integrating this information into the model enhances availability and traceability, while also demonstrating an industry-relevant application of the SSP Traceability (Modelica Association 2022) format Simulation Resource Meta Data (SRMD). The proposed structure is thought to initiate a discussion on the establishment of an end-user standardized layer on top of SRMD, capturing what meta-data is relevant to encapsulate together with a model or set of coupled models.

1.2 Research Method

The utilized research method is built on the established method, "Industry-as-laboratory" proposed by Potts (1993), where the industry provides the questions and then acts as a base for conducting experiments. The goal is to enable the study of real-world problems in a scientific manner. The method has since been further refined by Muller (2020) and has previously been applied successfully within the field of aeronautical engineering, see for example Eek, Hällqvist, et al. (2016) and Oprea (2022).

2 Theoretical Background

2.1 Verification and Validation

There have been many attempts in both academia and industry to clarify the difference between model verification and model validation (Wang et al. 2019); however, should we strive to separate the two different sets of activities? Model verification is popularly defined as the quest to answer the question of whether the model is built "right" whereas validation is the quest to answer the question of whether the right model has been built (Osman Balci 1997). In other words, verification concerns a "quality control" on conducted model transformations typically associated with the identification of "bugs" in the implementation. In contrast, validation concerns ensuring that the model is fit for its purpose. So, model validity needs to be assessed with respect to the purpose of the model. Verification does not. However, transforming model requirements into a model is a transformation that needs to be verified, just as the transformation from source code to an executable model that can be integrated in a simulator. In that sense, the modeling begins with the requirements. Any model purpose is therefore ideally expressed explicitly in the form of requirements to, among other things, simplify the model development (Hällqvist 2023). Examples of such functional requirements are shown in Enumeration 1.

Enumeration 1. Functional requirements of a model.

1. The model predicting the ambient air temperature shall predict the temperature with 95% accuracy, concerning the corresponding physical system, in its complete input space, see Section 2.3.
2. The model predicting the ambient air shall relate the model inputs altitude and speed to its response quantity in an *International Standard Atmosphere (ISA)* (*ISO 2533* 1975) atmosphere.

These two requirements can act as a solid foundation to a *model development process* (Carlsson et al. 2012). Transforming model requirements into a mathematical model should be accompanied by verification activities. Such verification activities can be done without any subjective judgment; however, they cannot be concluded until a 95% accuracy in the model response quantity, according to requirement one in Enumeration 1, can be guaranteed with sufficient probability to deem the risk of concluding the activities as acceptable. In early phases, this can be achieved through, for example, Uncertainty Quantification (UQ) (Riedmaier et al. 2020) techniques.

In later phases the accuracy can be accessed through comparisons against in-situ measurements (Beisbart and Saam 2019; Sargent 2010), accompanied by UQ analysis if deemed necessary. Once this point has been reached, there is no need to initiate any activities denoted as "validation"; As the model has been deemed to fulfill its specified purpose (the two requirements). With this line of reasoning, model validation is a subset of verification concerning only the verification of requirements that are implicit or require subjective judgment. These types of requirements are, as pointed out earlier, undesirable. Consequently, we should strive towards declaring an intended use free from implicit requirements and remove the need for model validation.

2.2 Intended Uses

Expressing explicit modeling requirements with verification criteria is a challenging task that requires substantial research. Murray-Smith (2019) emphasizes how important it is for a user to be aware of model limitations and accuracy for all of its intended use and that formal testing is often lacking during model development. Methods and tools to aid engineers in this process are essential. Hällqvist (2023) partitioning model purposes into two different categories: coarse-grained intended-uses and

fine-grained intended-uses:

A coarse-grained intended use qualitatively expresses the purpose of a M&S application in some format, formal or natural language, with a clear connection to, if needed for acceptance, one or more fine-grained intended uses.

A fine-grained intended use quantitatively expresses the purpose of a M&S application in a formal format, with a mathematical connection to one or more validation or predictive capability metrics, predictive capability defining a notion of capturing model representativeness.

To concretize requirements as fine-grained intended uses and to utilize these as foundation during verification is to provide increased traceability during the model design process. Correlations can be made towards Test-driven development (TDD) and maybe even more so towards Acceptance test–driven development (ATDD) where acceptance tests are written before feature development. As described by Martin and Melnik, this allows the designer to analyze the requirements, in a structured way, and to evaluate how they can be translated into tests (Martin and Melnik 2008). Verifying this translation should be done by both the *architect* and *model designer* to ensure that requirements are interpreted correctly and test cases cover all intended uses (Pugh 2010). Expressing a requirement as a test can be one of the most effective ways to verify its 'completeness and accuracy' and the process can be utilized to weed out implicit or ambiguous requirements thus reducing the risk of not 'designing the model right'.

2.3 Operational Domains

The work on concretizing model ODs is viewed as an essential part of expressing fine-grained intended uses. A model OD is viewed as an enclosed n-dimensional volume representing the model's feasible input space. A sought, or required, model OD could be seen as an outcome of the model specification activity. The modeler then has a challenge in realizing or identifying existing models capturing, selected aspects of the physical system to be modeled. Three such ODs are schematically visualized in Figure 1. The OD denoted $OD_{Model\ B}$ schematically represents the feasible input space of an existing legacy model of the System of Interest (SoI). The, by the M&S task, required model OD is visualized as $OD_{Model\ A}$. A first verification activity could encompass the evaluation and comparison of $OD_{Model\ A}$ and $OD_{Model\ B}$; where $OD_{Model\ B}$ can have been deduced analytically or empirically. This verification will concern iterative negotiations, between the model end-users and developer, regarding the overlapping regions of $OD_{Model\ A}$ and $OD_{Model\ B}$ to deduce if the developed model is fit for purpose.

A frequently used representation of the n-dimensional volume representing the OD has been that of an n-dimensional hypercube constrained by the minimum and maximum value of each input variable (FMI development group 2022). A hypercube representation may in certain cases result in a loss of information regarding how a model can be used, see Figure 2 where any attempt to limit the OD to a rectangular domain will reduce the representation of the OD with respect to the actual capability of the model. Both Roy and Oberkampf (2011) and Hällqvist, Eek, et al. (2023) utilize n-dimensional convex hulls to represent ODs; however, they are still models of the domain and suffer from the same problems that any model representation incurs, mainly, 'is it good enough?' Nonetheless, this approach enables transferring a more nuanced picture of the actual domain at the cost of simplicity." The documented use, in the context of the presented research, has focused on creating and utilizing hulls for verification, but not on how it can be stored for further reuse downstream in the M&S chain.

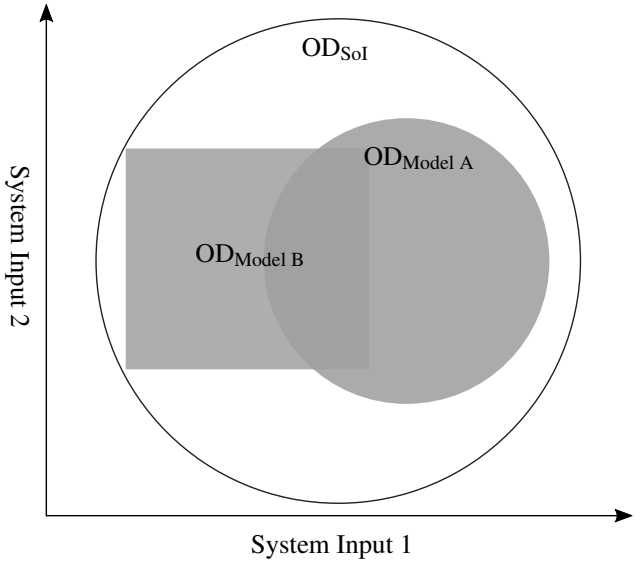


Figure 1. Schematic description of two-dimensional ODs: OD_{SoI} represents the system of interest input space, and $OD_{Model\ i}$ the OD of two different models representing different parts of the system of interest input space.

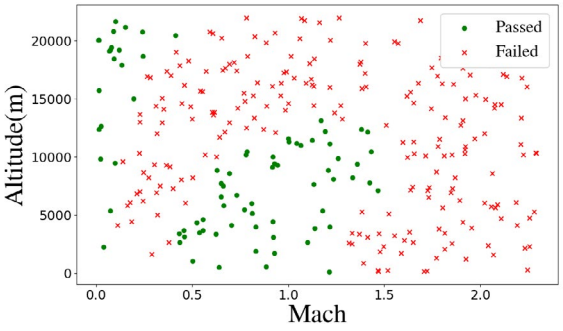


Figure 2. Verification samples mapping input space towards passed or failed evaluations.

2.3.1 Delimitations

Multiple solutions for passing information regarding verification results exist, the most straightforward one is to provide the coordinates of all the tests. This leaves the full responsibility of interpreting the verification results to the end-user. In certain situations, this may be preferred but in LSS this is not an option due to the broad user base and extensive quantities of packaged information. In the end, the choice should be the least complex solution that encapsulates sufficient information to express the model requirement. For example, whilst potentially providing a higher fidelity representation of a model OD, a concave hull is more complex to construct and utilize than a convex hull. Where a set of points can only have one solution in a convex volume, the number of solutions in a concave hull grows rapidly with the number of points (Asaeedi, Didehvar, and Mohades 2014). Other solutions such as clustering algorithms to map the OD into smaller regions of hypercubes, hyperspheres or hulls may provide a more detailed representation, but the choice of clustering algorithm must then be taken into account leading to increased complexity. We will acknowledge the existence of other solutions and limit the scope to the shape of the transferable n-dimensional volume as a convex hull or one of its simpler representations such as the hypercube, since the shape itself is not vital in establishing the methodology around its transfer and it has been utilized in a similar context in related research.

2.4 Model reuse and Traceability

Many of the aspects of model reuse can be correlated to the Findable, Accessible, Interoperable, Reusable (FAIR) principles (Wilkinson et al. 2016) and their role in enabling increased reuse of datasets or, to some extent, Long term archiving and retrieval (LOTAR) (Coïc et al. 2021) in its goal to provide a common standard for geometry data. All aspects of FAIR are needed as enablers for credible model reuse. The Functional Mock-up Interface (FMI) (FMI development group 2019) and System Structure and Parameterization (SSP) standards (Modelica Association Project System Structure and Parameterization 2019) enable some of the FAIR principles by providing common interfaces and data structures, enabling model reuse over a multitude of tools and simulation purposes. However, they do not capture all aspects concerning the *Findable* principle of FAIR. While there are mechanisms to convey information concerning units and permissible input ranges for models or systems, more comprehensive information of intended use is not supported in a structured way as of now (FMI development group 2022).

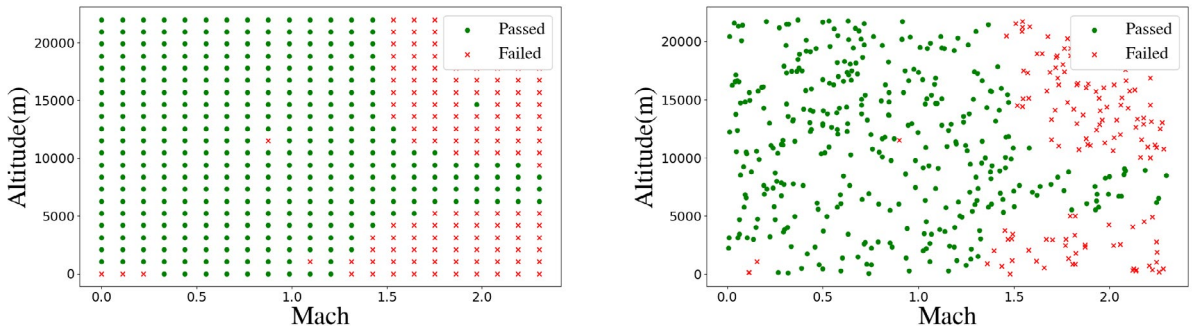
A model expressed in the form $\vec{y}_t = F(\vec{x}_t, \vec{x}_{t-1}, ..., \vec{x}_0)$, often referred to as a *computational model* (P. Fritzson 2004; Ljung and Glad 2004), adds something that static data does not; a data conversion or a *predictive capability* (Beisbart and Saam 2019). This creates an additional requirement that FAIR does not encompass, traceability when it comes to results. Within the healthcare sector, Erdemir et al. (2020) states that having the ability to associate results to input data and model version is "critical for accurate interpretation, repeatability, reproducibility, and debugging of the simulation predictions". There is no reason that this should not be true for the field of complex product development.

The area of storing and reusing engineering knowledge has been under intensive research for a long time (Robinson et al. 2004; Sivard 2001) and according to Pokojski, Knap, and Skotnicki (2021) any knowledge from subject matter experts that can be stored and reused will be beneficiary. Traceability between the executable model and the founding model requirements is one of the cornerstones when it comes to model use; any use of a model in a context where it can not be proved to be credible is by definition not credible (O. Balci and Ormsby 2000). A multitude of standards dictate how requirements are to be traced through a product life-cycle, e.g. *ISO 26262* (2018), *DO-178C* (2012), *NASA STD-7009* (2008) or *MIL-STD-3022* (2008). Since models and the resulting data often are not part of the end-product they can sometimes be exempted from mentioned standards. However, lacking such information traceability can impede the possibility of model reuse. It additionally restricts the utilization of models employed in product verification (Oprea 2022; Hällqvist 2023) or where models are included in the end-product, such as pilot training simulators (*Gripen Mission Trainers* 2024). Transparency and traceability of any data underlying decisions and a formal testing process increase the credibility of models greatly (Murray-Smith 2019).

Increasing utilization of simulation results in the product development process prompts harsher requirements regarding the traceability of models and simulation results (Level 2024). This is the reason for introducing the "Credible Simulation Process Framework" within the Simulation-based Engineering and Testing of Automated Driving (SET Level) project (Level 2024). The process is a base for the SSP Traceability standard (Modelica Association 2022).

2.4.1 Delimitations

To convey model ODs, the SRMD container of the SSP Traceability is selected as an initial bearer. Its intent defined as “SRMD files are used to define essential metadata for resources that can help users quickly understand the content and intent of a simulation” (Modelica Association Project System Structure and Parameterization 2019) goes well in line with the intent to store information regarding intended use. The format sets few boundaries to what can be stored within the container and it provides large freedom in how the resulting OD can be expressed. When looking at information abstraction levels, SSP is more at-



(a) Grid search, evenly distributed points over the input OD. **(b)** *Random search*, uniform distribution over the input OD.

Figure 3. Different methods used for model Operational Domain (OD) exploration.

tuned to system-level information, and FMI is closer to the model level. In some cases, it would be more fitting to utilize a layered FMI standard to convey the OD. This does however pose a limitation in usage, that the information can only be conveyed on a model level and not on a system level. To remedy this, and get access to both model and system standards, a simple solution of incorporating standalone models in a SSP is proposed. A single model can be viewed as a system and can easily be incorporated within a SSP container to get access to both FMI and SSP layered standards. It is possible to do the opposite but not without a loss of flexibility regarding the inner workings of the SSP and then only layered standards of FMI would be available.

2.5 Exploration

The primary method of model exploration involves exposing the model to various scenarios and evaluating the outcomes. Model exploration allows the *model designer* to verify the model against requirements, whether for new model design or model reuse. Many models are developed for, and subjected to, use where the quantities of interests are time-dependent, this is in contrast to scenarios performed under steady-state operating conditions. When synthesizing simulation scenarios the former will hence be referred to as dynamic scenarios and the latter as steady-state scenarios. The underlying purpose of model exploration utilized for verification differs somewhat from Design Space Exploration (DSE) or Hyperparameter Optimization (HPO); while the latter aims to find global extrema in the form of an optimal design, the former seeks to continuously increase the understanding of specific model behavior, often measured by a *coverage* metric (Atamturktur, Hemez, et al. 2009).

To empirically verify the model OD, various established methods in DSE and HPO are considered viable alternatives for model exploration. These methods include ***random search***, where values for each input variable are randomly sampled from a uniform distribution (see Figure 3b). In contrast, ***grid search*** systematically maps the entire input domain with evenly distributed points in a grid (see Figure 3a). Lastly, ***Bayesian search*** iteratively evaluates previous simulations to identify and further explore areas of interest (see Figure 4). This is often done by comparing the resulting *coverage* for new potential points.

According to Feurer and Hutter (2019), *black-box* methods such as *random search* and *grid search* suffer from the *curse of dimensionality* and can take a long time to search a multidimensional volume compared to guided search methods like *Bayesian search*. However, both *random search* and *grid search* are simple and straightforward in their implementation, requiring minimal tuning compared to *Bayesian* algorithms. Comparing *random search* to *grid search*, in x explorations, *random search* will evaluate x different values, whereas *grid search* will only explore $x^{1/n}$ different values since each row corresponds to the same exploration value (Feurer and Hutter 2019). For input combinations with low correlation, this can result in simulations that do not contribute new information. *Random search*, unlike *grid search*, is also an *embarrassingly parallel* algorithm (Herlihy and Shavit 2012), making it highly parallelizable with minimal overhead.

2.5.1 Delimitations

For this study, only steady-state scenarios will be used. These scenarios are considered complex enough to yield useful results while allowing for straightforward evaluation. However, future research may benefit from expressing *coarse-grained* intended use as use-cases, as proposed by Andersson and Carlsson (2012), to deduce dynamic scenarios and create variations used for verification.

Two different exploration methods are selected to verify the model OD: *Random search* and *Grid search*, primarily due to their ease of use. The assessment of coverage based on different metrics, as summarized by Atamturktur, Egeberg, et al. (2015), is also omitted from this study.

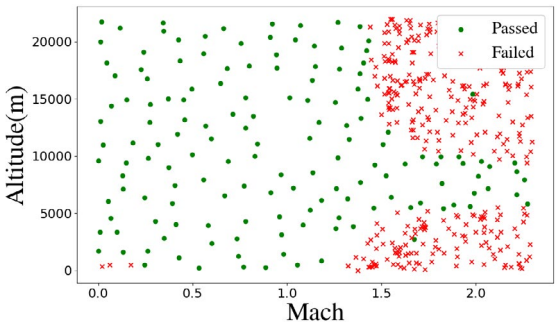


Figure 4. *Bayesian search*, placement of test coordinate depending on previous evaluations.

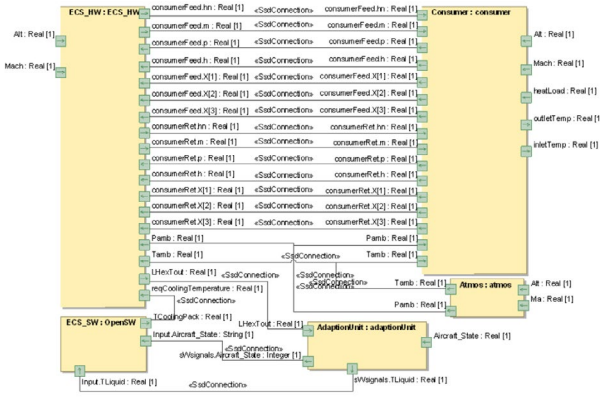


Figure 5. Application example architecture expressed as a SysML block diagram.

3 Application example

The application example simulator used in this study is a virtual reference system, originally developed in the EMBrACE project (ITEA 3 2019). This system is based on publicly available data regarding the modeled constituent sub-systems (Schminder et al. 2018; Hällqvist, Schminder, et al. 2018), and it offers a level of complexity and requirements similar to those encountered during the preliminary and detailed design of aircraft subsystems at Saab Aeronautics. The application example consists of a simple heat sink to consumer loop, including an Environmental Control System (ECS) and its controlling software, as illustrated in Figure 5. This system aligns well with the scope of the study and it provides opportunities to evaluate the use-case specified in Section 4. An initial OD volume is derived from the sub-system requirements. The OD is a subset of the full input space to increase results visibility; here limited to two input variables, altitude and mach. Other inputs are fixed to a specified value in collaboration with the *model designer*.

Two of the models in the application example architecture are chosen for this study: the ECS model and the model representing a generic consumer of cooling power. The inner workings of these two different models have been described in detail by Hällqvist, Munjulury, et al. (VI). In short, however, the ECS model includes both a traditional, bleed-driven, cooling system as well as a coolant distribution system. The coolant distribution system exploits a liquid coolant to transport heat from the consumer to a heat sink utilizing a modeled pump, a heat exchanger, and piping components all modeled using Modelica Standard Library (MSL), initially presented by Pop and P. A. Fritzson (2004), and the Saab in-house Modelica Fluid Light (MFL) (Eek, Gavel, and Ölvander 2017) Modelica library. The consumer model coupled to the ECS model is also modeled using components from the same Modelica libraries. However, the aircraft sub-systems that the models represent are typically developed by different organizations at Saab which motivates the co-simulation approach compared to developing a Modelica monolith.

4 Use-Case

The use-case presented in this section aims to exemplify the need for the research in focus. The use-case aims to capture a realistic and likely scenario in any organization applying M&S for decision-making, in any life-cycle phase. The presented use-case is seen to be applicable also in early life-cycle phases such as *concept development* (Raymer 2018; International Council on Systems Engineering 2015); however, the activities are likely less formal and rigorous to reflect the pace and information uncertainty inherent to the early phases.

4.1 Prerequisites

A need for a new model of a system, sub-system, or component emerges as a result of an engineering task deduced as most efficiently tackled through M&S. Additionally, a set of model requirements have been deduced, see for example Section 2.2, from the requirements on the physical system along with the M&S need. Several similar models exist, as a result of previous M&S activities, but their fit to the current model requirements is uncertain.

4.2 Actors

A total of three actors participating in the use-case are identified: the ***architect***, the ***model designer***, and the ***simulation engineer***. The *architect* supplies requirements on the physical system in focus and utilizes the M&S results to evaluate the system design. The *model designer*, in close collaboration with the *architect*, transforms system requirements to requirements on the corresponding virtual system implementation; furthermore, they design and verify the model against the supplied requirements. The *simulation engineer* utilizes the model to produce the results needed by the *architect*.

It is important to recognize that permutations of the above-stated roles may occur. Naturally, the setup decided to be most efficient in advancing the task at hand is the one that should be employed. The immediate need for trace-

ability of all information concerning the utilization of the model increases when the number of involved actors increases. However, please note that traceability to modeling requirements is here viewed as essential regardless of whether adopting a life-cycle perspective or not.

4.3 Main Scenario

1. The *architect* is to decide on a system design, and appropriate knowledge regarding a constituent subsystem is lacking.

2. The *architect* and the *model designer* identify a potential to fill this gap in knowledge by means of simulating the subsystem.

3. The *model designer* searches central model storage for possible models to reuse.

4. The *model designer* finds a model that could fit the requirements, but the documentation concerning the usage in the new context is incomplete.

5. The *model designer* explores the model to verify it against the new set of requirements. In each exploratory simulation, the model is evaluated using the steps presented in Enumeration 2 until a sufficient *coverage* (Atamturktur, Egeberg, et al. 2015) of the input space is achieved.

6. The *model designer* evaluates the results from 5. If the actor is unsuccessful, he or she adjusts the model and reiterates the main scenario steps from 5.

7. The *simulation engineer* utilizes the model when simulating the subsystem.

8. The *architect* utilizes the results as the basis for selecting, or refining, a system design.

Enumeration 2. Detailed description of the model evaluation steps.

1. Pre-processing, the test input point is selected from the initial OD using an exploration algorithm, and a steady-state scenario is synthesized from the values. (Figure 9)
2. Run simulation, the simulation encompasses three different phases, the duration of the phases is ideally objectively defined with a connection to the model characteristics. (Figure 6)

    A. Ramp-up phase, enables a smooth and numerically sound transition from a cold model state to the test point.

    B. Stabilization phase, allows the model to reach a steady-state at the test input point.

    C. Measurement phase, the model Quantity of Interests (QoIs) are recorded.

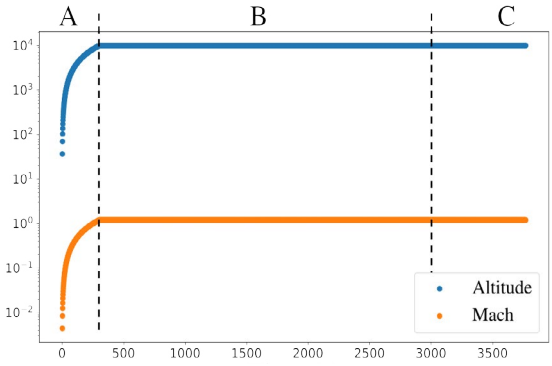


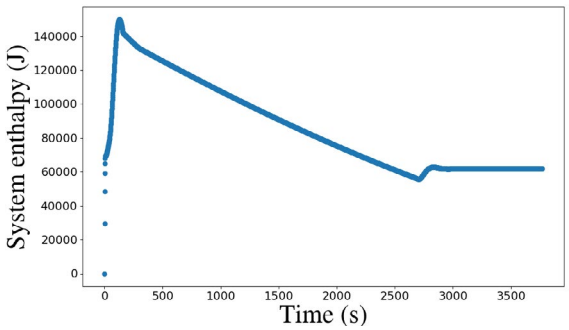
Figure 6. Synthesized simulation scenario of a set of input coordinates, i.e. input steady-state values representing one of the sets of coordinates of Fig. 9. The scenario is partitioned into three different phases: *A-Transitioning from cold to test coordinate*, *B-fixing the input to the selected test coordinate*, and *C-Evaluating test coordinate based on output measurements*.

3. Post-processing, the simulation results (Figure 7a and 7b) are evaluated with respect to the model requirements.

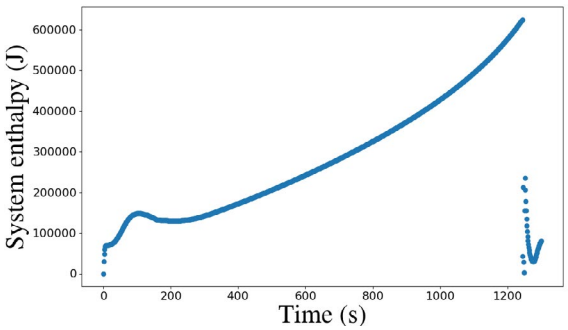
From the results of the model evaluation steps described in Enumeration 2, initial ODs in the form of hulls are generated. These reference hulls enable comparing OD between models and identifying limitations concerning the system's intended use. They also aid in identifying potential attributes for a standardized data format. The determination of sufficient coverage is left to the discretion of the model designer on a case-by-case basis.
The test framework DevelOpment, RIgorous, and automated assessment of models and Simulators (DORIS) (Hällqvist 2023), implemented in the now finalized project *Digital Twin for Automated Flight Test Evaluation and Model Validation* (Hällqvist 2019), is extended and utilized to evaluate models. The software is built around OMSimulator (Ochel et al. 2019) enabling simulations unified by Modelica Association (MA) (The Modelica Association 2019) standards such as FMI and SSP. It enables exploratory testing of models in an automated and repeatable way. The evaluation against requirements will be limited to check model robustness. A set of general model integration requirements is exemplified in Enumeration 3.

Enumeration 3. General model integration requirements adopted to exemplify the presented research.

1. The simulation shall not crash.
2. There shall not be any errors or warnings due to computational/solver problems.
3. Any single time step should be executed within reasonable time limits, as specified by the *model designer*.
4. The modeled system shall be able to reach steady-state, see 7a for example.



(a) Example of simulated QoIs that have succeeded in reaching steady-state. These results correspond to a passed evaluation, marked with a green dot in Figure 9.



(b) Example of simulated QoIs that have failed to reach steady-state. These results correspond to a failed evaluation, marked with a red cross in Figure 9.

Figure 7. Example of simulation results for different test coordinates, see Figure 5 for an overview of the simulated application example.

To evaluate how hulls can be utilized when aggregating multiple models into a larger system, two Functional Mock-up Units (FMUs) are extracted from the application example architecture. These two models are subjected to an identical verification exploration to generate their corresponding hulls. The hypothesis is that the system can only be utilized in the hull created by the intersection of the internal model hulls (x_{model}) for any shared input variable according to

$$x_{system} = \bigcap_{model=0}^{models} x_{model} \tag{1}$$

where x_{system} represent the system OD. Comparison between the model hulls and measurements of selected variables extracted from the system simulation is to provide a basis for conclusions regarding this hypothesis.

Additionally, a reference implementation of a simple *Bayesian search* is developed to evaluate the challenges and possibilities for further research. Although the method will not be used in the current study, some reasoning around the implementation will be included in the Discussion.

5 Results

When the domains initially exhibit a simple hypercube characteristic, it becomes evident from exploring the reference system (see Figure 2) that this representation is insufficient. In investigated model examples, fitting a hypercube to encompass the passed samples will incur a loss of information. It can also be seen in Figure 9 that the quality of the generated hull is also a factor to take into consideration. Where the 'Original hull' has a very low quality in its representation of the domain against the requirement, the 'Low quality hull' increases that quality but there are still a certain amount of verification experiments within the volume that fails. The last area, 'High quality hull', is a very high-quality area where no experiments have failed, but this is at the cost of utilization of the model; in other words, the high quality hull excludes many operating conditions that are identified as 'Passed' .

A simple nearest neighbor metric has been used for creating the different quality hulls. A metric value for each simulation is calculated based on the ratio of how close the simulation is to both failed and successful simulation, mathematically described through

$$sim_{ratio} = f(sim, sims_{fail})/f(sim, sims_{pass}) \tag{2}$$

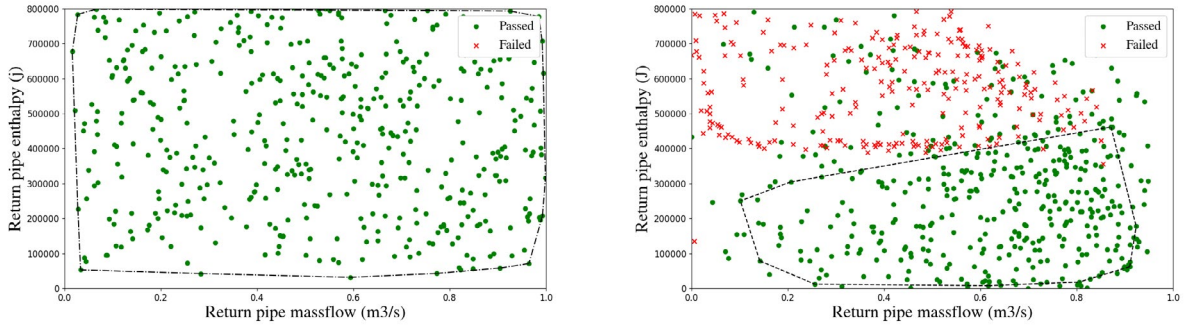
$$f(sim, sims) = \sum_{n=1}^{sims} (1/((distance(sim, sim_n)^2 \tag{3}$$

where sim is the current simulation and $sims_{fail/pass}$ signify all simulations that pass or fail a requirement.

The proposed extension of the SRMD format consists of an *OperationalDomain* tag that encapsulates the OD information. An implementation example is shown in Listing 1, followed by a description of the individual tags and attributes in Table 1, 2, 3, 4 and 5. The included attributes should be viewed as initial examples and should be evaluated further in a broader industrial context to account for requirements from other business domains. A full example can be found in Listing 2 provided in the Appendix.

A note on the types of OD proposed, a hypercube can be expressed as a convex hull but the points needed to do so are 2^n, therefore we also proposed to include "hypercube" as a separate volume type where the geometry is defined by two points "x_{min}, y_{min};x_{max}, y_{max}" for simplicity.

 10.3384/ECP20729 DOI



(a) Results of the consumer model verification and the resulting OD, a total of 300 test are performed.

(b) Results of the *ECS* verification and the resulting OD, a total of 800 tests are performed.

Figure 8. Model ODs that can be used to describe and aggregated system OD.

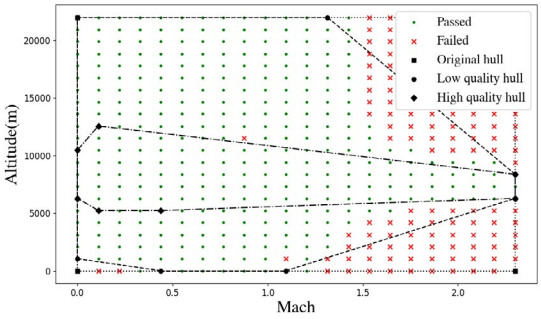


Figure 9. Different quality variants of Operational Domain (OD), represented as convex hulls.

Table 1. OperationalDomain tag details.

Attribute	Description
name	(str) Unique name of the OD
derived	(str) If present, enables traceability to the domain used as a basis for the current exploration
Element	
<Annotations>	If present, FMI or SSP XML standard annotations to support additional information and a human-readable definition of the OD.
<Volume>	Contains the resulting volumetric representation of the OD.
<Requirement>	Defines against what requirements the OD is verified.
<Error>	Provide information regarding the faults within the final hull

Listing 1. SRMD implementation example.

```
<OperationalDomain
    name="domain1.1"
    derived="domain1">
  <Annotations>
    <Annotation type="OD_Information">
      <Info>
        Simulation completed, no errors
      </Info>
    </Annotation>
  </Annotations>

  <Volume
    type="convex_hull"
    points="x1,y1;x2,y2;x3,y3"
    variables="Altitude,Mach"/>
  <Requirement
    simulation_status="no_errors"/>
  <Error fraction="0.1">
      <Point pos="x4,y4">
      ....
  </Error>
</OperationalDomain>
```

Table 2. Volume tag details.

Attribute	Description
type	(enum["convex_hull", "hypercube"]) The volume type OD
points	(array[array[float]]) The coordinates of the points defining the volume.
variables	(array[str]) The variable name mapping towards the model input space.

Table 3. Requirement tag details.

Attribute	Description
simulation_status	(enum["failed", "no_errors", "no_warnings"]) If present, place constraints on simulation execution.
execution_time	(float) If present, place constraints on single-step execution time in seconds.
hull_uncertainty	(float) If present, place constraints on hull uncertainty as the ratio of permissible faults within the volume.

Table 4. Error tag details.

Attribute	Description
fraction	(float) The final ratio between failed/passed points within the hull.
Element	
<Point>	[multiple] If present, coordinates of a failed point within the hull to enable an additional evaluation of credibility.

Table 5. Point tag details.

Attribute	Description
position	(array[float]) Coordinates of a point

5.0.1 OD comparison

When evaluating hulls to establish what performance and usage that can be expected when pairing the application example models, both the consumer and ECS are utilized. A return pipe connecting the consumer and ECS is selected for the comparison and both models were verified using a random search, they both have a nine-dimensional input space where suitable ranges for each input variable were acquired from the respective model designers. The comparison of ODs presented in Figure 8 is conducted over liquid enthalpy and mass flow; these two input variables were selected since they both appear to have some correlation to test failure in both model and system.

Mapping the system usage over the two models, see Figure 11, shows that the hypothesis should not be discarded. The results in this case are promising and the method utilized could give system architects a powerful new tool for evaluating how a system will behave earlier in the product development cycle. However, additional tests involving other models and systems are needed for any strong conclusions to be made but such tests are left for future research.

6 Discussion

O. Balci and Ormsby (2000) argues that a model accreditation recommendation can only be provided for data and scenarios connected to the model's intended use and that any results obtained from conditions outside this scope are not credible. Model exploration is a means to evaluate the model in such a way as to provide evidence supporting that model accreditation can be provided for the full OD. One of the larger questions regarding this is when a requirement can be guaranteed up to a certain confidence level. Taking Figure 9 as an example, the single failed experiment almost in the center of the plot showcases a situation without a simple solution. If the requirement of this particular model dictates a confidence level of 100%, it could be the difference between reusing a model and designing a new one. In some cases, the design cost difference between 100% and 95% can be extensive. Concretizing the confidence level for each requirement enables the creation of a cost-effective solution. As stated Section 2.1, we are to strive towards minimizing implicit requirements. This together with the utilization of a standardized machine-readable format together with the MA standards would fulfill the *Findable*, *Accessible*, *Interoperable*, and *Reusable* principles of FAIR and increase the traceability of model verification.

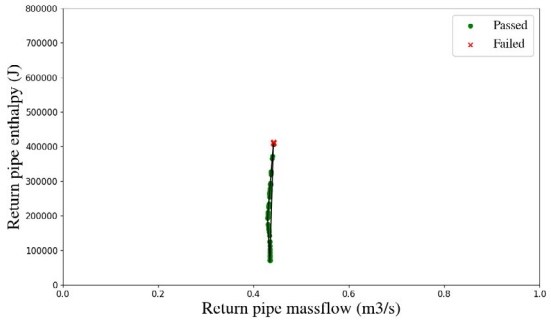


Figure 10. Results of the *system* verification and the resulting OD, a total of 500 tests are performed.

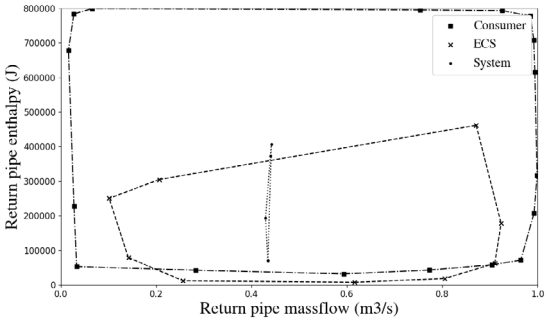


Figure 11. Comparison between the *consumer* model OD, the ECS model OD, and the resulting system utilization OD.

Stating confidence levels in requirements opens up an interesting opportunity for model reuse. As seen in Figure 9, multiple domains for a requirement can be specified depending on the confidence level. The current system may require a 95% confidence level, but during verification, ODs can be created not only for 95% but also for 90% and 80%. This could enable evaluating the model for uses outside the current project that requires a larger OD but has lower requirements regarding confidence level. The same approach may be applied to common requirements such as requirements concerning accuracy and speed. Variations of confidence levels and requirements can, in most cases, be achieved without additional simulations by evaluating already collected data from each simulation against additional requirements. Each model or application area will likely have some specific requirements that may enable easier utilization in new contexts and quick verification in reuse cases. The areas that could benefit the most from this approach will likely become clearer with increased usage of domains in general.

When exploring a model, as established earlier, it is quite ineffective to explore the OD using the described *Grid* or *Random* search approaches. However, as previously stated, a simple *Bayesian search* was implemented to evaluate potential challenges using this method in this context, see Figure 4. It is built upon the assumption more information regarding system behavior can be found in the border regions dividing the passed and failed simulations. It's therefore designed to prioritize new

simulations in such areas. The main finding from this implementation is that a challenge in utilizing a *Bayesian search* as a general solution is in evaluating the choice of tests chosen by the algorithm. For example, evaluating results based on a *Random search* is often straightforward, whereas the corresponding results when using a complex search algorithm may require specialized knowledge of both the search algorithm and the coverage metric. This is especially true as manual evaluation of the exploration strategy becomes more complex with increasing input dimensions. As of now, avoiding specialized search algorithms sidesteps the question of evaluating if the chosen optimization method and *coverage* objective is the correct choice for the current model.

7 Conclusion

Coupling a requirement-driven formulation of a model verification activity in the form of a Operational Domain (OD) would increase traceability and may provide multiple new opportunities for model and system verification. Before simulating, evaluating how aggregated models perform together can provide valuable insights regarding the system's behavior. During simulation, model validity can be monitored and provide warnings for models utilized in an unverified context. After simulating, more computationally intensive verification and debugging can be conducted utilizing logs and results. When combining multiple models, aggregation factors make it difficult to validate the system at scale (Wang et al. 2019), and verification utilized during all phases of usage can be seen as a means to monitor aggregation effects.

Acknowledgements

This research was conducted within the scope of the ITEA4 OpenSCALING project, funded by Vinnova and Saab AB. The authors would also like to thank Dan Louthander for his work with the needed DORIS framework adaptations.

References

Andersson, Henric (2012). "Variability and Customization of Simulator Products : A Product Line Approach in Model Based Systems Engineering". PhD thesis. Linköping University, Machine Design, p. 83. ISBN: 978-91-7519-963-4.

Andersson, Henric and Magnus Carlsson (2012-12-12). *Saab Aeronautics Handbook for Development of Simulation Models. Public Variant*. 1st ed. Linköping University Electronic Press.

Asaeedi, Saeed, Farzad Didehvar, and Ali Mohades (2014). *Alpha-Concave Hull, a Generalization of Convex Hull*. arXiv: 1309.7829 [cs.CG].

Atamturktur, Sezer, Matthew C. Egeberg, et al. (2015-01). "Defining coverage of an operational domain using a modified nearest-neighbor metric". In: *Mechanical Systems and Signal Processing* 50-51, pp. 349–361. DOI: https://doi.org/10.1016/j.ymssp.2014.05.040.

Atamturktur, Sezer, François Hemez, et al. (2009-02-09). "Predictive Maturity of Computer Models Using Functional and Multivariate Output". In: *Proceedings of the 27th Conference and Exposition on Structural Dynamics 2009, IMAC-XXVII*. Orlando, Florida USA.

Balci, O. and W.F. Ormsby (2000-12). "Well-defined intended uses: an explicit requirement for accreditation of modeling and simulation applications". In: *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*. 2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165). Vol. 1, 849–854 vol.1. DOI: 10.1109/WSC.2000.899883. URL: https://ieeexplore.ieee.org/document/899883 (visited on 2024-06-05).

Balci, Osman (1997). "Verification, validation and accreditation of simulation models". In: *Proceedings of the 1997 Winter Simulation Conference*. IEEE Computer Society, Washington, D.C., United States, pp. 135–141. DOI: 10.1109/WSC.1997.640389.

Beisbart, Claus and Nicole J. Saam (2019-04-24). *Computer Simulation Validation: Fundamental Concepts, Methodological Frameworks, and Philosophical Perspectives*. Springer. 1088 pp. ISBN: 978-3-319-70765-5. URL: https://www.ebook.de/de/product/30290750/computer_simulation_validation.html.

Carlsson, Magnus et al. (2012-01). "Methodology for Development and Validation of Multipurpose Simulation Models". In: *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics. DOI: 10.2514/6.2012-877.

Coïc, Clément et al. (2021-09). "Modelica, FMI and SSP for LOTAR of Analytical mBSE models: First Implementation and Feedback". In: *Linköping Electronic Conference Proceedings*. Linköping University Electronic Press. DOI: 10.3384/ecp2118149.

DO-178C (2012-01-05). URL: https://www.do178.org/ (visited on 2024-06-03).

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

Eek, Magnus, Robert Hällqvist, et al. (2016-06). "A Concept for Credibility Assessment of Aircraft System Simulators". In: *AIAA Journal of Aerospace Information Systems* 13.6, pp. 219–233. DOI: https://doi.org/10.2514/1.I010391.

Erdemir, Ahmet et al. (2020-12). "Credible practice of modeling and simulation in healthcare: ten rules from a multidisciplinary perspective". In: *Journal of Translational Medicine* 18.1, p. 369. ISSN: 1479-5876. DOI: 10.1186/s12967-020-02540-4. URL: https://translational-medicine.biomedcentral.com/articles/10.1186/s12967-020-02540-4 (visited on 2024-04-26).

Feurer, Matthias and Frank Hutter (2019). "Hyperparameter Optimization". In: *Automated Machine Learning*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Series Title: The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing, pp. 3–33. ISBN: 978-3-030-05317-8 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5_1. URL: http://link.springer.com/10.1007/978-3-030-05318-5_1 (visited on 2024-06-05).

FMI development group (2019-10-31). *FMI Functional Mock-up Interface*. https://fmi-standard.org/. Accessed: 2019-11-07. URL: https://fmi-standard.org/ (visited on 2019-11-15).

FMI development group (2022-05-10). *Functional Mock-up Interface Specification, FMI for Model Exchange, Co-Simulation, and Scheduled Execution*. Available online: https:

//fmi-standard.org/. Report 3.0. URL: https://fmi-standard.org/.

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

Gripen Mission Trainers (2024). Start. URL: https://www.saab.com/markets/brazil/press-releases/2023/gripen-mission-trainers-in-Brazil (visited on 2024-05-13).

Hällqvist, Robert (2019-05-14). *Digital Twin for Automated Flight Test Evaluation and Model Validation*. Ed. by VINNOVA, National Aeronautical Research Program 7, Ref Nr. 2019-02760.

Hällqvist, Robert (2023). "On The Realization of Credible Simulations: Efficient and Independent Validation Enabled by Automation". PhD thesis. Linköping University, Division of Fluid and Mechatronic Systems. ISBN: 978-91-7929-597-4.

Hällqvist, Robert, Magnus Eek, et al. (2023-01). "Towards Objective Assessment of Model and Simulator Predictive Capability". In: *Journal of Aerospace and Information Systems (JAIS), AIAA.*, pp. 1–16. DOI: https://doi.org/10.2514/1.I011153.

Hällqvist, Robert, Raghu Chaitanya Munjulury, et al. (2022-09-13). "Realizing Interoperability between MBSE Domains in Aircraft System Development". In: *MDPI: Electronics* 11.18. ISSN: 2079-9292. DOI: 10.3390/electronics11182901. URL: https://www.mdpi.com/2079-9292/11/18/2901.

Hällqvist, Robert, Jörg Schminder, et al. (2018-09-09). "A Novel FMI and TLM-based Desktop Simulator for Detailed Studies of Thermal Pilot Comfort". In: *Proceedings of the 31st Congress of the International Council of the Aeronautical Sciences*. International Council of the Aeronautical Sciences, ICAS20180203. ISBN: 978-3-932182-88-4.

Herlihy, Maurice and Nir Shavit (2012-06-25). *The Art of Multiprocessor Programming, Revised Reprint*. Google-Books-ID: vfvPrSz7R7QC. Elsevier. 537 pp. ISBN: 978-0-12-397795-3.

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

ISO 2533 (1975). URL: https://www.iso.org/standard/7472.html (visited on 2024-05-27).

ISO 26262 (2018). *ISO 26262*. URL: https://www.iso.org/standard/68383.html (visited on 2024-06-07).

ITEA 3 (2019-11-12). *EMBrACE*. https://itea3.org/project/embrace.html. URL: https://itea3.org/project/embrace.html (visited on 2024-01-26).

Level, S. E. T. (2024). *SET Level - The safety of automated driving*. URL: https://setlevel.de (visited on 2024-04-26).

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

Martin, Robert and Grigori Melnik (2008-02-01). "Tests and Requirements, Requirements and Tests: A Möbius Strip". In: *Software, IEEE* 25, pp. 54–59. DOI: 10.1109/MS.2008.24.

MIL-STD-3022 (2008-01-28).

Modelica Association (2022-10-27). *SSP Traceability Specification. Version 1.0.0-Beta2*. Standard, unreleased.

Modelica Association Project System Structure and Parameterization (2019). *System Structure and Parameterization*. URL: https://ssp-standard.org (visited on 2019-10-23).

Muller, G. (2020-11-01). *Industry-as-Laboratory Applied in Practice: The Boderc Project, Version: 1.3*. Ed. by University of South-Eastern Norway-NISE. Kongsberg, Norway. URL: http://www.gaudisite.nl/ (visited on 2022-12-15).

Murray-Smith, David (2019-02-08). "Some Issues in the Testing of Computer Simulation Models". In: *International Journal of Business and Technology* 5.1, pp. 1–10. ISSN: 2223-8387. DOI: 10.33107/ijbte.2016.5.1.01. URL: https://knowledgecenter.ubt-uni.net/ijbte/vol5/iss1/1.

NASA STD-7009 (2008). Place: Washington DC.

Ochel, Lennart et al. (2019-03-04). "OMSimulator – Integrated FMI and TLM-based Co-simulation with Composite Model Editing and SSP". In: *Proceeding of the 13th International Modelica Conference*, pp. 69–78. DOI: 10.3384/ecp1915769.

Oprea, Alexandra (2022). *On aircraft simulation in conceptual design*. Linköping: Department of Management and Engineering, Linköping University. 1 p. ISBN: 978-91-7929-476-2.

Pokojski, Jerzy, Lech Knap, and Stanisław Skotnicki (2021). "Concept of a Multi-Criteria and Multi-Disciplinary Design Activity Supporting Tool in the Design and Development Process of CPS". In: pp. 113–122. DOI: 10.3233/ATDE210089. URL: https://ebooks.iospress.nl/doi/10.3233/ATDE210089 (visited on 2024-05-01).

Pop, A. and Peter A. Fritzson (2004). "The Modelica Standard Library as an Ontology for Modeling and Simulation of Physical Systems". In: URL: https://api.semanticscholar.org/CorpusID:26864047.

Potts, C. (1993-09). "Software-engineering research revisited". In: *IEEE Software* 10.5, pp. 19–28. DOI: doi:10.1109/52.232392.

Pugh, Ken (2010-12-22). *Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration*. Upper Saddle River, NJ. ISBN: 978-0-321-71408-4.

Raymer, Daniel P. (2018). *Aircraft design: a conceptual approach*. 6th ed. American Institute of Aeronautics Astronautics. ISBN: 9781624104909.

Riedmaier, Stefan et al. (2020-08). "Unified Framework and Survey for Model Verification, Validation and Uncertainty Quantification". In: *Archives of Computational Methods in Engineering*. DOI: https://doi.org/10.1007/s11831-020-09473-7.

Robinson, Stewart et al. (2004-11). "Simulation model reuse: definitions, benefits and obstacles". In: *Simulation Modelling Practice and Theory* 12.7-8, pp. 479–494. DOI: doi:10.1016/j.simpat.2003.11.006.

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

Sargent, Robert G. (2010). "Verification and validation of simulation models". In: *Proceedings of the 2010 Winter Simulation Conference*, pp. 166–183. DOI: 10.1109/WSC.2010.5679166.

Schminder, Jörg et al. (2018). "Pilot Performance and Heat Stress Assessment Support Using a Cockpit Thermoregulatory Simulation Model". In: *Proceedings of the 31st Congress of the International Council of the Aeronautical Sciences*. International Council of the Aeronautical Sciences, ICAS20180463. ISBN: 978-3-932182-88-4.

Sivard, Gunilla (2001). "A generic information platform for product families". QC 20100812. PhD thesis. KTH, Production Engineering, pp. v, 213.

Steinkellner, Sören (2011). "Aircraft Vehicle Systems Modeling and Simulation under Uncertainty". Licentiate thesis.

Linköping University, Division of Machine Design. ISBN: 9789173931366.
The Modelica Association (2019). *Modelica and the Modelica Association*. https://www.modelica.org/. Accessed: 2018-06-21. URL: https://www.modelica.org/ (visited on 2019-11-15).
Wang, Yanan et al. (2019-03-04). "A survey on VV&A of large-scale simulations". In: DOI: 10.1108/IJCS-01-2019-0004.
Wilkinson, Mark D. et al. (2016-03-15). "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific Data* 3.1, p. 160018. ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18. URL: https://www.nature.com/articles/sdata201618 (visited on 2024-04-26).

8 Appendix

Full example of an SRMD hull implementation, Listing 2, showcasing an initial hull provided by *model designer* and then the resulting verified hull. These two hulls correspond to the "original hull" and "high quality hull" in Figure 9.

Listing 2. Full SRMD implementation example.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<srmd:SimulationResourceMetaData version="1.0.0" name="Simulation meta data"
  generationTool="Manual" generationDateAndTime="2024-02-06T13:21:41Z"
  xmlns:srmd="http://ssp-standard.org/SSPTraceability1/SimulationResourceMetaData"
  xmlns:ssc="http://ssp-standard.org/SSP1/SystemStructureCommon"
  xmlns:stc="http://ssp-standard.org/SSPTraceability1/SSPTraceabilityCommon"
  xmlns:xlink="http://www.w3.org/1999/xlink">

  <OperationalDomain name="initial_domain" >
    <Annotations>
      <Annotation type="OD_Information">
        <Info>
          Initial hypercube provided by model designer Mr/Mrs. X.
        </Info>
      </Annotation>
    </Annotations>

    <Volume
      type="hypercube"
      points="0,0;22000,2.3"
      variables="Altitude,Mach"/>
  </OperationalDomain>

  <OperationalDomain name="no_errors_confidence_1" derived="initial_domain" >
    <Annotations>
      <Annotation type="OD_Information">
        <Info>
          High confidence operational domain of no error requirement,
          confidence level of 100%.
        </Info>
      </Annotation>
    </Annotations>

    <Volume
      type="convex_hull"
      points="10500,0;7000,0;5000,0.1;5000,0.4;6000,2.3;8000,2.3;13000,0.1"
      variables="Altitude,Mach"/>
    <Requirement simulation_status="no_errors"/>
    <Error fraction="0.0">
    </Error>
  </OperationalDomain>
</srmd:SimulationResourceMetaData>
```

 10.3384/ECP20729 DOI

Modelica supported automated design

Ion Matei[1] Maksym Zhenirovskyy[1] John Maxwell[1] Saman Mostafavi[1]

[1]Intelligent Systems Laboratory, SRI, United States, {ion.matei, maksym.zhenirovskyy, john.maxwell, saman.mostafavi}@sri.com

Abstract

We propose a component-based, automated, bottom-up method to system design, using models are expressed in the Modelica language. This bottom-up approach is based on a meta-topology that is iteratively refined via optimization. Each topology link is described by a universal component that is defined in terms of atomic components (e.g., resistors, capacitors for the electrical domain) or more complex canonical components with a well defined function (e.g., operational amplifier-based inverters). The activation of such links is done via discrete switches. To address the combinatorial explosion in the resulting mixed-integer optimization problems, we convert the discrete switches into continuous switches that are physically realizable and formulate a parameter optimization problem that learns the component and switch parameters. We encourage topology sparsity through an L_1 regularization term applied to the continuous switch parameters. We improve the time complexity of the optimization problem by reconstructing intermediate design models when components become redundant and by simplifying topologies through collapsing components and removing disconnected ones. To demonstrate the efficacy of our approach, we apply it to the design of various electrical circuits.

Keywords: component-based, design, optimization, nonlinear programming

1 Introduction

In this paper, we describe a general approach for designing physical systems using a bottom-up approach that implements the "change design" process in Figure 1. This type of problem can be formulated as a mixed integer program that includes a combinatorial part to select the component types and a continuous optimization part that selects parameters of components to meet requirements. A brute force approach to solving such an optimization problem suffers from combinatorial explosion, and heuristics based on branch-and-bound methods do not scale with the number of discrete optimization variables (Clausen 2003; Morrison et al. 2016). To limit the effects of combinatorial explosion, we introduce an algorithm that transforms the mixed-integer formulation into a nonlinear program, with physically realizable solutions.

To facilitate the description of the algorithm and of the results, we focus on design problems in the electrical domain. However, the approach can be generalized to other physical domains. We use the Modelica language to describe the basic components and the generated design solutions, which allows subject matter experts to interpret and evaluate the generated designs.

The design models use a universal component that embeds the behavior of basic components in the electrical domain (e.g., resistor, inductor, capacitor, short connection, and open connection) or more complex components based on operational amplifiers (OpAmps) in various configurations. For example, a universal component based on inverting and non-inverting OpAmp configurations is shown in Figure 2. Each branch of the component is activated or deactivated by a switch that controls the current that flows through it. The design problem is to find the correct switch assignments and component parameter values to meet the requirements, which can be specified in terms of the time evolution of certain quantities of interest or the characteristics of a transfer function in the case of filter design. We start with a topology that describes how the universal components are connected and includes points for setting boundary conditions (e.g., voltage/current sources) and taking measurements. The design problem is then formulated as an optimization problem that minimizes a loss function $\mathscr{C}(\hat{\boldsymbol{y}}_{0:T}(\boldsymbol{p},\boldsymbol{s}),\boldsymbol{y}_{0:T})$, where $\boldsymbol{p}$ and $\boldsymbol{s}$ are the parameters and switches of the basic components, respectively, $\boldsymbol{y}_{0:T}$ is a target vector of measurements over time interval $[0,T]$, $\hat{\boldsymbol{y}}_{0:T}(\boldsymbol{p},\boldsymbol{s})$ is the model's predicted measurements, and $\mathscr{C}$ is a metric that measures the error between the model predictions and the target measurements (e.g., mean square error). The optimization problem also takes into account dynamic constraints, and bounds on component parameters (e.g., resistances must be non-negative).The main contributions of this paper are as follows:

- *Continuous relaxation with lossless realization:* We developed an optimization algorithm that relaxes the integer constraints on the switches by treating them as continuous variables in the range [0, 1]. The parameters of the components and their associated switches are optimized using gradient-free search algorithms and simulations based on Functional Mockup Units (FMUs) (Blochwitz et al. 2011). To encourage sparsity in the design solution, we also add an L_1 regularization term to the loss function. The non-zero switches are not approximated by 0 or 1, but are realized as electric resistors, ensuring no loss in optimality but a possible

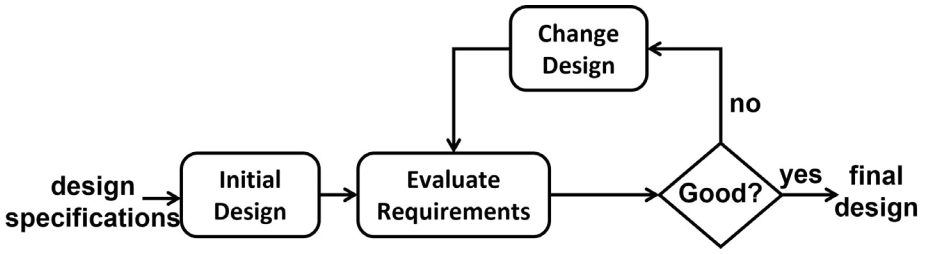


Figure 1. The design optimization process: an initial design is continuously refined until requirements are satisfied.

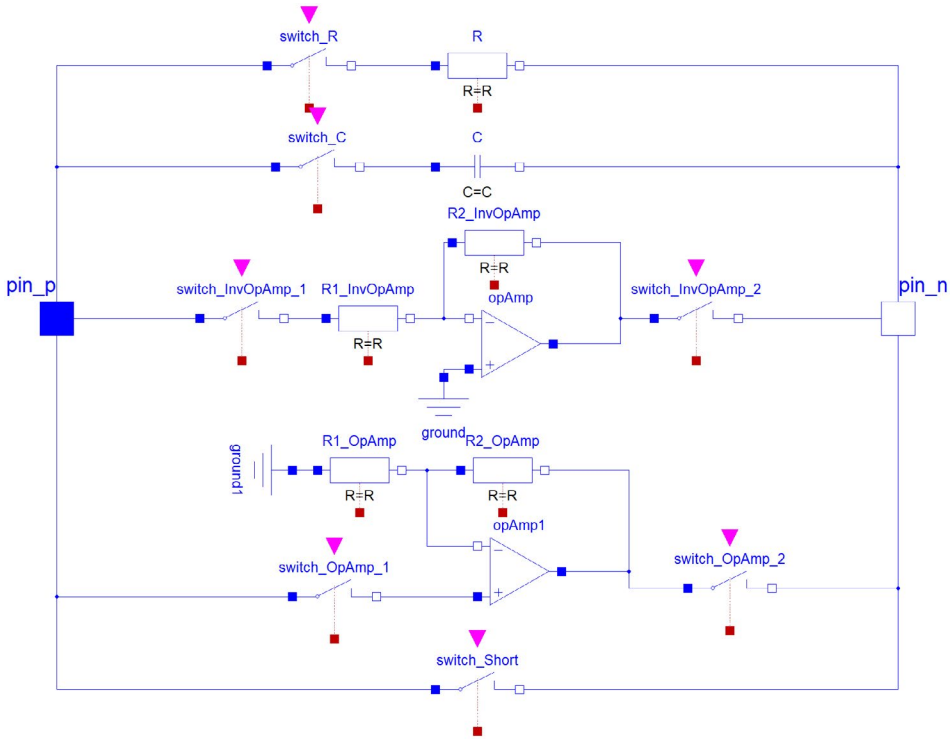


Figure 2. Universal component based on inverting and non-inverting OpAmp configurations.

loss in sparsity. Since we cannot guarantee finding the global optimum, we also use parallel optimization runs with random initial conditions to generate a diverse set of design solutions.

- *Scalability improvement via model simplifications:* During optimization, when certain components are no longer needed (i.e., their switches are set to zero), we eliminate them and reconstruct the design model. This reduces the complexity of the model, as measured by the number of equations, and leads to faster simulation times. In addition, we developed a graph theory-based algorithm that further simplifies the designs generated by the optimization procedure. The algorithm removes unnecessary components, combines compatible components in series and parallel connections into equivalent single components, and annotates the resulting design models for visual representation and simulation in tools that support the Modelica language.

Paper structure: In Section 2, we present an algorithm for automated design that uses continuous relaxation. In Section 3, we discuss how we improve the efficiency of our design algorithm by reducing the complexity of the intermediate design models that are simulated during the design space exploration. Finally, in Section 4, we present the designs generated by the proposed algorithm for various circuit design problems and types of universal components.

2 Design optimization

When using branch-and-bound heuristics to solve mixed integer programs, we may encounter situations where the cost of the relaxed problem is better than the cost obtained by converting the optimization variables into integer values. In this section, we present a method to avoid such a case. The key idea is to interpret the switches in a way that allows for their physical implementation, even when they do not have integer val-

ues. In the universal component definition, each branch has a corresponding switch that opens or closes a connection. When the switch is open, no current flows through the component, leading to its exclusion from the design model. In the Modelica electrical library, the switch (`Modelica.Electrical.Analog.Ideal.IdealOpeningSwitch`) is modeled such that when the switch is open, there is a high resistance that blocks the flow of current. When the switch is closed, there is a very low resistance and the current flows freely. This switch is controlled by a boolean input, the value of which determines the switch mode. We draw inspiration from the definition of the Modelica switch to create a continuous switch that is controlled by a parameter that takes values in the range [0, 1]. The switch is defined by the equations

$$v = a((\varepsilon-1)s+1), \quad (1)$$
$$i = a((1-\varepsilon)s+\varepsilon), \quad (2)$$

where v is the switch voltage, i is the current through the switch, a is an auxiliary variable, $s \in [0,1]$ is the switch control, and ε is a small hyper-parameter that determines the residual resistance when the switch is closed. The switch equation can be simplified to

$$v = \frac{(\varepsilon-1)s+1}{(1-\varepsilon)s+\varepsilon} i,$$

showing that for $s = 0$ we have $v = i/\varepsilon$ and for $s = 1$ we have $v = \varepsilon i$, the expected behavior of a switch. We do not use this simplified representation of the switch for numerical stability reasons. The introduction of the auxiliary variable a prevents the presence of equations with terms that involve divisions by very small numbers. However, the disadvantage is that the resulting system of equations for the design model becomes a differential algebraic equation (DAE) rather than an ordinary differential equation (ODE). This limitation restricts the type of optimization approach that can be used, as we cannot directly utilize platforms that support automatic differentiation (AD) (e.g., the `torchdiffeq` package in Pytorch). In addition to the requirements loss function $\mathscr{C}$, we introduce a sparsity-promoting L_1 regularization term, resulting in the total optimization loss:

$$\mathscr{L}(\boldsymbol{p},\boldsymbol{s}) = \mathscr{C}(\hat{\boldsymbol{y}}_{0:T}(\boldsymbol{p},\boldsymbol{s}),\boldsymbol{y}_{0:T}) + \lambda\|\boldsymbol{s}\|_1,$$

where $0 \le s_i \le 1$, with $\boldsymbol{s} = (s_i)$, and λ is a positive weight that controls the sparsity strength. If in the optimization solution not all entries of $\boldsymbol{s}$ are zero or one, we map them into electric resistors with equivalent resistances, $\frac{(\varepsilon-1)s_i+1}{(1-\varepsilon)s_i+\varepsilon}$. Thus, we can physically realize them, without affecting the optimal cost function, i.e., the design requirements.

The pseudocode for this algorithm is shown in Algorithm 1. We use a gradual approach to achieve sparsity. We start with a small λ value to make sure that we generate an initial design that satisfies the requirements. Then we gradually increase λ until the requirements cost function is no longer improved. Ideally, for each λ, we would like to obtain the optimal solution. The strategy for updating λ is reminiscent to a primal-dual approach (Bertsekas 1999), where we minimize $\mathscr{C}$ under an L_1 sparsity constraint.

In our approach, we incrementally increase the value of λ until it begins to negatively impact the requirements cost function. At this point, we halt the process and perform a final optimization without the L_1 regularization term. The result of this final optimization will be our design solution. Box constraints are commonly used in our problem setup, but we use variable transformations to eliminate them and use an unconstrained optimization algorithm to minimize $\mathscr{L}$. For example, we can eliminate the constraint $a \le x \le b$ by using the transformation $x = a + (\sin(\tilde{x})+1)(b-a)/2$, where $\tilde{x}$ is the new unconstrained optimization variable. It is not guaranteed that the optimization will converge to the global minimum, as the cost function's nonlinear dependence on the optimization parameters means we cannot accurately predict the structure of the problem. Ideally, we would find at least a local minimum for each λ value, but it is possible that the optimization algorithm may take too many iterations to converge. As a result, we set a limit on the number of iterations allowed between λ updates for practical reasons.

All optimization algorithms will require the evaluation of the design model. We use a black-box approach to optimization, where the cost evaluation is done by querying a computational model of the design: an FMU (Blochwitz et al. 2011). In the cosimulation version of the FMU, such a representation contains the algorithm used for simulating the model (e.g., CVODE solver (Hindmarsh et al. 2005)), in addition to the design description. FMUs can be integrated in several languages (e.g., Python, C, Java) and computational platforms (e.g., Matlab/Simulink, OpenModelica, Dymola). The optimization algorithms were implemented in Python based on the `Scipy` optimization package. We used a gradient free (i.e., a direct method) optimization algorithm that relies only on the objective function, namely Powell's method (Powell 1964). Empirically, it provides a better convergence rate than other gradient-free algorithms such as Nelder-Mead, and is faster than global, gradient-free optimization algorithms (e.g., genetic algorithms). The integration of FMUs into the optimization algorithms was done using the `PyFMI` library (Andersson, Åkesson, and Führer 2016).

3 Model Construction and Simplification

We automatically construct a Modelica model for a domain given a universal component and a specification of the initial topology. For instance, if the user wanted to use a 5x6 grid, then the program would generate a Modelica

Algorithm 1 Continuous relaxation design algorithm

Require: δ: solution tolerance
Require: λ: L_1 loss weight
Require: Δ: L_1 loss weight increase rate
Require: FMU of the initial design model
Require: $\boldsymbol{p}, \boldsymbol{s}$: initial parameter and switch values
Require: $\boldsymbol{y}_{0:T}$: target measurements

$\mathscr{C}_{prev} = \infty$
while True **do**
  $\boldsymbol{p}, \boldsymbol{s} \leftarrow \arg\min_{\boldsymbol{p},\boldsymbol{s}} \mathscr{C}(\hat{\boldsymbol{y}}_{0:T}(\boldsymbol{p},\boldsymbol{s}), \boldsymbol{y}_{0:T}) + \lambda \|\boldsymbol{s}\|_1$
  $\mathscr{C}^* = \mathscr{C}(\hat{\boldsymbol{y}}_{0:T}(\boldsymbol{p},\boldsymbol{s}), \boldsymbol{y}_{0:T})$
  if $\mathscr{C}^* \leq \mathscr{C}_{prev}$ **then**
    $\lambda \leftarrow \Delta\lambda$
    $\mathscr{C}_{prev} = \mathscr{C}^*$
    eliminate components corresponding to zero switches and reconstruct the model
  else
    $\boldsymbol{p}, \boldsymbol{s} \leftarrow \arg\min_{\boldsymbol{p},\boldsymbol{s}} \mathscr{C}(\hat{\boldsymbol{y}}_{0:T}(\boldsymbol{p},\boldsymbol{s}), \boldsymbol{y}_{0:T})$
    return $\boldsymbol{p}, \boldsymbol{s}$
  end if
end while

model with 30 grid points with components connecting pairs of points vertically and horizontally (see Figure 3). This model is embedded in another model which specifies the components that set the boundary conditions, i.e., the voltage source and the resistor load (see Figure 4).

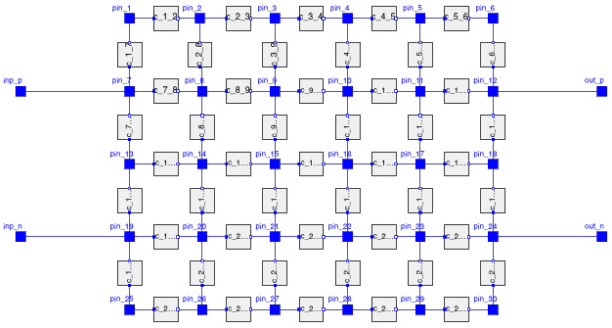


Figure 3. Modelica model for the grid. Universal components connect the grid points.

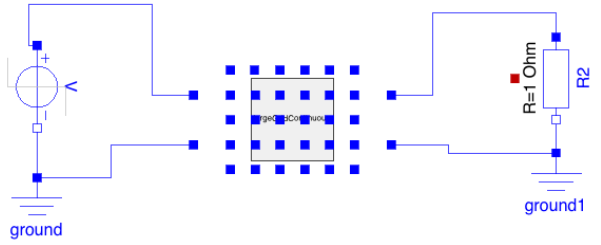


Figure 4. Modelica model for the scenario that gives the boundary conditions of a grid.

In the continuous relaxation approach to optimization, each universal component has switches that allow internal components to be enabled or disabled. These switches can be set from the top level model. When a component is disabled, then the Modelica compiler ignores it when constructing an FMU, thus no equations pertaining to the respective components are added. This process is implemented by conditionally declaring the basic components of the universal component. Consequently, a basic component appears in the instance of a universal component only when a corresponding flag is set to true. The flags of the basic components in all instances of the universal component are continuously updated during the optimization process.

After the optimizer has found a solution (i.e., has determined which components should be enabled and what their parameter values should be), we produce another Modelica model that flattens the universal components and just shows the internal components. At this point we perform two simplification operations: eliminate isolated components and dangling components. These operations are necessary to deal with the cases where switch, resistor or capacitor values are close to zero. Such a situation indicates the presence of open connections. Figure 5 shows a design solution example based a universal component that uses passive components only, and that contains isolated (capacitor between vertices 26 and 27) and dangling (components between vertices 14, 20, 21, 22) components. The design solution can contain isolated components since switches are not exactly zero, meaning that there may be some very small residual currents passing through components. Thus, it may appear that we have components that are isolated but in fact only a small, negligible current passes through them. The isolated components are eliminated by first generating the largest set of connected components that include the boundary conditions (i.e., the voltage source and the resistor load), and discarding the remaining ones. The design solution may also contain components that appear to be dangling, i.e.,

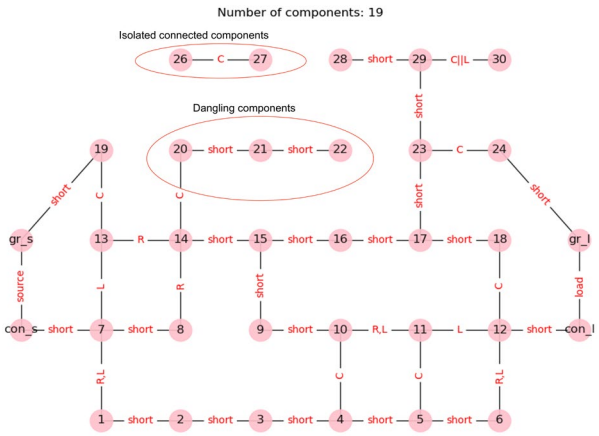


Figure 5. Graph representation of a design solution: vertices are connection points and edges components.

they are connected at one end only. The reason for such a phenomenon is the same as in the isolated components case: residual currents passing through them. The dangling components are found by looking at the cycles of the design. If a component does not belong to a cycle then it must be dangling, thus it is eliminated. We implemented code that generates a visually interpretable layout for the components based on the and-or graph also of the components that are between two grid points. The layout was achieved by annotating the flattened Modelica design model with Modelica notation that generates the visual effects. Finally, we have code to simplify the model by merging compatible serial or parallel components. The code goes through this process iteratively, until no merging can be achieved. The resulting model has correct equivalent parameter values (i.e., resistances in serial connections are added) and it can be simulated using Modelica.

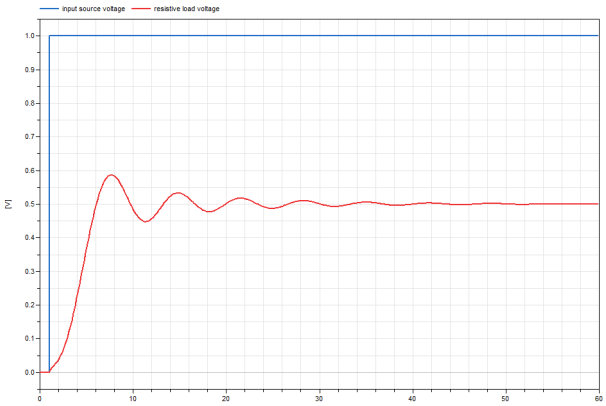


Figure 7. Cauer low pass analog filter: input source voltage vs. resistive load voltage.

4 Results

In this section we present design results based on Algorithm 1 for various design examples.

4.1 Cauer analog low pass filter with passive components

Our goal is to design a filter whose output from a step response matches the output of the Cauer analog low pass filter of the fifth order (see Figure 6). The input voltage versus the load voltage plot is shown in Figure 7. To

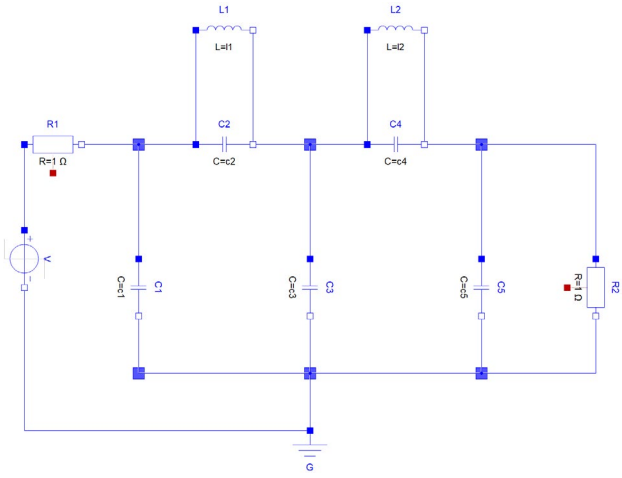


Figure 6. Modelica model of the Cauer analog, low pass filter of the fifth order.

improve the likelihood to find a design solution, we start with a dense initial topology expressed as a 5x6 grid, with a universal component based on passive electrical components. The number of optimization variables corresponding to this initial topology is 343, including component parameters and switch values. The dense initial topology is likely to ensure the existence of several local minima that are close to satisfy the design requirements. To explore multiple of such local minima, we leverage parallel executions of design optimization processes, where each process starts with random initial component parameters, and initial weight for the L_1 cost, and where all switches are initialized to 0.5. We run 20 parallel processes that explore various design solutions. The design optimization algorithm was implemented in Python, and the evaluation of the design loss function was done via FMU-based simulations using the `fmypi` Python package. We refer to each optimization corresponding to an instance of the L_1 loss weight as *outer iteration*. An outer iteration was implemented using the gradient free Powell algorithm, where we limit the execution of the algorithm to 150 (inner) iterations. The limited number of iteration affects only the early outer iterations, since 150 iterations may not be sufficient to converge to a local minima. However, since we use a sequence of outer iterations, where each such outer iteration uses the previous optimization variables as initial values, in practice we do converge to a design that satisfies requirements. More importantly, each outer iteration reduces the time complexity since, after each outer iteration we eliminate redundant components whose switch values are approximately zero. The number of variables drops from 343 at the first iteration to values in the twenties or smaller, at the last iteration. Remarkably, after the first iteration that uses no L_1 regularization term, all processes eliminate more than 250 optimization variables as a result of switches being set to zero. The time per iteration is determined by three factors: the number of iteration of the Powell algorithm, the number of optimization variables and the FMU simulation time. Not unexpectedly, the most expensive outer iteration is the first one, that corresponds to 343 optimization variables. As the design models become simpler, the outer iteration times reduce to tens of seconds. An example of a design solution that realizes the behavior of the Cauer analog filter implemented using passive components is shown in Figure 8.

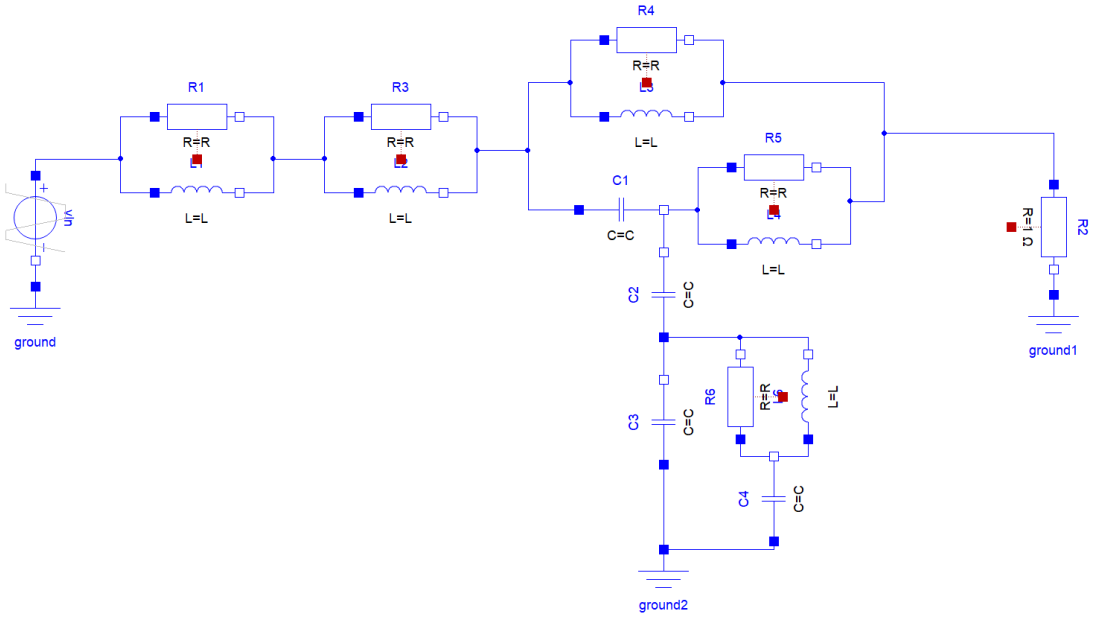


Figure 8. Design solution for the Cauer analog low pass filter based on passive components generated by Algorithm 1.

4.2 Voltage level shifter design with operational amplifiers

We present the results of designing a voltage level shifter (see Figure 9), using Algorithm 1. The universal component employed to generate the initial grid topology consists of a resistor, capacitor, and operational amplifier arranged in a non-inverting configuration, together with open and short connections. We run 10 parallel executions of Algorithm 1 for 150 outer iterations, with a limit of 300 inner iterations for the Powell algorithm in each outer iteration.

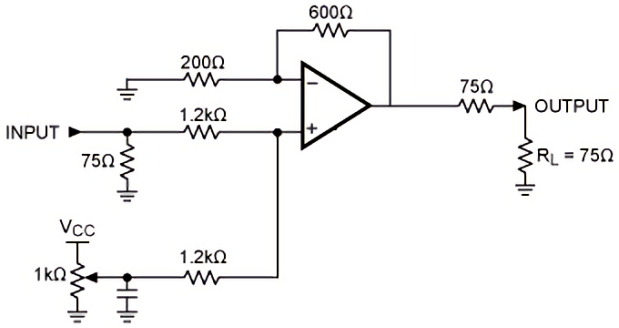


Figure 9. Voltage level shifter circuit used to generate the ground truth data in the form of the voltage across the load resistor (R_L).

Two examples of design solutions produced by Algorithm 1 for the voltage level shifter are depicted in Figures 10 and 11. Notably, both solutions have a component count that is similar to that of the original level shifter depicted in Figure 9, with 10 and 9 components for the two solutions compared to 8 components in the original circuit (not counting the load resistor and the voltage source components). Additionally, both solutions utilize a single OpAmp.

4.3 Cauer analog low pass filter with active filters

We repeated the design optimization problem for the Cauer low pass filter, where the branches of the universal component include first and second-order low and high-pass filters, implemented using operational amplifiers, together with resistor, capacitor, short and open connection components. We started with a 2x6 grid as the initial topology and ran 10 parallel executions of Algorithm 1 for 250 outer iterations, with a limit of 1000 inner iterations for the Powell algorithm in each iteration. After a final simplification, we chose one of the solutions and arrived at a circuit shown in Figure 12 that includes 8 operational amplifiers. The Modelica Standard Library (MSL) has an implementation of the Cauer analog filter that uses only 5 operational amplifiers but also includes 4 negative resistors, where each negative resistor can be implemented using an operational amplifier. Our design solution therefore has a similar number of operational amplifiers as the one in the MSL.

Table 1 summarizes the design results of the above examples in comparison with the original circuits that were used to generate the ground truth. When counting the number of resistors and OpAmps in the MSL active implementation of the Cauer filter, we included the number of resistors and OpAmps needed to implement the negative resistors. The loss function used in the optimization algorithm focuses on behavior and complexity (via the L1 regularization term). The loss function can be augmented with additional objectives that can include component costs, for example. The computational time depends

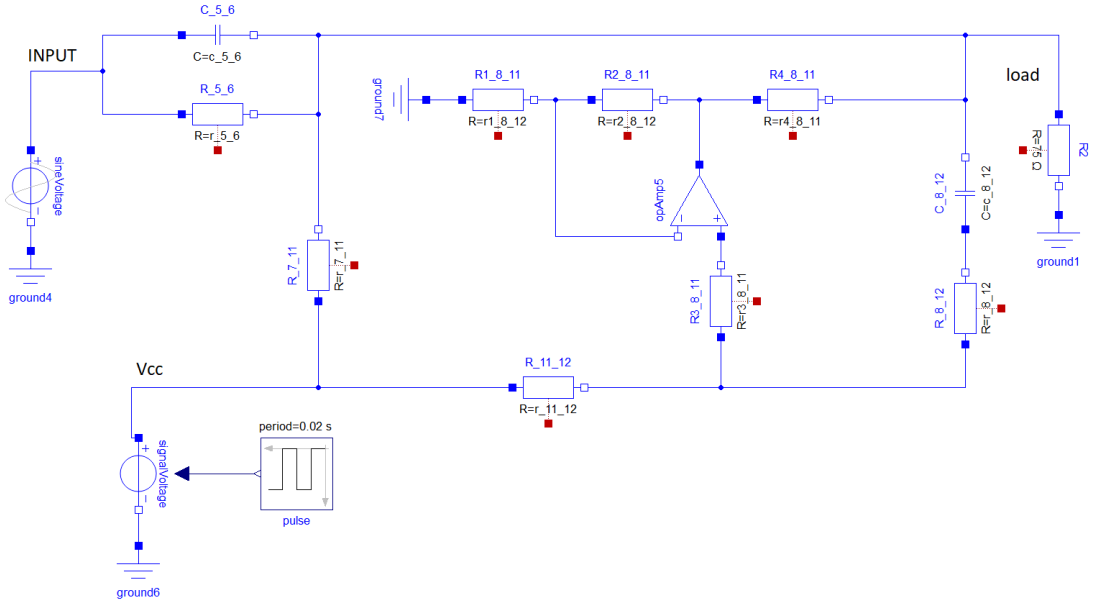


Figure 10. Design of the voltage level shifter with operational amplifiers using Algorithm 1: design solution 1.

on the number of iterations of the optimization algorithms and the FMU simulation time. The latter can be decreased or increased by manipulating the number of collocation points or the solver tolerances. Depending of the complexity of the initial models and the weight of the L1 regularization term, the optimization algorithms can take from tens of minutes to several hours.

5 Differential programming for gradient-based optimization

The algorithm introduced in the previous sections uses gradient-free optimization to search for the component parameters. The advantage of such algorithms is that they work directly with computational representations of the design model (i.e., FMUs). The disadvantage is that they become slower as the number of optimization variables increases. An alternative to gradient-free algorithms is gradient-based algorithms, and the optimization problem would translate into a nonlinear program with dynamical constraints. Solving such a problem would requires having access to the gradients of the objective and constraint functions. When dealing with design models represented as ODEs, we can map the design optimization problem into a framework that supports automatic differentiation (AD) (e.g., Pytorch (Paszke et al. 2017) or Jax (Bradbury et al. 2018)), and solve the problem using gradient descent algorithms. Such platforms are endowed with ODE solvers that support AD (Chen et al. 2018). To formulate the problem in frameworks such as Pytorch or Jax, we first need to extract the equations from the Modelica model of the design. One approach is to generate an XML representation for the DAE using the `dumpXMLDAE` function of the OpenModelica (Fritzson et al. 2010; Open Source Modelica Consortium n.d.) scripting language. Alternatively, we can process the flattened Modelica using a Python Modelica parser such as `modparc` (Dong-Ping 2013). Similar equation extraction can be done using commercial Modelica tools such as `Dymola`, or `SystemModeler`. The extracted equations are converted into symbolic objects such as `Sympy` (Meurer et al. 2017) objects, and mapped into deep-learning platform objects that support automatic differentiation. This process leads to a constrained optimization problem that in the case of the continuous relaxation approach is given by:

$$\min_{\boldsymbol{x},\boldsymbol{p},\boldsymbol{s}} \quad \mathscr{C}(\hat{\boldsymbol{y}}_{0:T}(\boldsymbol{p},\boldsymbol{s}),\boldsymbol{y}_{0:T}) + \lambda\|\boldsymbol{s}\|_1 \tag{3}$$

$$\text{subject to:} \quad \dot{\boldsymbol{x}} = f(\boldsymbol{x},\boldsymbol{z};\boldsymbol{p},\boldsymbol{s}), \tag{4}$$

$$g(\boldsymbol{x},\boldsymbol{z};\boldsymbol{p},\boldsymbol{s}) = 0, \tag{5}$$

$$\hat{\boldsymbol{y}} = h(\boldsymbol{x},\boldsymbol{z};\boldsymbol{p},\boldsymbol{s}), \tag{6}$$

where (4)-(5) is the DAE in the semi-explicit form, representing the dynamics of the design model, and $h(\boldsymbol{x},\boldsymbol{z};\boldsymbol{p},\boldsymbol{s})$ is the sensing model.

To solve (3), we can convert (4) into a set of equality constraints using direct collocation methods (Hargraves and Paris 1987; Herman and Conway 1996), or we can use local (e.g., Chebyshev polynomial expansions (Boyd 2001)) or global (e.g., neural networks) parameterizations of the state solution and solve for the representation parameters (e.g., weights and biases of the neural network). For example, if we use neural networks to represent the state $\boldsymbol{x}(t) = NN_x(t;\beta_x)$ and the algebraic variables $\boldsymbol{z}(t) = NN_z(t;\beta_z)$, the optimization problem (3) will be solved in terms of the parameters β_x, β_z, $\boldsymbol{p}$, $\boldsymbol{s}$. In addition, automatic differentiation can be used to evaluate the time derivative of the state. Our attempts to use a differentiable programming paradigm to solve design problems were met with mixed results. In the case where the model is represented

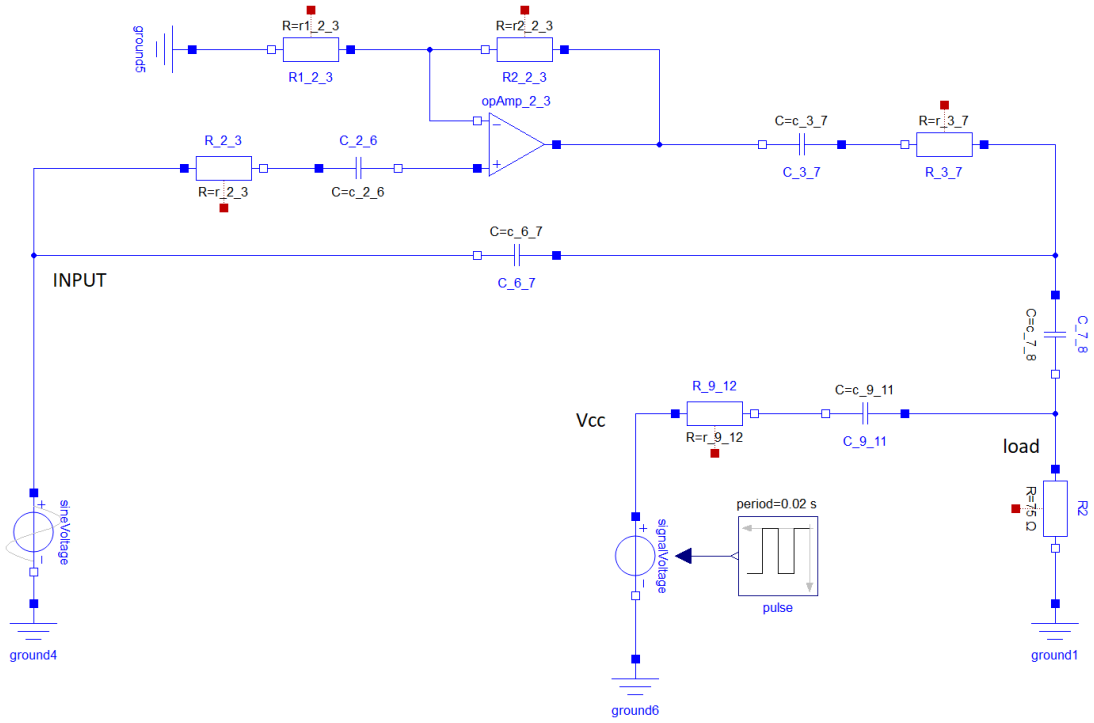


Figure 11. Design of the voltage level shifter with operational amplifiers using Algorithm 1: design solution 2.

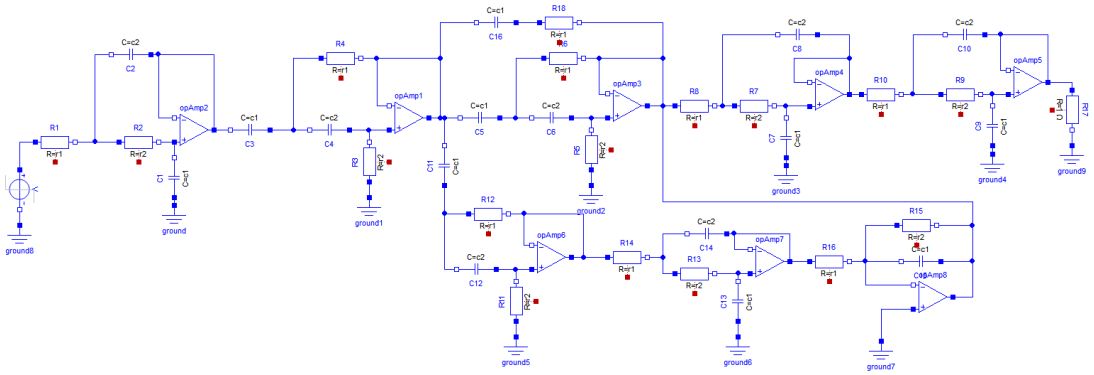


Figure 12. Design solution for the Cauer analog low pass filter with operational amplifiers using Algorithm 1.

as an ODE, we obtained good results. For example, in (Ion Matei et al. 2020) we showed how to learn control policies for an inverted pendulum using a model predictive control approach solved using Pytorch. When dealing with DAEs though, the gradient-based optimization algorithm, when combined with direct collocation methods to approximate time derivatives, tend to converge slowly. In addition, the parameterized DAE solution does not always check against the DAE simulation executed with the optimal component and switch parameters. Unfortunately, we cannot always guarantee that the design model can be represented as an ODE, especially since the model is repeatedly reconstructed and simplified. Thus, the results shown in this paper use a direct method (i.e., Powell algorithm), instead a gradient-based approach. Ideally, we would like to have a sensitivity analysis method embedded in the DAE solvers, so that we can access the Jacobian of the state with respect to the model parameters. Such a method is present for instance in the SUNDIALS software family, introduced in (Gardner et al. 2022; Hindmarsh et al. 2005), with DAE solvers such as CVODES and IDAS that include both direct and adjoint-based approaches to compute sensitivities. Currently though, deep learning platform do not offer such a functionality, except for the case where the DAE can be transformed into an ODE. Moreover, even when dealing with ODE, gradient-descent algorithm that include solvers supporting automatic differentiation tend to slow down as the number of optimization parameters increases. We addressed this challenge in (I. Matei et al. 2023), where we showed that block coordinate descent algorithm in combination with direct collocation method speed up training by several order of magnitude.

Circuit	Number of resistors	Number of capacitors	Number of inductors	Number of OpAmps
Original passive Cauer filter	1	5	2	0
Designed passive Cauer filter	5	3	5	0
Original active Cauer filter	19	8	0	9
Designed active Cauer filter	17	16	0	8
Original voltage level shifter	7	1	0	1
Designed voltage level shifter (sol. 1)	8	2	0	1
Designed voltage level shifter (sol. 2)	5	4	0	1

Table 1. Summary of the design results for various examples.

We are currently working on extending this approach to DAE models. There are Julia libraries that can also be used for a gradient-based approach. For example, ModelingToolkit.jl and its component library, ModelingToolkitStandardLibrary, are modeling languages for symbolic-numeric computation (Ma et al. 2021). They combine symbolic computational algebra systems ideas with causal and acausal equation-based modeling frameworks. We did not use this library in our work because it lacks many components from the Modelica Standard Library, thus requiring a model-transformation component for mapping Modelica models into Julia representations. White the differential programming paradigm is an appealing avenue for dealing with numerical complexity, we cannot always guarantee that the model we use are smooth. It is possible for such models to be hybrid (i.e., include discrete and continues variables) and thus not differentiable.

6 Conclusions

In this paper, we presented an automated design process utilizing a bottom-up approach. The process begins with an initial possibly large topology of universal components that is iteratively refined until a sparse solution is found. The initial design is based on universal components, each of which can exhibit a range of behaviors through basic components. This combination of modes and topology ensures a broad coverage of the design space. We demonstrated an approach for addressing the combinatorial explosion typical of design optimization problems. The approach relaxes discrete variables to continuous variables by transforming discrete switches into continuous switches. These continuous components are physically realizable, resulting in no loss in performance. Additionally, sparsity is induced through an L_1 regularization cost that encourages the parameters of the continuous switches to be zero. The proposed approach is supported by automated model simplification and reconstruction that reduce the complexity of the design model, in turn decreasing the time complexity for the continuous optimization algorithms that require model simulations. The continuous optimization algorithms are gradient-free. We are currently investigating the application of a differential programming paradigm to the design problem described in this paper, which would allow us to utilize gradient-based algorithms. The major challenge we face is extending automatic differentiation support to DAEs that typically require stiff, implicit numerical solvers, while avoiding the need for implementing model-transformation modules to convert Modelica models to new representations.

References

Andersson, C., J. Åkesson, and C. Führer (2016). *PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface*. Technical Report in Mathematical Sciences 2. Centre for Mathematical Sciences, Lund University.

Bertsekas, D.P. (1999). *Nonlinear Programming*. Athena Scientific.

Blochwitz, T. et al. (2011). "The Functional Mockup Interface for Tool independent Exchange of Simulation Models". In: *In Proceedings of the 8th International Modelica Conference*.

Boyd, John P. (2001). *Chebyshev and Fourier Spectral Methods*. Second. Dover Books on Mathematics. Mineola, NY: Dover Publications. ISBN: 0486411834 9780486411835.

Bradbury, James et al. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. URL: http://github.com/google/jax.

Chen, Ricky T. Q. et al. (2018). "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc.

Clausen, Jens (2003). "Branch and Bound Algorithms-Principles and Examples". In.

DongPing, X. (2013). *ModParc*. https://github.com/xiedongping/modparc.

Fritzson, Peter et al. (2010-02). *OpenModelica System Documentation*. URL: https://github.com/OpenModelica/OpenModelica-doc/blob/v1.9.1/OpenModelicaSystem.pdf.

Gardner, David J. et al. (2022). "Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software (TOMS)*.

Hargraves, C. R. and S. W. Paris (1987). "Direct trajectory optimization using nonlinear programming and collocation". In: *Journal of Guidance, Control, and Dynamics* 10.4, pp. 338–342.

Herman, Albert L. and Bruce A. Conway (1996). "Direct optimization using collocation based on high-order Gauss-Lobatto quadrature rules". In: *Journal of Guidance, Control, and Dynamics* 19.3, pp. 592–599.

Hindmarsh, Alan C et al. (2005). "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software (TOMS)* 31.3, pp. 363–396.

Ma, Yingbo et al. (2021). *ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling*. arXiv: 2103.05244 [cs.MS].

Matei, I. et al. (2023). "Sensitivity-Free Gradient Descent Algorithms". In: *Journal of Machine Learning Research* 24.300, pp. 1–26. URL: http://jmlr.org/papers/v24/22-1002.html.

Matei, Ion et al. (2020). "Deep Learning for Control: a non-Reinforcement Learning View". In: *2020 American Control Conference (ACC)*, pp. 2942–2948. DOI: 10 . 23919 / ACC45564.2020.9147287.

Meurer, Aaron et al. (2017-01). "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3, e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103.

Morrison, David R. et al. (2016). "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning". In: *Discrete Optimization* 19, pp. 79–102.

Open Source Modelica Consortium (n.d.). *OpenModelica User's Guide*. URL: https : / / openmodelica . org / doc / OpenModelicaUsersGuide/latest/.

Paszke, Adam et al. (2017). "Automatic differentiation in PyTorch". In.

Powell, M. J. D. (1964-01). "An efficient method for finding the minimum of a function of several variables without calculating derivatives". In: *The Computer Journal* 7.2, pp. 155–162.

Proposal for A Context-oriented Modelica Contributing to Variable Structure Systems

Zizhe Wang[1,2*] Manuel Krombholz[2*] Uwe Aßmann[2] John Tinnerholm[3] Christian Gutsche[1,2]
Volodymyr Prokopets[2] Sebastian Götz[2]

[*]Co-first author
[1]Boysen-TU Dresden-Research Training Group, Dresden, Germany
[2]Software Technology Group, Technische Universität Dresden, Germany
[3]Department of Computer and Information Science, Linköping University, Sweden
zizhe.wang, manuel.krombholz, uwe.assmann, christian.gutsche,
volodymyr.prokopets, sebastian.goetz1@tu-dresden.de
john.tinnerholm@liu.se

Abstract

Context-aware systems are widespread in our daily lives, but modeling languages that address the notion of context are rare. Variable structure systems (VSS) allow for structural and behavioral changes in physical models at runtime (while the simulation is running) based on different situations. It is desirable to explicitly describe under which contextual situation a specific variant of the simulation model should be used and how to implement the switching between these variants at runtime. In this case, contexts could be used to control the variability of context-aware systems. Equation-based modeling languages are suitable for modeling complex multi-domain, multi-physical systems, and among them, Modelica is the state-of-the-art. Unfortunately, the capabilities for VSS in Modelica are strongly limited. As a result, several frameworks have been proposed to address this problem by supporting different VSS types. However, it remains unclear which framework contributes to which VSS type. Furthermore, approaches have been developed to support VSS, but none can explicitly describe contexts and their transitions. In this work, we first introduce VSS and its different types. Then, we provide an overview of which framework targets which VSS type. Finally, we propose a new language extension based on Modelica, ContextModelica, that provides semantics for the direct context definition, enabling the use of context to control and manage variability.

Keywords: modeling and simulation, Modelica, variable structure systems, context, context-oriented programming, ContextModelica

1 Introduction

1.1 Context-aware systems

Context-aware systems are widely present in different aspects of our daily lives. According to Dey, Abowd, et al. 2000, a context is "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object considered relevant to the interaction between a user and an application, including the user and application themselves". Many context-aware systems operate according to system contexts (e.g., "a robot should stop working when a human enters its operation area", "an iPhone will make emergency calls if a car crash has been detected"). Different context-oriented techniques have been developed to enhance context-aware systems, including Context-Oriented Programming (Hirschfeld, Costanza, and Nierstrasz 2008), commonly referred to as COP. Elyasaf, Cardozo, and Sturm 2023 and Elyasaf and Sturm 2023 state "Although COP languages have existed for over 15 years, they are still very limited for developing context-aware systems. Also, modeling languages that address the notion of context are rare." Thus, how the idea of COP could be implemented in equation-based modeling languages, such as Modelica, remains a research question.

1.2 Variable Structure Systems and Modelica

Utkin 1977 introduced variable structure systems (VSS), which consist of continuous subsystems with a proper switching logic and enable dynamic control of simulation systems. In real applications, certain conditions, such as contexts (Elyasaf and Sturm 2022), can be used to control the variability (different modes). Modes refer to different states; different modes correspond to different models defined by distinct equation systems.

Figure 1 shows a minimal example. On a sunny day (*Context* = *Sunny*), solar radiation is present, and the mode "*Solar Power*" is activated. This mode engages the corresponding equation system, which represents the solar panels. Thus, in the *Sunny* context, the solar panels are activated and begin producing electricity from solar energy. In the evening (*Context* = *Night*), solar radiation is absent, and the mode "*Standby*" with its corresponding equation system is activated (while other modes and their equation systems are deactivated). The equation system for this mode represents a physical state where the solar panels

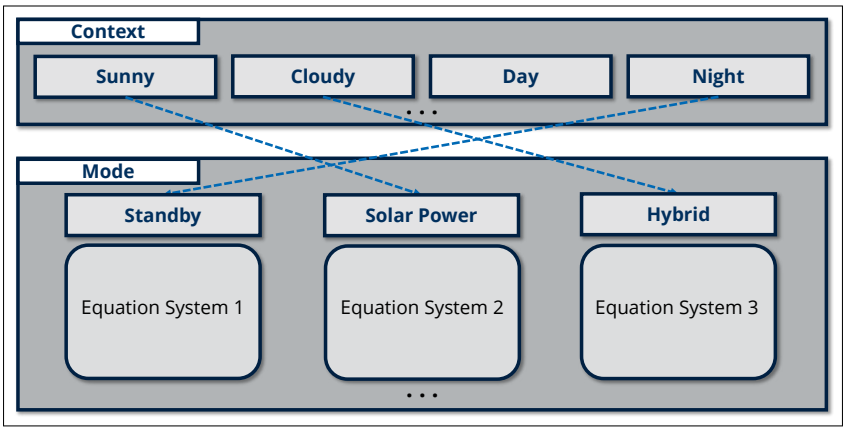


Figure 1. A minimal example of using contexts to control different modes.

are inactive. In realistic applications, multiple systems can be switched simultaneously. All of this occurs at runtime (simulation time); there is no need to power off the system, switch modes, and then re-initialize and restart the simulation. Typical application fields where VSS can be beneficial include circuit switching, mechanical elements breaking apart, systems with clutches, different rocket stages, and robot reconfigurations.

Modern modeling environments handle complex physical systems using equation-based modeling languages, also known as acausal modeling languages. The Modelica language (Fritzson and Engelson 1998) (Modelica, for short) is the state-of-the-art example, widely used in various industries like energy grids (Senkel et al. 2021) and building systems (Wetter et al. 2014). However, like most equation-based modeling languages, the possibilities for VSS in Modelica (current version 3.6) are limited. Only a few frameworks have been designed to support VSS in Modelica, and in most cases, switching modes at runtime fails. Zimmer 2010 attributes these limitations to the static treatment of the differential-algebraic equations (DAEs) and the lack of expressiveness in the Modelica language.

A classic example of VSS is the "breaking pendulum" (Figure 2) which can be described as follows: a ball attached to a string moves as a pendulum initially (mode 1). After a few seconds, the string breaks, and the ball moves as if in free fall (mode 2). This example includes two modes, each corresponding to a different model: one describes the pendulum (Listing 2), and the other describes the free fall of the ball (Listing 3). Mode switching is triggered by *time*. It is important to note that the two models have different equation systems. Classical Modelica environments, such as OpenModelica (Fritzson, Pop, et al. 2022) and Dymola (Elmqvist 1979), which are based on the current Modelica specification (Modelica Association 2023), cannot handle this situation effectively. The simulation will fail at the moment when the modes are switched. Typically, different modes are modeled and simulated separately. Ideally, developers would model and integrate different modes within a single model.

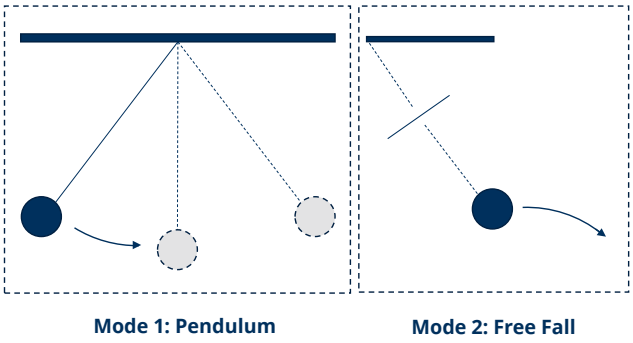


Figure 2. Two modes of the breaking pendulum.

1.3 Problem Statement and Research Objectives

During literature research, two main problems related to VSS in Modelica have been identified:

1. Despite various frameworks targeting VSS in Modelica (Table 1) and Modelica-like environments, it remains unclear which solution contributes to which VSS type. A detailed overview is lacking.

2. Enabling contexts significantly impacts the control of variability and the realization of context-aware systems. However, the idea of COP has not yet been implemented in Modelica. How to introduce contexts in Modelica remains an interesting research question.

This work aims to address these two problems. The main goals are:

1. To provide a clear classification of VSS in Modelica and an overview of frameworks supporting their VSS types.

2. To propose the extension *ContextModelica* that introduces contexts into Modelica.

Table 1. Frameworks targeting VSS in Modelica and other equation-based modeling environments.

Frameworks
Mosilab (Nytsch-Geusen et al. 2005)
Sol (Zimmer 2010)
Hydra (Giorgidze 2012)
Modelyze (Broman and Siek 2012)
DySMo (Möckel, Mehlhase, and Nytsch-Geusen 2015)
MoVasE (Esperon, Mehlhase, and Karbe 2015)
PyVSM (Stüber 2017)
Modia.jl (Elmqvist, Neumayr, and Otter 2018)
OM.jl (Tinnerholm, Pop, Sjölund, et al. 2020)
ModelingToolkit.jl (Ma et al. 2021)

1.4 Structure of the Work

Section 2 provides a detailed explanation and classification of VSS in Modelica. Section 3 summarizes various frameworks designed to support different VSS types in Modelica or Modelica-like environments, offering an overview to understand which framework addresses which specific VSS problem. In Section 4, we propose ContextModelica, developed based on OpenModelica.jl, in short, OM.jl[1] (Tinnerholm, Pop, Sjölund, et al. 2020). Our extension combines Modelica with the concept of context from the language engineering field. We demonstrate ContextModelica with an example and discuss the current challenges. This section also explores the potential benefits of integrating context-aware features into existing Modelica models. Finally, Section 5 presents the conclusions and an outlook, including discussions and suggestions for future research.

2 VSS in Modelica

To provide an extension that enables modeling and managing VSS using contexts in Modelica, the first step is to understand what VSS are. Definitions of VSS vary slightly across different domains in the literature. VSS were first introduced by Utkin 1977. Mehlhase 2015 offers an overview of publications with definitions related to VSS. In short, VSS can be summarized as "structural change during runtime (simulation time)". In Modelica, VSS correspond to the switching of equation systems based on different situations while the simulation is running. However, different types of structural change during runtime exist, and Modelica supports only some of them in a limited way. Consequently, various frameworks have been designed to enhance VSS possibilities in Modelica and Modelica-like environments. Unfortunately, since the types of structural changes during runtime have not been discussed in detail, it remains unclear, which framework addresses which specific VSS type.

[1]https://github.com/JKRT/OM.jl (A Modelica compiler written in Julia)

To the best of our knowledge, **variables** and **differential index**[2] have the most impact on realizing VSS in Modelica. In general, three different types can be distinguished based on these two factors:

1. **Two modes share the same variables and differential index.** Thus, the structural change does not introduce new variables, and the differential index remains unchanged.

2. **Two modes have different variables but the same differential index.** In this case, the structural change introduces new variables and corresponding equations, while the differential index stays the same.

3. **Two modes have the same variables but different differential indices.** Here, the structural change involves a change in the differential index.

At this point, some issues related to VSS arise in Modelica (Benveniste, Caillaud, et al. 2019). In most cases, the simulation fails when switching from one mode to another, primarily because Modelica is static and the compiler cannot handle types 2 and 3 at runtime.

Regarding type 2 where each mode contains a different set of variables, there are several sub-types. The number of variables may either change or remain the same during the mode transition. For simplicity, this work does not specify different sub-types of variables.

3 State of the Art

This section provides an overview of the frameworks for supporting different VSS types in Modelica and other equation-based modeling environments (most of them are Modelica-like). Table 2 summarizes the applicability of approaches for different VSS types. All approaches can be used for type 1.

Mosilab (Nytsch-Geusen et al. 2005) uses a Modelica extension to describe the models and transitions through a state chart.

Mosilab supports types 1 and 2 but does not support type 3, as the environment only simulates index-0 models and lacks an index-reduction mechanism.

Sol (Zimmer 2010) is an experimental language designed as a proof of concept to support variable-structure models using dynamic casualization.

Although Sol is similar to Modelica, it is a separate language. It enables the modeling of VSS with *SolSim* and allows changes to the differential index.

[2]For a general differential algebraic equation (DAE) $F(t,x,x') = 0$, the differential index is defined as "the minimum number of differentiations required to translate the DAE system into a system of the ordinary differential equations (ODEs)" (Campbell and Gear 1995) (Benveniste, Bourke, et al. 2014). Thus, ODEs have a differential index of 0.

Hydra (Giorgidze 2012) is an embedded acausal modeling language implemented in Haskell according to the paradigm of functional hybrid modeling. Hydra lacks the object-oriented characteristic present in modeling languages such as Modelica.

Modelyze (Modeling Kernel Language) (Broman and Siek 2012) is a host language designed for embedding equation-based DSL based on gradual typing. Modelyze has been developed with OCaml.

Dymola extensions Elmqvist, Mattsson, and Otter 2014 and Mattsson, Otter, and Elmquvist 2015 present extensions of Dymola to enable the possibility of VSS. In the first work VSS with varying DAE index is not supported, the second work extends the Pantelides algorithm (Pantelides 1988) and allows VSS with varying DAE index. These extensions have limited functionality. They have only been tested with simple examples. Because of this, these extensions have not been implemented in the latest stable release of Dymola yet (as of May 2024).

DySMo (Dynamic Structure Modeling) (Mehlhase 2015) is a Python application that enables the simulation of VSS. A case study by Möckel, Mehlhase, and Nytsch-Geusen 2015 demonstrated the use of DySMo in the context of building simulation.

DySMo is a script-based approach designed for simulating VSS rather than modeling them. In this approach, different models are simulated separately, and their results are then integrated using Python.

MoVasE (Modelica Variable-structure Editor) (Esperon, Mehlhase, and Karbe 2015) enables structural changes to models by defining conditional exchanges externally.

Compared to DySMo, MoVasE has the advantage of not requiring manual creation and maintenance of all modes and transitions. However, this approach still has limitations in terms of the dynamic addition and removal of components.

PyVSM (Python Variable-structure Model) (Stüber 2017) is another script-based approach using Dymola's Python interface. The idea is the same as in DySMo: Using Modelica for simulating different modes and Python for switching between them.

Modia.jl (Elmqvist, Neumayr, and Otter 2018) is a Modelica-like software written in Julia. It has been initiated by the inventor of Dymola. After several attempts to support VSS in Dymola, as discussed in Elmqvist, Mattsson, and Otter 2014 and Mattsson, Otter, and Elmquvist 2015, the authors explored Julia's potential in modeling. Modia.jl utilizes predefined acausal components, as described in Neumayr and Otter 2023.

OM.jl OpenModelica.jl (Tinnerholm, Pop, Sjölund, et al. 2020) is a Modelica compiler written in Julia, developed by the OpenModelica development team from Linköping, Sweden. Leveraging Julia's just-in-time (JIT) compilation and multi-dispatch features, OM.jl supports modeling VSS. It can also connect ModelingToolkit.jl with Modelica (Tinnerholm, Pop, Heuermann, et al. 2021).

ModelingToolkit.jl (Ma et al. 2021)[3] is a Julia package for modeling and simulation that integrates Julia's ecosystem with the modeling. Inspired by Modelica, it features a Modelica-like syntax. Compared to Modelica, ModelingToolkit.jl supports not only ODEs and DAEs, but also partial differential equations (PDEs), stochastic differential equations (SDEs), and other types of equation systems. Like Modia.jl and OM.jl, ModelingToolkit.jl supports various VSS types due to Julia's capabilities.

Table 2. Overview of Modelica-based and Modelica-like frameworks for different VSS types. ✓ indicates that the approach supports this VSS type, while ✗ indicates that it does not.

VSS Types	*Applicability of Approaches*
Type 1	All approaches (✓)
Type 2	Standard Modelica (✗) Mosilab (✓, but only index 0) Sol (✓) Dymola extensions (✓) DySMo (✓) MoVasE (unknown, lack of literature) PyVSM (✓) Modia.jl (✓) OM.jl (✓) ModelingToolkit.jl (✓)
Type 3	Standard Modelica(✗) Mosilab (✗) Sol (✓) Dymola extensions (✓) DySMo (✓) MoVasE (unknown, lack of literature) PyVSM (✓) Modia.jl (✓) OM.jl (✓) ModelingToolkit.jl (✓)

Despite some approaches supporting all VSS types, **none of the above approaches support the explicit specification of contexts and their transitions.**

[3] https://docs.sciml.ai/ModelingToolkit/dev

4 ContextModelica

The previous section introduced different approaches to support VSS in Modelica. However, none of these approaches provide semantics to support contexts and context management within Modelica for modeling and managing context-aware systems. Therefore, we propose the *ContextModelica* - an extension of the Modelica language based on OM.jl, which already includes capabilities for supporting structural transitions between variants in Modelica.

This section will examine the concepts behind this extension in more detail and describes how it can be applied to manage VSS, along with an illustrative example. Finally, we will discuss the current challenges and limitations of the extension.

4.1 Units of Variability

In a software language, variability relies on *variation points* (Webber and Gomaa 2004). The Variation Point Model (VPM) is designed to model variation points contained in reusable software components (Webber and Gomaa 2004). Variation points are the units of variation in a specification of a program. For Modelica, several kinds of variation points can be considered, some of which are intended by the language designers.

Class and subclass Modelica is a *static* object-oriented language in which classes can be specialized by subclasses. These subclasses can be defined in variations. Therefore, a class is a static variation point in Modelica, and it is common to replace a class with one of the members of its transitively defined derived subclasses.

Equation block An equation block defines a set of variables or derivatives, constituting the provided interface of the block. In Modelica, equation blocks can be guarded by `if/else` and `when` statements, allowing them to be dynamically varied (dynamic variation point).

Equation A single equation can also be a variation point. It is a special case of a block variation point.

As discussed, modes in VSS may differ in variables and the differential index. Modes relate to variation points in that these constraints about variables and the differential index must hold also for all variants of a Modelica variation point. This means that for any pointwise variation, the VSS types 1-3 can be distinguished. For instance, a variation point of type 1 can be a class, block, or equational variation point.

For a *class variation point*, VSS type 1 is the simplest type, where polymorphism of the class resolves the transition to another subclass. At runtime, the subclass can be varied by wrapping all variant subclasses in a simple case expression. In Modelica, polymorphism is not available because subclasses must be selected statically. A *block variation point* of VSS type 1 can also be handled in Modelica if the block is encapsulated by a case expression. All frameworks discussed in section 3 offer dynamic block variation. Usually, an *equational variation point* of VSS type 1 can also be managed because one equation is a simple equational block.

We will demonstrate later how ContextModelica can be employed for the variation points described above.

4.2 Context

While addressing the lack of VSS manageability in Modelica, one potential solution is to introduce a language concept called a *context*. Contexts are common used in software development to separate concerns (Hirschfeld, Costanza, and Nierstrasz 2008). By integrating contexts into Modelica, we can achieve better code structure and improved manageability of VSS.

To this end, we have extended the Modelica language to include this concept and thus created ContextModelica. This extension introduces two new semantics, as shown in Listing 1. First, all modes can be listed in a separate section using the keyword "context", with each mode associated with corresponding condition. Second, the new semantics allow for the addition of multiple equation systems, with each system labeled by the corresponding context. This means that the equation system represents the model or mode when the context is active. Additionally, the set of contexts must include an initial state, which is the mode that is active at the start of the simulation.

The advantage of mapping contexts to their applicable conditions is that developers no longer need to manage the resulting transitions between contexts. This separation of concerns leads to cleaner and more readable code, particularly when compared to the use of `if/else` and `when` statements in large-scale systems.

Listing 1. Semantics in ContextModelica.

```
model ExampleModel
    /*parameters & variables*/

    equation on initial (ContextA)
        /*corresponding equations*/

    equation on ContextB
        /*corresponding equations*/

    equation on ContextC
        /*corresponding equations*/

        ...

    context
        initial on /*condition*/;
        ContextB on /*condition*/;
        ContextC on /*condition*/;
        ...
    end context;

end ExampleModel;
```

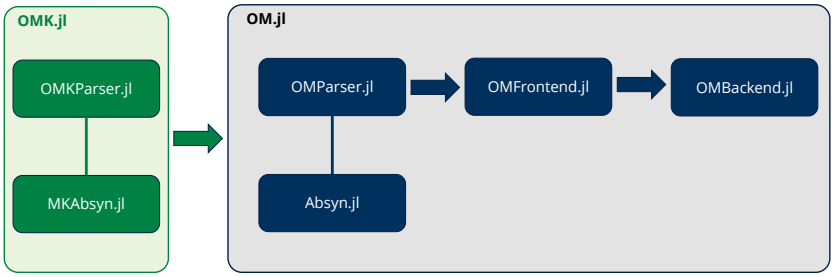


Figure 3. Structure of ContextModelica. The blue section represents the original OM.jl, while the green section indicates the added preprocessor.

4.3 How It Works

ContextModelica[4] is implemented as a language extension of Modelica in Julia (OM.jl). It benefits from the structural transitions available in OM.jl, which can be used to construct state machines in Modelica. More precisely, the defined contexts and their associated conditions from a given ContextModelica model are translated into a *context transition automaton* comprising states and transitions, representing the possible changes of the contexts. This results in *n*(n-1)* state transitions, where *n* is the number of existing states. The context transition automaton can be realized through the dynamic recompilation features of OM.jl.

The structure of ContextModelica is illustrated in Figure 3. *OMK.jl* functions as a preprocessor for OM.jl and was developed by reusing components of OM.jl, including the ANTLR parser generator (Parr and Quong 1995) and the abstract syntax tree (AST) module. Both have been slightly modified to support the new semantics introduced in ContextModelica, specifically the definition of contexts with their corresponding conditions and the equation systems that can be tagged with context labels. In addition to these modules, we added a code generator backed by some OpenModelica packages. It traverses the AST constructed by the parser and then generates the corresponding state machine using the syntax of structural transitions provided by OM.jl. Therefore the code generator gathers context labels, active conditions, and the associated equation sets, creating sub-models within a larger model. Afterward, the transitions supported by OM.jl are inserted. The resulting state machine is an undirected graph where every state has a transition to every other state. The output can then be passed to OM.jl, which generates the corresponding Julia code for further simulation.

ContextModelica inherits the ability of OM.jl to support the change of differential index, thus supporting type 3. Type 2 is currently not supported because all variables and parameters share a common set. This is due to the focus on varying the actual behavior in the individual contexts, which is primarily determined by the equation systems. Future modifications should allow separate definitions for local variables and parameters. In conclusion, ContextModelica supports two VSS types: types 1 and 3.

[4]https://github.com/dev-manuel/OMK.jl

4.4 Example

We demonstrate the proposed ContextModelica using the classical "breaking pendulum" model, as shown in Figure 2. Listing 2 and Listing 3 show the Modelica models for the "Pendulum" mode and the "FreeFall" mode respectively. With the classical Modelica software which has limited functionality of VSS the developers need to model and simulate them separately. In ContextModelica, these two models can be integrated into one model as VSS with two modes, as Listing 4 shows. Two different equation sets together with switch mechanisms will be defined in the model. The outcome of the preprocessor is shown in Listing 5. It includes the whole context transition automaton containing the models and transitions required by OM.jl for further simulation. This model corresponds to VSS type 3 because the differential index of the "Pendulum" and "FreeFall" modes are different. The result is shown in Figure 4.

Listing 2. A pendulum model written in Modelica.

```
model Pendulum
    parameter Real g = 9.81;
    parameter Real L = sqrt(200);
    Real x(start = 10);
    Real y(start = 10);
    Real vx; Real vy;
    Real phi(start=1.0); Real phid;
    equation
        der(phi) = phid;
        der(x) = vx;
        der(y) = vy;
        x = L * sin(phi);
        y = -L * cos(phi);
        der(phid) = -g / L * sin(phi);
end Pendulum;
```

Listing 3. A free fall model written in Modelica.

```
model FreeFall
    Real x; Real y; Real vx; Real vy;
    parameter Real g = 9.81;
    parameter Real vx0 = 0.0;
    equation
        der(x) = vx;
        der(y) = vy;
        der(vx) = vx0;
        der(vy) = -g;
end FreeFall;
```

Listing 4. Syntax for modeling the "Breaking Pendulum" model in ContextModelica. This model corresponds to VSS type 3.

```
model BreakingPendulum
    Real x; Real y; Real vx; Real vy;
    Real phi(start=1.0); Real phid;
    parameter Real g = 9.81;
    parameter Real vx0 = 0.0;
    parameter Real L = sqrt(200);

    //context = Initial (Pendulum)
    equation on initial
        der(phi) = phid;
        der(x) = vx;
        der(y) = vy;
        x = L * sin(phi);
        y = -L * cos(phi);
        der(phid) = -g / L * sin(phi);
    //context = FreeFall
    equation on FreeFall
        der(x) = vx;
        der(y) = vy;
        der(vx) = vx0;
        der(vy) = -g;

    //switch of contexts
    context
        initial on t < 5;
        FreeFall on t >= 5;
    end context;

end BreakingPendulum;
```

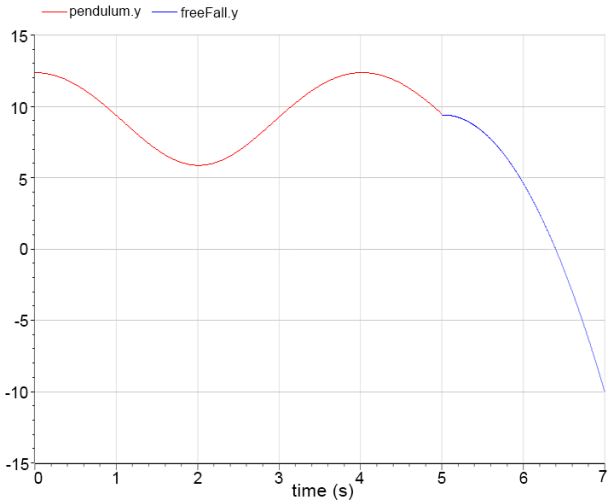


Figure 4. Simulation result of the "breaking pendulum" model with ContextModelica.

Compare the same "breaking pendulum" model implemented in ContextModelica (Listing 4) and OM.jl (Listing 6). ContextModelica enables the explicit definition of contexts directly while defining the corresponding equation systems for each mode, eliminating the need to define structural modes separately. The transition process is also simplified. In OM.jl, the transition process must be defined with a separate equation system, while in ContextModelica, this is unnecessary. The explicit definition of contexts in ContextModelica results in a cleaner structure and readable code for realizing and managing VSS, especially in large context-aware systems.

Listing 5. Transpiled model compatible with OM.jl.

```
model BreakingPendulum
    // BreakingPendulum = BP
    structuralmode
        BP__Context_Initial
        bP__Context_Initial_instance;
    structuralmode
        BP_FreeFall
        bP_FreeFall_instance;

    Real x; Real y; Real vx; Real vy;
    Real phi(start=1.0); Real phid;
    parameter Real g=9.81;
    parameter Real vx0=0.0;
    parameter Real L = sqrt(200);

    model BP__Context_Initial
        equation
            /*equation set*/
    end BP__Context_Initial;

    model BP_FreeFall
        equation
            /*equation set*/
    end BP_FreeFall;

    equation
    initialStructuralState(
        bP__Context_Initial_instance);
    structuralTransition(
        bP__Context_Initial_instance,
        bP_FreeFall_instance,
        t >= 5);
end BreakingPendulum;
```

Listing 6. Syntax of "Breaking Pendulum" model in OM.jl

```
model BreakingPendulum

    model FreeFall
        /*parameters*/
        /*variables*/
    equation
        /*equations*/
    end FreeFall;

    model Pendulum
        /*parameters*/
        /*variables*/
    equation
        /*equations*/
    end Pendulum;

structuralmode Pendulum pendulum;
structuralmode FreeFall freeFall;

equation
    initialStructuralState(pendulum);
    structuralTransition(
        pendulum, freeFall,
        t >= 5
        );

end BreakingPendulum;
```

The example also shows how ContextModelica can be deployed to the block and the equational variation points. Each equation block, or single equation in specific cases, will be varied by switching on/off different contexts/-modes. In conclusion, ContextModelica supports VSS types 1 and 3 as well as block and equational variation points, as summarized in Table 3.

Table 3. Supported VSS types and variation points of ContextModelica.

Supported VSS types	*Supported variation points*
VSS type 1 ✓	Class and subclass
VSS type 2	Equation Block ✓
VSS type 3 ✓	Equation ✓

4.5 Challenges

One challenge is synchronizing variable values when transitioning from one mode to another. Currently, all variables and parameters in a model must be defined as global ones, making them valid across all modes. For example, in Listing 4 the variables `phi` and `phid` are only used in the first mode (Pendulum), which results in redundant variables for the second mode (FreeFall). This limitation means that specific variables and parameters cannot be defined within their corresponding modes. This characteristic leads to the lack of support for VSS type 2 and this may negatively impact the performance, especially in large systems. Another challenge is that, currently, the OM.jl version only supports structural transitions in the top-level model of a program. As a result, one of the limitations is that only the top-level model of a program can have user-defined contexts. This means the use of contexts in submodels is not supported at the moment (e.g. different contexts can be defined under the "BreakingPendulum" model, but no contexts could be defined under the "FreeFall" submodel). Still, the number of modes/contexts in the top-level model is not limited. Because of this, ContextModelica does not fully support class/subclass variation point. Another challenge is the overlap of transition conditions. If two conditions can be evaluated to be true at the same time, unexpected behavior may occur because the transition is not deterministic. For now, the developers need to ensure that the conditions are mutually exclusive to avoid such issues.

5 Conclusion and Future Work

To fully demonstrate and explain the restricted VSS feature in Modelica, we have discussed the background and explained how combining COP with Modelica can help manage variability in context-aware systems. Modeling variability using contexts reveals the switch mechanisms, aiding developers in understanding and maintaining models more effectively. Following this, we presented a classification of VSS types as well as a detailed overview of various frameworks designed to support VSS in Modelica or Modelica-like environments, covering different VSS types. Unfortunately, none of these frameworks support the explicit specification of contexts, making it difficult to manage variability in context-aware environments. Therefore, we proposed the ContextModelica, a context-oriented extension of Modelica *ContextModelica* with easy-to-understand semantics. This approach also avoids the complexities of using `if/else` and `when` statements in large-scale systems. ContextModelica supports VSS types 1 and 3, as well as "equation block" and "equation" variation points. **To our knowledge, the proposed ContextModelica is the first approach that introduces the concept of context and COP into Modelica.** It extends the Modelica language with the explicit specification of context, providing a novel solution to model and manage variability in context-aware systems.

Note that the VSS can be quite complex, and this complexity must be addressed in future work. On one hand, contexts can either be mutually exclusive or overlapping, which adds complexity to our implementation. We need to carefully consider and address these scenarios to ensure that our system can handle both exclusive and non-exclusive contexts effectively. On the other hand, in our example, we only covered contexts that are time-relevant. However, there can also be time-irrelevant contexts. For instance, after the "FreeFall" mode, when the ball hits the ground and switches to the "BouncingBall" mode, it is challenging to define the exact moment the ball hits the ground. In such cases, time-irrelevant contexts are useful, e.g., when the ball hits the ground and its acceleration vectors changes direction, at this moment, the third mode "BouncingBall" is activated, as shown in Listing 7. While ContextModelica can technically handle this scenario, we do not consider it a verified example without thorough testing. More tests are needed to explore potential issues that might arise in such cases.

Listing 7. Syntax for adding the "BouncingBall" mode.

```
model BreakingPendulum
    /*parameters*/
    /*variables*/

    equation on initial
        /*equations*/

    equation on FreeFall
        /*equations*/

    equation on BouncingBall
        /*equations*/

    //switch of contexts
    context
        initial on t < 5;
        FreeFall on t >= 5;
        BouncingBall on vy < 0;
    end context;

end BreakingPendulum;
```


10.3384/ECP20753 DOI

Another complexity of VSS is the concept of **unilateral constraints**, as explored in the works of Ch Glocker and Pfeiffer 1992, Friedrich Pfeiffer and Christoph Glocker 2000, Enge and Maißer 2005, and Enge-Rosenblatt 2017, as well as in the PhD theses of Christoph Glocker 1995 and Enge 2005, where the switching between modes is driven by these constraints. For example, this occurs when a normal "Pendulum" mode transitions to a string-bound free-flying mode Figure 5, or in switching diodes used in power electronics. In the first example, defining the exact point of transition is difficult, unlike in a scenario involving a pendulum string breaking, which can be clearly identified. The transitions in the second example differ depending on the direction of the switching.

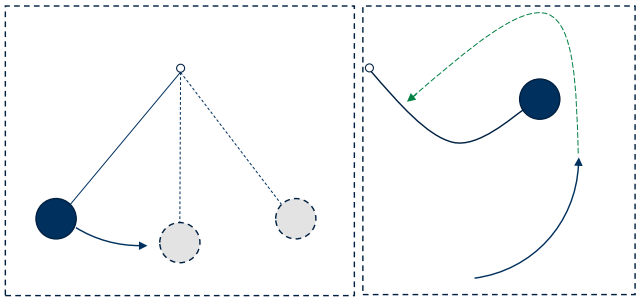


Figure 5. Transition from the pendulum mode to the string-bound free-flying mode.

The two challenges discussed in Section 4.5 are also crucial for future work. Firstly, VSS type 2 is not supported since all variables and parameters should be defined as global variables and parameters, this may lead to redundancy of variables and thus performance issues, especially in large systems. Secondly, only contexts in the top-level model are supported. It would be more practical to also enable defining and using contexts in submodels. This will also allow ContextModelica to support the *class and subclass* variation point. It should be noted that OM.jl supports both structural transitions and recompilation constructs. However, currently, ContextModelica only supports structural transitions. Implementing the recompilation constructs in ContextModelica would help solve these two challenges and improve the performance significantly. Listing 8 shows an example of recompilation constructs used in OM.jl for the "breaking pendulum" model[5]. In this example, variables and parameters for different submodels can be defined separately in the submodels rather than as global variables and parameters. Implementing recompilation constructs to support VSS type 2 and nested contexts in submodels should be considered for future development. Furthermore, more practical and industry-oriented examples should be examined using ContextModelica.

Listing 8. Syntax using recompilation constructs in OM.jl.

```
model BreakingPendulum

    model FreeFall
    /*parameters & variables*/
    equation
        /*equations*/
    end FreeFall;

    model Pendulum
    /*parameters & variables*/
    equation
        /*equations*/
    end Pendulum;

parameter Boolean breaks = false;
FreeFall freeFall if breaks;
Pendulum pendulum if not breaks;

equation
    when 5.0 <= time then
        recompilation(breaks, true);
    end when;

end BreakingPendulum;
```

Acknowledgements

The authors would like to thank the Boysen–TU Dresden–Research Training Group for the financial and general support that has made this contribution possible. The Research Training Group is co-financed by the Friedrich and Elisabeth Boysen Foundation and TU Dresden. This work is also financially supported by the German Research Foundation (Project No. 453596084 - SFB/TRR 339). The authors would also like to thank Prof. Peter Fritzson and Dr. Adrian Pop from Linköping University for their valuable discussions, as well as Anastasiia Korolenko for her contribution to the test suite.

References

Benveniste, Albert, Timothy Bourke, et al. (2014). *On the index of multi-mode DAE Systems (also called Hybrid DAE Systems)*. Research Report. Inria.

Benveniste, Albert, Benoıt Caillaud, et al. (2019). "Multi-mode DAE models-challenges, theory and implementation". In: *Computing and Software Science: state of the Art and Perspectives*, pp. 283–310.

Broman, David and Jeremy G Siek (2012). "Modelyze: a gradually typed host language for embedding equation-based modeling languages". In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-173*.

Campbell, Stephen L and C William Gear (1995). "The index of general nonlinear DAEs". In: *Numerische Mathematik* 72.2, pp. 173–196.

Dey, Anind K, Gregory D Abowd, et al. (2000). "The context toolkit: Aiding the development of context-aware applications". In: *Workshop on Software Engineering for wearable and pervasive computing*. University of Washington Seatle, pp. 431–441.

[5] https://github.com/JKRT/OM.jl/tree/master/test/Models/VSS

Elmqvist, Hilding (1979). "DYMOLA-a structured model language for large continuous systems". In: Summer Computer Simulation Conference, Toronto, Canada.

Elmqvist, Hilding, Sven Erik Mattsson, and Martin Otter (2014). "Modelica extensions for multi-mode DAE systems". In: *Proceedings of the 10th international Modelica conference*. 96. Linköping University Electronic Press Linköping, Sweden, pp. 183–193.

Elmqvist, Hilding, Andrea Neumayr, and Martin Otter (2018). "Modia-dynamic modeling and simulation with julia". In: Juliacon, University College London, UK.

Elyasaf, Achiya, Nicolás Cardozo, and Arnon Sturm (2023). "A framework for analyzing context-oriented programming languages". In: *Journal of Systems and Software* 198, p. 111614.

Elyasaf, Achiya and Arnon Sturm (2022). "Modeling Context-aware Systems: A Conceptualized Framework." In: *MODELSWARD*, pp. 26–35.

Elyasaf, Achiya and Arnon Sturm (2023). "A Framework for Analyzing Modeling Languages for Context-Aware Systems". In: *SN Computer Science* 4.2, p. 149.

Enge, Olaf (2005). "Analyse und Synthese elektromechanischer Systeme". PhD thesis. Technische Universitaet Chemnitz (Germany).

Enge, Olaf and Peter Maißer (2005). "Modelling electromechanical systems with electrical switching components using the linear complementarity problem". In: *Multibody System Dynamics* 13, pp. 421–445.

Enge-Rosenblatt, Olaf (2017). "Equation-based modelling and simulation of hybrid systems". In: *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pp. 27–36.

Esperon, Daniel Gomez, Alexandra Mehlhase, and Thomas Karbe (2015). "Appending variable-structure to modelica models (WIP)". In: *Proceedings of the Conference on Summer Computer Simulation*, pp. 1–6.

Fritzson, Peter and Vadim Engelson (1998). "Modelica—A unified object-oriented language for system modeling and simulation". In: *ECOOP'98—Object-Oriented Programming: 12th European Conference Brussels, Belgium, July 20–24, 1998 Proceedings 12*. Springer, pp. 67–90.

Fritzson, Peter, Adrian Pop, et al. (2022). "The OpenModelica integrated environment for modeling, simulation, and model-based development". In: Mic.

Giorgidze, George (2012). "First-class models: On a noncausal language for higher-order and structurally dynamic modelling and simulation". PhD thesis. University of Nottingham (England).

Glocker, Ch and F Pfeiffer (1992). "Dynamical systems with unilateral contacts". In: *Nonlinear Dynamics* 3, pp. 245–259.

Glocker, Christoph (1995). "Dynamik von Starrkörpersystemen mit Reibung und Stößen". PhD thesis. Technische Universitaet Muenchen (Germany).

Hirschfeld, Robert, Pascal Costanza, and Oscar Nierstrasz (2008). "Context-oriented programming". In: *Journal of Object technology* 7.3, pp. 125–151.

Ma, Yingbo et al. (2021). "Modelingtoolkit: A composable graph transformation system for equation-based modeling". In: *arXiv preprint arXiv:2103.05244*.

Mattsson, Sven Erik, Martin Otter, and Hilding Elmquvist (2015). "Multi-mode DAE systems with varying index". In: *Proceedings of the 11th International Modelica Conference*, pp. 89–98.

Mehlhase, Alexandra (2015). "Konzepte für die Modellierung und Simulation strukturvariabler Modelle". PhD thesis. Technische Universitaet Berlin (Germany).

Möckel, Jens, Alexandra Mehlhase, and Christoph Nytsch-Geusen (2015). "Exploiting variable-structure models in the context of building simulations within Modelica". In: *Proceedings of BS2015. International Building Performance Simulation Association*.

Modelica Association (2023-03). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.6*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.6/MLS.pdf.

Neumayr, Andrea and Martin Otter (2023). "Variable Structure System Simulation via Predefined Acausal Components". In: *Proceedings of the 15th International Modelica Conference*.

Nytsch-Geusen, Christoph et al. (2005). "MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics". In: *Proceedings of the 4th International Modelica Conference TU Hamburg-Harburg*. Vol. 2. Citeseer.

Pantelides, Constantinos C. (1988). "The Consistent Initialization of Differential-Algebraic Systems". In: *SIAM Journal on Scientific and Statistical Computing* 9.2, pp. 213–231. DOI: 10.1137/0909014.

Parr, Terence J. and Russell W. Quong (1995). "ANTLR: A predicated-LL (k) parser generator". In: *Software: Practice and Experience* 25.7, pp. 789–810.

Pfeiffer, Friedrich and Christoph Glocker (2000). *Multibody dynamics with unilateral contacts*. Vol. 421. Springer Science & Business Media.

Senkel, Anne et al. (2021). "Status of the transient library: Transient simulation of complex integrated energy systems". In: *Modelica Conferences*, pp. 187–196.

Stüber, Moritz (2017). "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*, pp. 132–031.

Tinnerholm, John, Adrian Pop, Andreas Heuermann, et al. (2021). "OpenModelica. jl: A modular and extensible Modelica compiler framework in Julia targeting ModelingToolkit. jl". In: *Modelica Conferences*, pp. 109–117.

Tinnerholm, John, Adrian Pop, Martin Sjölund, et al. (2020). "Towards an Open-Source Modelica Compiler in Julia". In: *Asian Modelica Conference 2020, Tokyo, Japan*. Linköping University Electronic Press, pp. 143–151.

Utkin, Vadim (1977). "Variable structure systems with sliding modes". In: *IEEE Transactions on Automatic control* 22.2, pp. 212–222.

Webber, Diana L and Hassan Gomaa (2004). "Modeling variability in software product lines with the variation point model". In: *Science of Computer Programming* 53.3, pp. 305–331.

Wetter, Michael et al. (2014). "Modelica buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Zimmer, Dirk (2010). "Equation-based modeling of variable-structure systems". PhD thesis. ETH Zurich (Switzerland).

Building Power System Models for Stability and Control Design Analysis using Modelica and the OpenIPSL

Srijita Bhattacharjee[1] Luigi Vanfretti[1] Fernando Fachini[2]

[1]Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, NY, USA
{bhatts10, vanfrl}@rpi.edu
[2]Electric Transmission Strategic Initiatives, Dominion Energy, VA, USA
fernando.fachini@dominionenergy.com

Abstract

Ensuring the stability of complex power system models is a critical challenge in the field of electrical power engineering, and the tuning of Power System Stabilizers (PSS) plays a pivotal role in this endeavor. Modelica, an open-access modeling language, emerges as a powerful tool for this purpose, due to its distinctive features that facilitate efficient power system modeling. This paper explores the capabilities of Modelica using the OpenIPSL library to create models to analyze control system designs developed for a multi-machine power system model. It particularly focuses on using the features of Modelica for the linearization, control-oriented analysis, and time-simulation of the model. The results demonstrate the effectiveness of using Modelica for control system design analysis and performing linear model-based analysis. This work aims to show how Modelica can be used to perform these tasks on a single platform efficiently, thereby streamlining the process of power system design and analysis.

Keywords: Power System Modeling, Linearization, Stability Analysis, Controller Design Analysis, OpenIPSL

1 Introduction

1.1 Motivation

Modern power systems exhibit a complex architecture that requires the use of both physics-based models for sophisticated control system designs. The design of robust control systems is crucial to ensure reliable grid operation, which facilitates the management of complex power system dynamics. A significant aspect of this involves conducting stability analysis and tuning of Power System Stabilizers (PSS), which are essential to damp electromechanical oscillations that can adversely affect system stability (F. J. De Marco, Martins, and Ferraz 2012). To address this need of developing models for control design and analysis, Modelica, in conjunction with the OpenIPSL library, has been used effectively to create a detailed University Campus Microgrid model, demonstrating its effectiveness in linear analysis, which is often challenging with traditional power system analysis tools (Fachini, Bhattacharjee, et al. 2023).

This work explores the capabilities of Modelica (Fritzson and Engelson 1998), and the Dymola tool (Brück et al. 2002), to develop power system models that are suitable for control system design and analysis. The model developed here emerges from the power system literature (F. De Marco, Rullo, and Martins 2021), which can be used to address intra-plant and inter-area oscillations when considering a power plant with multiple machines. The developed can be further explored to create a more detailed design for specific control tasks beyond those for which it was originally developed (F. J. De Marco, Martins, and Ferraz 2012).

1.2 Background and Related Works

The evolution of power system analysis has advanced computing technologies, notably through the development of software tools designed to improve the accuracy and efficiency of modeling and simulation (Isaacs 2017). This transition has been marked by significant shifts from traditional methods to more sophisticated software-oriented approaches that integrate the capabilities of modern computing frameworks (Guironnet et al. 2018). These advances have facilitated a detailed analysis of the dynamics of the power system, setting the stage for the addressing of the complex engineering challenges that arise from the adoption of renewable energy sources (Fachini, Luigi Vanfretti, et al. 2021; Plietzsch et al. 2022). The widespread commodification of computing technologies in the 1900s and 2000s led to the commercialization of domain-specific proprietary software and the rise of open-source software for power system analysis, often exploiting proprietary general-purpose computing languages and environments, that is, mainly tools based on `MATLAB` (Chow and Cheung 1992; Milano and Luigi Vanfretti 2009). This technological evolution set the stage for the addressing of more complex system challenges. One such challenge is linearization of power system models, a task that is complex due to the limitations of domain-specific tools, many of which lack symbolic linearization capabilities.

Many industry standard tools such as Siemens PSS/E depend on additional tools to perform numerical perturbations for linearization (Nikolaev et al. 2020). Likewise, CEPEL in Brazil has developed two independent tools, one for nonlinear time simulation and another for linear analysis (Martins et al. 2000). However, developers of both

tools need to provide symbolic expressions, which presents challenges in maintaining modeling consistency between the internal model descriptions within each tool (Luigi Vanfretti et al. 2013). Researchers have developed certain tools, such as PSAT, that support symbolic linearization (Milano 2005). However, they require users to input symbolic expressions and have a complete understanding of the source code of the software to modify or expand it (Li, Luigi Vanfretti, and Chompoobutrgool 2012). Other software tools such as DOME have been developed that utilize Python for power system analysis, demonstrating the viability and utility of scripting languages in this field, particularly for their modularity, ease of integration with various libraries and suitability for academia (Milano 2013).

In contrast, Modelica offers a compelling alternative, providing robust support for graphical modeling through software such as OpenModelica (al. 2020) and Dymola, thus significantly improving user experience and accessibility. Modelica emerges as a formidable language for power system modeling, especially when integrated with the Open-Instance Power System Library (OpenIPSL) (Baudette et al. 2018; De Castro et al. 2023), as elaborated in (Winkler 2017). Unlike the conventional power system approach of building a monolithic simulation tool, Modelica serves as a language that numerous software programs can implement, including proprietary options such as Dymola, Modelon Impact, Wolfram System Modeler, etc.

Along with these compliant tools, what the Modelica language offers is a unique advantage: it facilitates model linearization (including symbolic-based linearization) without the need of users or developers to specify additional (linear) models, excelling over other alternatives. This work explores the capabilities of Modelica symbolic analysis to *automatically* derive the linear model from *exactly* the same model used for non-linear time-domain simulation.

1.3 Paper Contribution

The main contributions of this paper are:

- To construct a multi-machine power system model by utilizing Modelica and the OpenIPSL library, specially designed to study intra-plant and inter-area oscillations (F. De Marco, Rullo, and Martins 2021).
- To extend the model in applying a control system design derived from the literature (F. J. De Marco, Martins, and Ferraz 2012).
- To demonstrate the benefits of object-oriented modeling for complex power system models.
- To demonstrate the application of the Modelica language and the OpenIPSL library for control system design analysis as a strong alternative to domain-specific power system tools with simulation results.

While the article aims to illustrate how Modelica and OpenIPSL can be used for the purposes stated above, some familiarity with the Modelica language would be beneficial to the reader. When necessary, the paper briefly introduces some language constructs and concepts used to guide the reader.

1.4 Paper Structure

The structure of the paper is as follows: Section 2 demonstrates the application of the object-oriented modeling technique to construct the different components of the system. Section 3 explains the process of creating the model used for linearization. Section 4 describes the nonlinear simulation of the multimachine system. The simulation results for the designed control system are presented in Section 5. Finally, Section 6 concludes the work and outlines the future direction of the work.

2 Object-Oriented System Modeling

Figure 1 shows the package structure of the three-machine infinite bus package `ThreeMIB` with several sub-packages, namely `Generation Units`, `Networks`, `Systems`, `PF_Data`, etc. Due to space constraints, only the `Generation Units` package is expanded in Figure 1 to show its internal structure. The sub-packages `Generation Units`, `Networks`, `Systems` are further explained below to describe the process of system modeling. This package is available in the Github repository: `https://github.com/ALSETLab/AMCONF2024_ThreeMIB`

2.1 Component Modeling

The OpenIPSL contains different component models that are built using object orientation. For components, object-oriented modeling can be illustrated using the instance for the `bus` component. As observed in the Modelica code excerpt in Listing 1, the `Bus` model extends a `partial model` named `pfComponent` from the `*.Electrical.Essentials` package. This inheritance approach is a hallmark of object-oriented modeling in Modelica, allowing the bus model to reutilize and extend predefined functionalities, such as initial parameter setups for algebraic variables that are crucial for setting initial state values in the models it comprises. Central to object-oriented design, attributes like `final enablev_0=true` and `final enableangle_0=true` are strategically enabled for initializing values, while `final enableP_0=false` is disabled to comply with KCL, illustrating the model's ability to customize through selective inheritance. The `PwPin` instance, named `p`, exemplifies encapsulation, initializing its algebraic variables, `vr` and `vi`, from `v_0` and `angle_0`. Furthermore, the variables `v` and `angle`, in Lines 13 and 14, which represent the magnitude and angle of the voltage, are managed within the model to reflect its link with other components of the system. The calculations in for voltage (Lines 13-14) and zero current enforcement (Lines 15-16) through `p.ir` and `p.ii` not only confirm the model's functionality but also ensure its integration within the larger system, underscoring the efficacy and adaptability of object-oriented modeling for complex

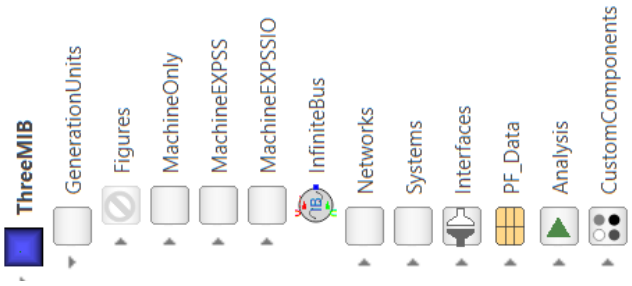


Figure 1. Package Structure

power systems.

Listing 1. Excerpt of the `OpenIPSL.Electrical.Buses.Bus` model

```
model Bus "Bus model"
  extends OpenIPSL.Electrical.Essentials.
      pfComponent(
    ...
    final enableP_0=false,
    ...
    final enablev_0=true,
    final enableangle_0=true);
  OpenIPSL.Interfaces.PwPin p(vr(start=v_0*
      cos(angle_0)), vi(start=v_0*sin(
      angle_0)));
  Types.PerUnit v(start=v_0) "Bus v.
      magnitude";
  Types.Angle angle(start=angle_0) "Bus v.
      angle";
  ...
equation
  v = sqrt(p.vr^2 + p.vi^2);
  angle = atan2(p.vi, p.vr);
  p.ir = 0;
  p.ii = 0;
  ...
end Bus;
```

Similarly to the bus component, other OpenIPSL component models are used to develop the system models described below. More information on the components available in OpenIPSL can be found in (L. Vanfretti et al. 2016; Baudette et al. 2018; De Castro et al. 2023).

2.2 System Modeling

This section explains how the object-oriented features of Modelica (Fritzson 2014) and OpenIPSL components are used to construct the multi-machine power system model from (F. De Marco, Rullo, and Martins 2021), as shown in Figure 4, allowing modular reusable components that simplify the design of the system model and enhance the simulation flexibility. The model consists of three generation units, six buses, two transmission lines, three transformers, three loads, and an infinite bus. Some of the sub-packages are explained below.

- *Generation Units*: The `GenerationUnits` subpackage is expanded in Figure 1 to show the internal structure. The sub-packages within offer various configurations of the three generation units named `G1`, `G2`, and `G3`:

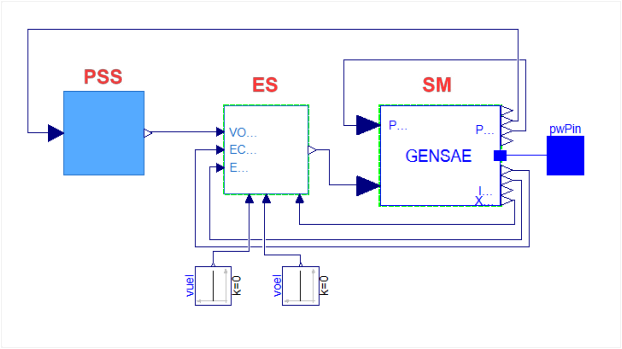


Figure 2. `GenerationUnits.MachineEXPSS.Generator1` Generator1 model diagram view

  - `MachineOnly`: Consists of only the synchronous machine (SM).
  - `MachineEXPSS`: Consists of a synchronous machine equipped with an excitation control system (ES) and power system stabilizer (PSS).
  - `MachineEXPSSIO`: Consists of a synchronous machine equipped with an excitation control system (ES) and power system stabilizer (PSS) along with an input and output (IO) interface.

The three generation units are chosen from `MachineEXPSS` to be used in the multi-machine model shown in Figure 4. Each unit is modeled as a separate component. The structure of each generation unit consists of a synchronous machine (SM), which is the primary component for generating electrical power, an excitation control system (ES) which regulates the field voltage, maintaining the terminal voltage stability, and a power system stabilizer (PSS) which provides damping of the power system oscillations by modulating the ES. The diagram view of one of the generation units `G1` is shown in Figure 2. The graphical placement and connections of the components ensure that the mathematical relationships are correctly established when `connect(...)` statements are generated. In Modelica, a `connect(...;...)` statement links the compatible ports of two components, enabling them to interact within the simulation environment as described in Chapter 9 (Modelica Association 2023).

The SM and ES are parameterized using the values of an implementation made in the Siemens PSS/E software. `*.raw` and `*.dyr` files from (Illinois Center for a Smarter Electric Grid (ICSEG) 2024) are used to set parameter values and to obtain a power flow solution that populates Modelica `records` within the `PF_Data` subpackage in Figure 1. These help provide an initial guess for the algebraic variables that are used to initialize the model (see more details in (Dorado-Rojas et al. 2021). PSS models are specifically parameterized according to optimized transfer functions from studies on PSS tuning for phase compensation(F. J. De Marco, Martins, and Ferraz 2012; F. De Marco, Rullo, and Martins 2021). Furthermore, the parameters of the individual components that are required

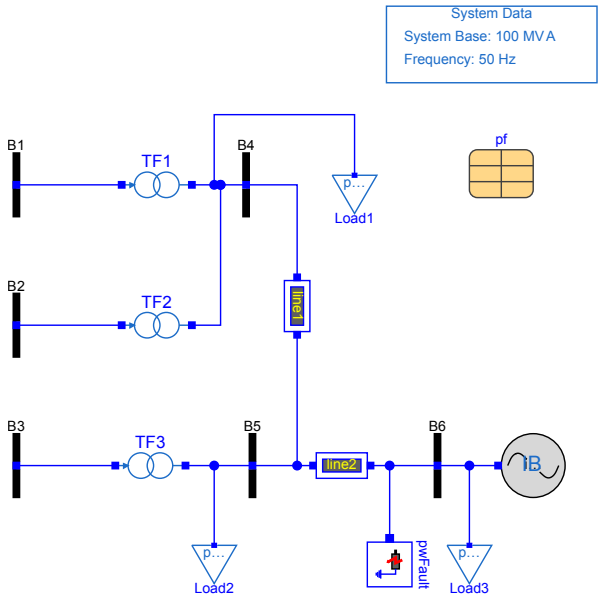


Figure 3. `ThreeMIB.Networks.BasePFnFault` multi-machine Power System Base Model

at higher levels, such as the model shown in Figure 4, are propagated to provide a user-friendly interaction.

- *Networks*: The sub-package named `Networks` is comprised of a `partial model` of the multi-machine system `ThreeMIB.Networks.Base` including the buses `B1`-`B6`, transmission lines `line1` and `line2`, the transformers `TF1`-`TF3`, and the `System Data` component. Instead of programmatically building it, this model is built graphically. The components are dragged and dropped, connected, and parameterized; the Modelica tool automatically generates the corresponding source code as shown in Listing 2.

Listing 2. Connect equations of the `partial model` called `ThreeMIB.Networks.Base`

```
1 partial model Base "Partial model
    containing network elements"
2   ...
3   OpenIPSL.Electrical.Branches.PwLine line1
      (R=0, X=0.036, G=0, B=0);
4   ...
5 equation
6   connect(B1.p, TF1.p);
7   connect(TF1.n, B4.p);
8   connect(B2.p, TF2.p);
9   connect(line1.n, line2.p);
10  ... (more connect equations follow)
11 end Base;
```

As observed in Listing 2, the instantiation of the `PwLine` component named as `line1` and the parameter `X=0.036` is set through a modifier, that is, it is changed from the default values. Similar code is generated for all other component instantiation and parameterized. Observe that there are fewer components in Figure 3 than those shown in Figure 4. This is because the model in 4 is built through inheritance, i.e., it inherits the components in Figure 3 and adds new ones. This method allows for the creation of varied system model variants from partial models, which are extended and customized through modifications. The automatically generated `connect` equations link the `PwPin` within each component instance. For example, as observed from Line 6 of Listing 2, `B1.p` is connected to `TF1.p`, thereby interfacing bus `B1` to transformer `TF1`. Similarly, line 9 shows how `line1` and `line 2` are interfaced through the `connect` equations. For illustration, this is labeled in red in Figure 4. Similar equations are automatically generated by the tool for other connections that were done graphically.

The partial model `ThreeMIB.Networks.Base` is extended by adding other components, namely the power flow component `pf`, the loads `Load1`-`Load3`, and the fault component `pwFault`. This does not include the generation units, which are discussed later. The resulting base model is called `ThreeMIB.Networks.BasePFnFault` shown in Figure 3.

- *Systems*: The sub-package `Systems` comprises the final model of the multi-machine power system, as shown in Figure 4. To create this, the `ThreeMIB.Networks.BasePFnFault` enclosed in the dotted blue box is extended and the three generation units `G1`, `G2`, and `G3` enclosed in the dotted green box are dragged and dropped from the `ThreeMIB.GenerationUnits.MachineEXPSS` package. Once connected to the corresponding buses, the generation units are parameterized with power flow data contained within the `pf` record component. The resulting model is the `ThreeMIB.Systems.Grid`, which can be readily used for typical power system time-domain simulations. This particular package also consists of the models built for linearization and nonlinear simulation which is discussed in detail in the following sections.

3 Deriving Linear Models

3.1 Refactoring Models for Linearization

This section discusses the creation of the linearized model, here referred to as "plant". To generate a model that can be linearized, the base model `ThreeMIB.Networks.Base` is extended and instantiated as `ThreeMIB.Systems.GridIO`, and the power flow component `pf` and fault component `pwFault` are added graphically. It is worth noting here that the load components added to this model `Load1`-`Load3` are chosen as those with an external input. Figure 5 shows the detailed extended model with the inputs in the green boxes and the outputs within the orange one. This is achieved by choosing the generation units from the package `ThreeMIB.GenerationUnits.MachineEXPSSIO` with an IO interface depicting a structure as shown in Figure 6. The inputs, enclosed within the dotted green boundary in Figure 6, are simply

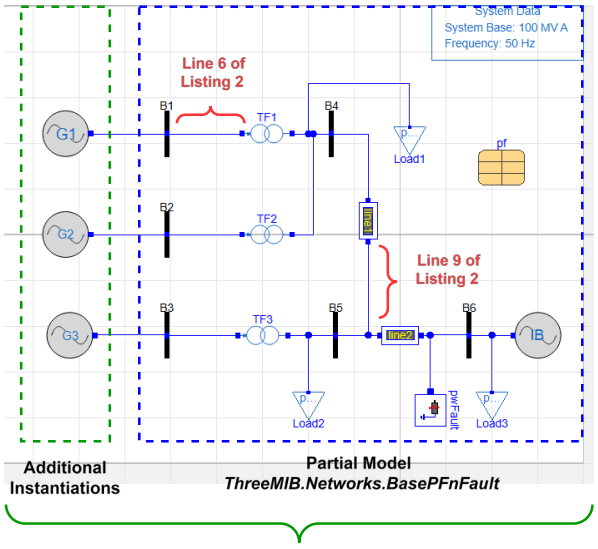


Figure 4. `ThreeMIB.Systems.Grid` multi-machine power system model

connected to a `RealInput` interface from the Modelica Standard Library (MSL). In the case of the outputs, `RealOutput` interfaces from the MSL need to be provided, which are shown enclosed by a dotted orange boundary in Figure 6. These interfaces must be linked to the desired output variables. This is carried out in the textual layer of the model, as shown in Listing 3.

Listing 3. Linking output variables to the `RealOutput` interfaces.

```
1  model GridIO
2    "Multimachine power grid model with input
         /output interfaces ..."
3    extends ThreeMIB.Interfaces.
         OutputsInterface;
4    extends ThreeMIB.Networks.Base(...
5    GenerationUnits.MachineEXPSSIO.
         Generator1EXPSSIO G1(...)
6    ...  // More instantiations follow
7  equation
8    SCRXin = G1.feedbackSCRX.y;
9    SCRXout = G1.sCRX.EFD;
10   Vt = G1.gENSAE.ETERM;
11   ANGLE = G1.gENSAE.ANGLE;
12   SPEED = G1.gENSAE.SPEED;
13   ... // More connect statements follow
14 end
```

Each of the `RealOutput` interfaces must be linked to the output of different components. For example, on Line 10 of Listing 3, the generator's terminal voltage `G1.gENSAE.ETERM` is linked to the interface `Vt`. This is done similarly for other machine variables. Meanwhile, to access the output of the PSS (which is the input of the ES), the `RealOutput` interface `SCRXin` is linked to `G1.feedbackSCRX.y` as seen in Line 8 and similarly the output of the ES, `SCRXout`, is linked to the field voltage `G1.sCRX.EFD` in Line 9. The plant model shown in Figure 5 can now be utilized as a block with the specified

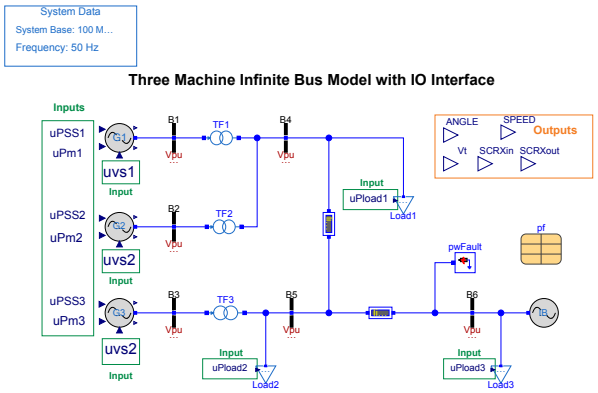


Figure 5. `ThreeMIB.Systems.GridIO` multi-machine power system model with IO interface

inputs and outputs for the analysis of the design of the control system. Figure 7 illustrates this concept where the inputs to the `ThreeMIB.Systems.GridIO` can be set to zero with only one desired functional input and output. The entire model enclosed in the red dotted lines is treated as a single-input-single-output (SISO) block. This modularity improves the adaptability and utility of the model in diverse linearization and simulation needs.

3.2 Linearization Process

Each Modelica-compliant tool, such as Dymola or OpenModelica, supports symbolic analysis to automatically generate a linear model from the same model used for non-linear time-domain simulation. Within Dymola, the `Modelica_LinearSystems2` (`MLin2`) library can be used to perform this task, which allows easy conversion of models to representations of linear time-invariant systems (Baur, Otter, and Thiele 2009). Listing 4 shows the command needed to linearize the model shown in Figure 5. A state space object and `*.mat` file are generated as the resulting output `ss` which is suitable for further analysis in Dymola or external tools, supporting tasks such as eigenvalue computation, frequency response analysis, and advanced control design such as pole placement and LQG controller design.

Listing 4. Linearization using Modelica_LinearSystems2

```
1 ss := Modelica_LinearSystems2.ModelAnalysis
      .Linearize("ThreeMIB.Systems.GridIO");
```

Once linearized, the system, input, and output matrices can be observed from Dymola's command window.

4 Nonlinear Simulation

This section explores the initialization process and the selection of solvers in time-domain simulations, demonstrating how these features can accommodate various use cases with models developed using OpenIPSL.

4.1 Initialization

Providing suitable initial guess values for large-system models under various operating conditions can be challenging. To address this, a Modelica `record` template within

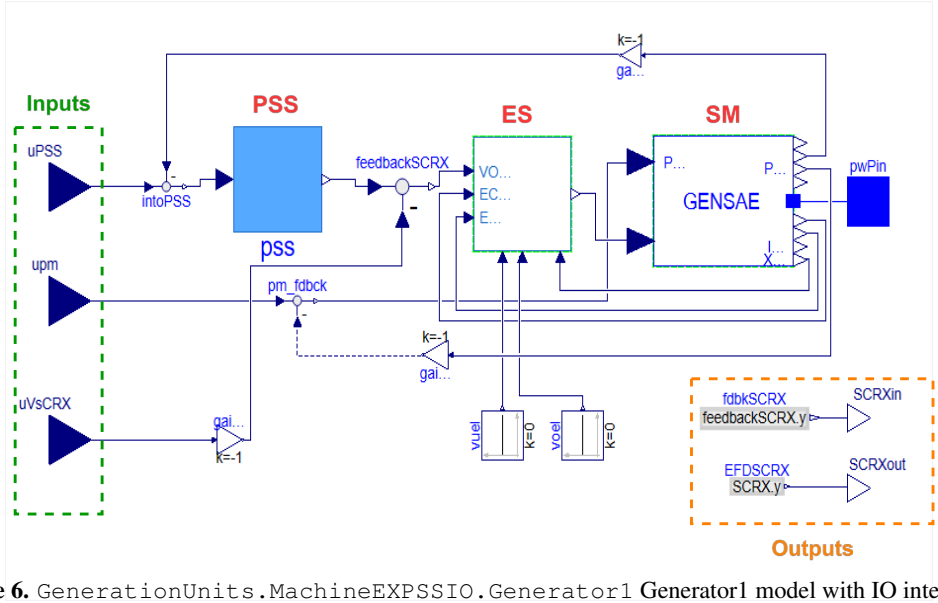


Figure 6. `GenerationUnits.MachineEXPSSIO.Generator1` Generator1 model with IO interfaces.

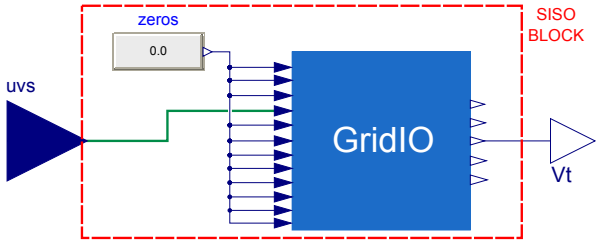


Figure 7. `ThreeMIB.Systems.GridIOsiso` multi-machine power system with IO interface used as a SISO Block

`ThreeMIB.PF_Data.powerflow` is associated with each component of the model to facilitate the entry of data from the power flow solution as starting values. This mapping is done once when creating the model. As shown in Figure 5, the component `pf` is added directly as the block of the yellow record template in the diagram. This allows for the selection of specific data values for buses, machines, loads, and transformers. This record structure can be automatically implemented using the `pf2rec` Python utility, which transforms the power flow simulation results into Modelica records (Dorado-Rojas et al. 2021).

Similarly, as mentioned in Section 2 the `OpenIPSL.Electrical.Essentials.pfComponent` can be provided with data that are used to calculate the starting values within each of the components that extend from the `pfComponent`. For example, it can be observed in Listing 1, how the start values for the real and imaginary parts of the voltage phasor, `vr` and `vi` are calculated from data of voltage magnitude and angle, `v_0` and `angle_0` (see Line 8).

4.2 Solvers

Domain-specific power system tools like Siemens PSS/E usually provide a single solver for which the models' equations have been discretized; a popular choice is to use the trapezoidal integration method combined with a Netwon-Raphson solver to solve the DAEs. This approach typically restricts simulations to a few seconds with a fixed time step. Modelica tools do not face this limitation when simulating the models from the OpenIPSL library. As noted in (Henningsson, Olsson, and Luigi Vanfretti 2019), Dymola has advanced solvers for sparse large-scale DAE models, enhancing the competitiveness of power system simulations with Modelica compared to Siemens PSS/E. To utilize these advanced features in Dymola, the utility `ThreeMIB.Utilities.SetupSolverSettings` offers a series of functions to enable or disable them. For example, it allows settings like `Advanced.Define.DAEsolver := true/false` and `Advanced.SparseActivate := true/false`, which activates the DAE solvers and optimizes for sparsity, respectively. Note that for linearization tasks, these advanced settings should be deactivated to ensure the generation of accurate state-space models.

5 Results

The power system models developed in this work are utilized to analyze the control system design developed in the study for PSS tuning using phase compensation (F. J. De Marco, Martins, and Ferraz 2012). Modelica tools provide means to visualize and analyze the results. Custom functions can be used with the necessary path to the models to perform the required analysis. Within Dymola, the `Modelica_LinearSystems2` (`MLin2`) library provides commands to directly linearize and plot the frequency response from the single-input-single-output version of the model in Figure 7 as observed in Listing 5.

Listing 5. Custom Function for Bode Plot using Modelica_LinearSystems2

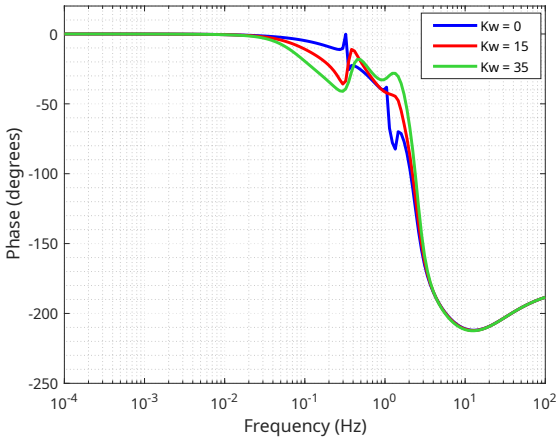
```
1 function bodeplot_GridIOsiso
2   extends Modelica.Icons.Function;
```

```
  input Modelica.Units.SI.Time tlin = 30;
algorithm
  ...
  // linearize and plot
  Modelica_LinearSystems2.ModelAnalysis.
      TransferFunctions(
  "OpenIPSL.ThreeMIB.Systems.GridIOsiso",
      simulationSetup=
  Modelica_LinearSystems2.Records.
      SimulationOptionsForLinearization(
  linearizeAtInitial=false,
  t_linearize=tlin));
end bodeplot_GridIOsiso;
```

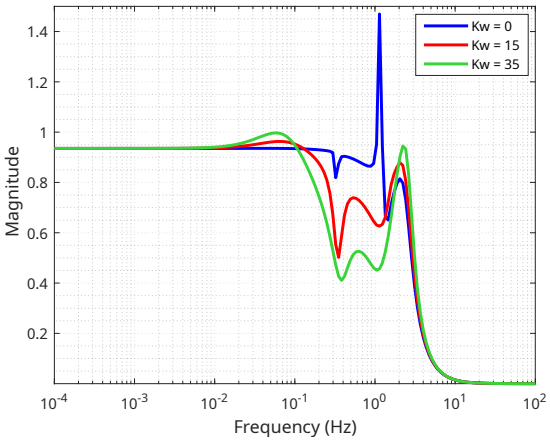
Figure 8 illustrates the frequency response of the system, showing both the magnitude and phase of the terminal voltage as functions of frequency for increasing values of the PSS gain (K_w). These values were obtained from the designs in (F. J. De Marco, Martins, and Ferraz 2012). The adjustment K_w adjusts the phase change introduced by the system as shown in Figure 8a. When the PSS is disabled by setting $K_w = 0$, the phase curve introduces a negative phase shift in the frequency response. Increasing K_w to 15 and 35 shows an improvement in phase around the resonant frequency, reducing the phase lag, which is crucial to effectively damp system oscillations. The magnitude plot as shown in Figure 8b reveals the system's sensitivity to frequency changes for different PSS gains. With increasing K_w, there are noticeable peaks in the magnitude response, particularly around the resonant frequencies, suggesting an enhanced ability of the PSS to counteract perturbations effectively. However, higher gains ($K_w = 35$) introduce sharp peaks that could lead to potential system instability under certain conditions.

The PSS is tuned by receiving a feedback signal from the rotor speed of the synchronous machine. Figure 9 demonstrates the time-domain simulation of the rotor speed of Generator 1 after a load disturbance at t = 30.5 seconds, clearly demonstrating the impact of varying PSS gain values on the stability of the system. With the PSS disabled ($K_w = 0$), the rotor speed experiences substantial oscillations, indicating poor damping characteristics. With an increase of K_w to 15 and 35 there is an improvement in damping performance, with the rotor speed quickly stabilizing and exhibiting minimal oscillatory behavior. This analysis underscores the effectiveness of PSS in enhancing the system's dynamic response to disturbances, highlighting the critical role of appropriate PSS tuning in maintaining system stability.

To further investigate this power system dynamics, Figure 10 provides further insight into the stability of the system by illustrating the pole positions of the `GridIOsiso` model under varying PSS gains. With the PSS disabled ($K_w = 0$), the poles marked with pink crosses highlight a critically damped system with potential for sustained oscillations. Increasing K_w to 15 (red) and 35 (dark red) shows a shift in the poles, which move toward the left in the complex plane. This indicates improved damping and stability, thus emphasizing the significant influence of PSS



(a) Phase for different values of K_w



(b) Magnitude for different values of K_w

Figure 8. Bode Plot of the GridIOsiso model for three values of the PSS gain (K_w)

tuning on the system dynamics. Moreover, this illustrates the value of complementing non-linear simulations with linear methods when assessing control system designs.

6 Conclusions

In this work, Modelica and the OpenIPSL library have been utilized to build a multi-machine power system model developed for the analysis of intra-plant and inter-area modes. The model is refactored and extended to implement a control system design and analyze its performance. This is done by exploiting the object-oriented modeling features of Modelica. Linearization capabilities provide an advantage over other domain-specific tools in implementing this design and performing an analysis of the model. Given the complexity of power systems and the critical role of stability and dynamic behavior as illustrated in Figure 10, careful control design analysis is essential to ensure the robustness of the system to dynamic conditions and disturbances.

This study demonstrates how Modelica and OpenIPSL

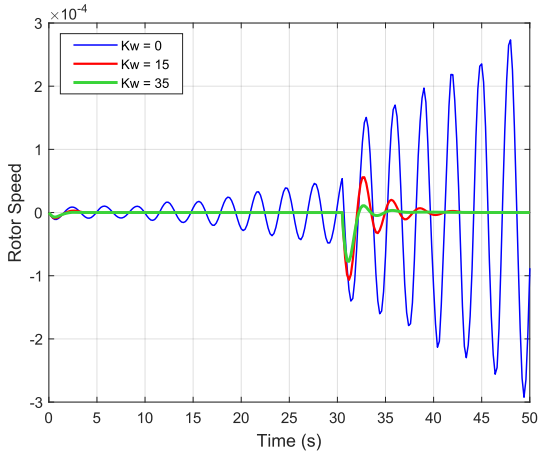


Figure 9. Time domain simulation of the rotor speed of G1 under a load disturbance at t= 30.5 sec. for different values of PSS gain (K_w)

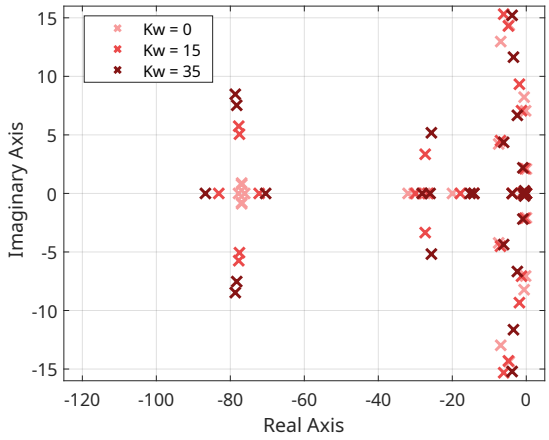


Figure 10. Poles of the GridIO model

can be used to assist in control design analysis and improve power system dynamic performance by testing control system designs. The modifications (model re-factoring and extension) aim to effectively address the complexities of power system stability studies effectively. Additional work includes the development of detailed examples of the actual design of the PSS using the unique features of Modelica and the integration of the models presented into the OpenIPSL library.

To access the models in this paper before they are integrated into OpenIPSL, the reader can find them in the following GitHub repository: `https://github.com/ALSETLab/AMCONF2024_ThreeMIB`

Acknowledgements

This paper is in part, based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139.

References

al., Peter Fritzson et (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.

Baudette, Maxime et al. (2018). "OpenIPSL: Open-instance power system library—update 1.5 to "iTesla power systems library (iPSL): A modelica library for phasor time-domain simulations"". In: *SoftwareX* 7, pp. 34–36.

Baur, Marcus, Martin Otter, and Bernhard Thiele (2009). "Modelica libraries for linear control systems". In: *Proceedings 7th Modelica Conference*. DOI: 1, pp. 593–602.

Brück, Dag et al. (2002). "Dymola for multi-engineering modeling and simulation". In: *Proceedings of modelica*. Vol. 2002. Citeseer.

Chow, J.H. and K.W. Cheung (1992). "A toolbox for power system dynamics and control engineering education and research". In: *IEEE Transactions on Power Systems* 7.4, pp. 1559–1564. DOI: 10.1109/59.207380.

De Castro, Marcelo et al. (2023). "Version [OpenIPSL 2.0.0]-[iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations]". In: *SoftwareX* 21, p. 101277.

De Marco, Fernando, Pablo Rullo, and Nelson Martins (2021). "Synthetic power system models for PSS tuning and performance assessment". In: *2021 IEEE Electrical Power and Energy Conference (EPEC)*. IEEE, pp. 107–112.

De Marco, Fernando Javier, Nelson Martins, and Julio Cesar Rezende Ferraz (2012). "An automatic method for power system stabilizers phase compensation design". In: *IEEE Transactions on power systems* 28.2, pp. 997–1007.

Dorado-Rojas, Sergio A et al. (2021). "Power flow record structures to initialize openipsl phasor time-domain simulations with python". In: *Modelica Conferences*, pp. 147–154.

Fachini, Fernando, Srijita Bhattacharjee, et al. (2023). "Exploiting Modelica and the OpenIPSL for University Campus Microgrid Model Development". In: *Modelica Conferences*, pp. 285–292.

Fachini, Fernando, Luigi Vanfretti, et al. (2021). "Modeling and validation of renewable energy sources in the openipsl modelica library". In: *IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, pp. 1–6.

Fritzson, Peter (2014). *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons.

Fritzson, Peter and Vadim Engelson (1998). "Modelica—A unified object-oriented language for system modeling and simulation". In: *ECOOP'98—Object-Oriented Programming: 12th European Conference Brussels, Belgium, July 20–24, 1998 Proceedings 12*. Springer, pp. 67–90.

Guironnet, Adrien et al. (2018). "Towards an Open-Source Solution using Modelica for Time-Domain Simulation of Power Systems". In: *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pp. 1–6. DOI: 10.1109/ISGTEurope.2018.8571872.

Henningsson, Erik, Hans Olsson, and Luigi Vanfretti (2019). "DAE Solvers for Large-Scale Hybrid Models." In: *Modelica*, pp. 157–050.

Illinois Center for a Smarter Electric Grid (ICSEG) (2024). *Three Machines Infinite Bus Benchmark System*. Available online: https://icseg.iti.illinois.edu/three-machines-infinite-bus-benchmark-system/.

Isaacs, Andrew (2017). "Simulation Technology: The Evolution of the Power System Network [History]". In: *IEEE Power and Energy Magazine* 15.4, pp. 88–102. DOI: 10.1109/MPE.2017.2690527.

Li, Wei, Luigi Vanfretti, and Yuwa Chompoobutrgool (2012). "Development and implementation of hydro turbine and governor models in a free and open source software package". In: *Simulation Modelling Practice and Theory* 24, pp. 84–102.

Martins, Nelson et al. (2000). "A small-signal stability program incorporating advanced graphical user interface". In: *Proceedings of the VII SEPOPE*.

Milano, Federico (2005). "An open source power system analysis toolbox". In: *IEEE Transactions on Power systems* 20.3, pp. 1199–1206.

Milano, Federico (2013). "A Python-based software tool for power system analysis". In: *2013 IEEE Power & Energy Society General Meeting*. IEEE, pp. 1–5.

Milano, Federico and Luigi Vanfretti (2009). "State of the art and future of OSS for power systems". In: *2009 IEEE Power & Energy Society General Meeting*. IEEE, pp. 1–7.

Modelica Association (2023). *Modelica Language Specification v3.6.0*. Available online: https://specification.modelica.org/maint/3.6/MLS.html. Accessed: 14 Aug 2024.

Nikolaev, Nikolay et al. (2020). "PSS/E Based Power System Stabilizer Tuning Tool". In: *2020 21st International Symposium on Electrical Apparatus & Technologies (SIELA)*. IEEE, pp. 1–6.

Plietzsch, Anton et al. (2022). "PowerDynamics.jl–An experimentally validated open-source package for the dynamical analysis of power grids". In: *SoftwareX* 17, p. 100861.

Vanfretti, L. et al. (2016). "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations". In: *SoftwareX* 5, pp. 84–88. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2016.05.001. URL: https://www.sciencedirect.com/science/article/pii/S2352711016300097.

Vanfretti, Luigi et al. (2013). "Unambiguous power system dynamic modeling and simulation using modelica tools". In: *2013 IEEE Power & Energy Society General Meeting*. IEEE, pp. 1–5.

Winkler, Dietmar (2017). "Electrical power system modelling in modelica–comparing open-source library options". In.

Integrating the IEEE/CIGRE DLL Modeling Standard to Use "Real Code" Models for Power System Analysis in Modelica Tools

Hao Chang[1] Luigi Vanfretti[1]

[1]Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY, USA,
{changh7, vanfrl}@rpi.edu

Abstract

Vendors of power system simulation tools are investigating the incorporation of actual controller code into specialized simulation environments. To facilitate this, IEEE and CIGRE have collaboratively created the IEEE/CIGRE DLL Modeling Standard. However, adoption by simulation tool providers has been minimal. The limited adoption is because 'real code' models per the IEEE/CIGRE DLL Modeling Standard must be provided as DLLs by equipment vendors. Thus, to support the standard, tools need to support a standard-specific interface and provide additional functions to execute the models.

This paper presents a method for integrating 'real controller code' models (RCMs) built according to the IEEE/CIGRE DLL Modeling Standard into Modelica-based tools. This is achieved by linking precompiled C code to Modelica models and using components from the OpenIPSL library. The approach is demonstrated with an RCM of a simplified silicon-controlled rectifier excitation system (SCRX). The paper discusses the details of the implementation, challenges, and solutions. The findings show that this method allows RCMs to be used in Modelica tools for power system simulations, providing a valuable alternative to specialized simulation tools.

Keywords: IEEE/CIGRE DLL Modeling Standard, Generator excitation, Power Systems, Power System Simulation, External Object

1 Introduction

1.1 Motivation

In modern control systems engineering, the ability to test and validate control strategies under diverse and realistic conditions is paramount. Traditional controller testing methods often fail to replicate real-world scenarios, leading to discrepancies between the simulated and actual performance of the system under test. To bridge this gap, the integration of controller code into simulation environments has emerged as a crucial step, often referred to as "Software-In-the-Loop" (SIL) simulation (Schaub, Hellerer, and Bodenmüller 2012). By incorporating the controller code into SIL, the number of discrepancies between simulation results and field measurements can be reduced, improving the accuracy and reliability of simulation models (Ramasubramanian et al. 2024). However, in the field of power system simulation, this remains a challenging situation for multiple reasons. One of the difficulties faced is that of exchanging models between electromagnetic transient (EMT) simulation platforms and/or dynamic simulation tools (transient stability or phasor simulators). To a large extent, this is mainly due to the lack of a standardized equation-based modeling language for model exchange, leading to inconsistencies in simulation results between different tools. This inconsistency can result in speculation about the accuracy of the model or the adequacy of a simulation tool, highlighting the need for a more consistent model exchange mechanism (Rogersten, Vanfretti, and Li 2015).

Power system simulation tool vendors and users have started to explore the integration of 'real controller code' models (RCMs) into domain-specific simulation environments. They have established a joint effort within two professional organizations (CIGRE and IEEE [1]) to develop a domain-specific approach to perform such integration, known as the IEEE/CIGRE DLL Modeling Standard (ICDMS). Unfortunately, the proposed approach has only been adopted by a few power system simulation tool vendors, limiting the use of such RCMs to those tools. This adoption has been limited because the RCMs, according to the IEEE/CIGRE DLL Modeling Standard (ICDMS), are to be provided as DLLs (Dynamic Link Libraries) by equipment vendors. Hence, to support this standard within a simulation environment, a standard-specific interface needs to be called, and to run the models additional ancillary functions need to be developed.

To expand the potential use of such models beyond domain-specific power system tools and leverage the built-in features of the Modelica language for integrating external objects, this paper presents a novel method for incorporating precompiled C code to support the ICDMS

[1]According to https://www.electranix.com/ieee-pes-tass-realcodewg/ this is under the IEEE Task Force "Use of Real-Code in EMT Models for Power System Analysis" and according to https://tinyurl.com/ieee-cigre-dll-tor this is a Joint Task Force under CIGRE Study Committe B4, with Title: "Guidelines for Use of Real-Code in EMT Models for HVDC, FACTS and Inverter based generators in Power Systems Analysis".
CIGRE is the International Council on Large Electric Systems, which is a professional global non-profit in the field of high voltage.
The Institute of Electrical and Electronics Engineers (IEEE) is a professional association for electronics engineering, electrical engineering, and other related disciplines

domain-specific standard within the Modelica language and the OpenIPSL library.

1.2 Related Works

The modeling and simulation community has successfully developed interoperable standards, such as the Functional Mock-up Interface (FMI) (Junghanns et al. 2021) and the Functional Mock-up Interface for Embedded Systems (eFMI) (Lenord et al. 2021), which aim to streamline model exchange and integration in simulation environments. However, there are still significant challenges to achieve widespread adoption, especially in engineering areas where domain-specific approaches are the rule, which is the case for the electrical power industry (Vanfretti, Li, et al. 2013).

Meanwhile, within the power industry itself, previous efforts to standardize equipment models have not been successful due to their lack of adoption. One particular example is that of the generic software interface developed as part of the IEC 61400-27-1:2020 standard "Wind energy generation systems - Part 27-1: Electrical simulation models - Generic models" (see `https://webstore.iec.ch/publication/32564`), which intended to provide both generic models and an interface method for vendor-specific wind turbine models.

These grid standards have been unsuccessful as equipment manufacturers have been slow to adopt them and provide equipment models according to the standards, resulting in persistent difficulties in model exchange. Manufacturers are discouraged in adopting any of these standards due to the customers' preference for tool-specific implementations (e.g., PSCAD and PSS/E), which leads to tool lock-in. Although there have been efforts in Europe to develop the Common Grid Model Exchange Specification (see `https://tinyurl.com/cgmes2p5`); simulation tools built and used outside Europe have not yet adopted this standard. What this implies for user's that need functionalities not yet supported by domain-specific simulation tools, or that want to use Modelica-complaint simulation environments, is that the domain-specific approach has to be somehow supported within the Modelica ecosystem. This is what is attempted in this paper for the case of the IEEE/CIGRE DLL Modeling Standard (ICDMS).

In addition to implementing the ICDMS, means to simulate the reminder of the power grid in Modelica tools are required. Fortunately, an effort to port the behavioral model descriptions in Modelica replicating those of the PSS/E software (the simulation tool most used in the US and the Nordic countries) has been in place for almost a decade (T. Bogodorova et al. 2013; Vanfretti, Tetiana Bogodorova, and Baudette 2014; Zhang et al. 2015), which makes it possible to reproduce power system dynamic simulation results like those expected by industry practitioners. The OpenIPSL(de Castro et al. 2023) is a Modelica library that provides robust models and enhanced portability aimed at building an open-source software-based encyclopedia of dynamic power system models that can be exploited by multiple modeling tools that are compliant with the Modelica language specification. The OpenIPSL is used here to set up power grid simulation models in which the RCMs are included.

1.3 Contributions

This paper presents a method for integrating RCMs built according to the IEEE/CIGRE DLL Modeling Standard (ICDMS) into Modelica-based tools. This is achieved by linking precompiled C code to Modelica models and using components from the OpenIPSL library. The approach is demonstrated with an RCM of a simplified silicon-controlled rectifier excitation system (SCRX).

Our demonstration involves modifying and compiling the code of the SCRX RCM into Dynamic Link Libraries (DLLs), following the ICDMS. This standardization ensures compatibility with domain-specific standards and facilitates the seamless incorporation of controller code into Modelica simulations. The primary contribution of this paper is the detailed description of the process used to integrate the precompiled DLLs into the simulation environment, enabling extensive testing and validation of the controller code.

2 Background on the IEEE/CIGRE DLL Modeling Standard

To explain how the ICDMS functions, the simulation workflow shown in Figure 1 is used. It starts with "Allocate Memory", where memory for inputs, outputs, and parameters is allocated. Model Initialization then sets initial conditions and parameters. The Update Input step reads the current input values, followed by Run Calculation, where the model computes the output based on input values and parameters. Finally, Update Output writes the results to the output variables, completing one simulation cycle. This workflow repeats, allowing dynamic simulation of the controller's behavior.

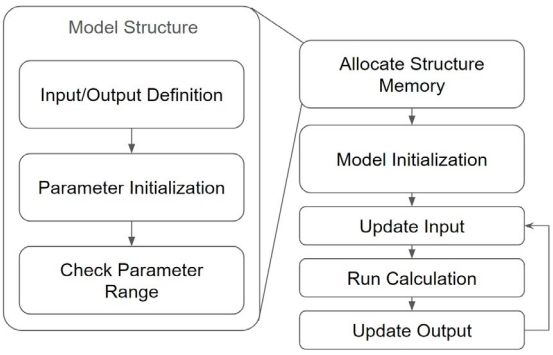


Figure 1. Model Structure and Simulation Workflow according to the IEEE/CIGRE DLL Modeling Standard

The implementation of the workflow in Figure 1 follows the ICDMS by defining a clear structure for input signals, output signals, and parameters using standardized data types and units. Standardized functions

such as *Model_GetInfo*, *Model_CheckParameters*, and *Model_Initialize* ensure proper initialization and parameter validation. The *Model_Outputs* function performs the main computational tasks, adhering to the fixed time step approach common in the real-world controller firmware. Each model to be developed using this approach needs to be compiled into a DLL, enabling its use across various simulation tools. However, this requires that the ICDMS be supported by the simulation tool.

3 Implementing the IEEE/CIGRE DLL Modeling Standard in Modelica

In the following section, the SCRX RCM is used as an example to illustrate the implementation of the interface between the C code and the Modelica language. The same methodology can be applied to other controller codes following the same ICDMS. It should be noted that the SCRX RCM is one of the examples used in the development of the ICDMS.

This example demonstrates the capability to interface a DDL of an RCM with a Modelica library and simulation tool. In most cases, controller manufacturers will not disclose their controller structure and may only provide parameter values, which would require a clean-room re-implementation similar to those in (Laera et al. 2022) to be used in a Modelica tool. However, if they follow the approach proposed in this paper, RCMs provided by manufacturers could be used to run simulations without the need of a complete re-implementation in Modelica.

3.1 External Object Integration

In ICDMS, structures store all the information about a controller including the simulation time step, the number of input/output, the parameter values and other information. To access a structure defined in `C`, we have to define a class in Modelica as shown in Listing 1. A `constructor` and `destructor` must be specified in a class to initialize and de-initialize an object from a class. This is essential for the computer to allocate and free the memory that stores the data of the structure.

Listing 1. SCRX Class Real-code Modelica Implementation.

```
class SCRX9_DLL
  extends ExternalObject;
  function constructor
    output SCRX9_DLL scrx9_dll;
    external "C" scrx9_dll = init_scrx_model()
        annotation (Library="SCRX9",
        LibraryDirectory="modelica://OpenIPSL/
        Resources/Library");
  end constructor;
  function destructor
    input SCRX9_DLL scrx9_dll;
    external "C" deinit_scrx_model(scrx9_dll)
        annotation (Library="SCRX9",
        LibraryDirectory="modelica://OpenIPSL/
        Resources/Library");
  end destructor;
end SCRX9_DLL;
```

The external C function `init_scrx_model` is called at line 5 of Listing 1 to allocate memory space. In `annotation`, the library name and directory have to be specified for the compiler to know where to look for the required functions.

Listing 2. `init_scrx_model` Function Implementation.

```
__declspec(dllexport) void* __cdecl
    init_scrx_model(void)
{
    IEEE_Cigre_DLLInterface_Instance* instance =
         (IEEE_Cigre_DLLInterface_Instance*)
        malloc(sizeof(
        IEEE_Cigre_DLLInterface_Instance));
    ...
    /*PARAMETER INITIALIZATION*/
    ...
    double * states = malloc(6 * sizeof (double)
        );
    instance->DoubleStates = states;
    Model_Initialize(instance);

    return (void *) instance;
}
```

The C functions shown in Listing 2 initialize all the parameters (Line 5) of the instance and allocate memory space (Line 7) to save key state values, when the constructor is called. Since most controller consists of integrators that require memory, line 7 allocates memory space to store the states of the integrators. Line 11 returns the address of the instance to access this initialized instance later in Modelica functions. From line 4 of Listing 1, the returned address is returned again by the constructor as an external object of class `SCRX9_DLL`.

Having initialized and allocated memory, the model needs to be accessed and integrated to a power system model. As an excitation control system, the example model features two primary inputs: ETERM, representing the generator's terminal voltage, and XADIFD, representing the field current, both initialized to steady-state values to avoid initialization problems. The EFD output is the generated field voltage. The object `scrx9_struct` wraps the states and parameters of the SCRX controller initialized in Line 3 of the Listing 2. The algorithm section updates the controller's state from the input port using the `update` function shown in Listing 4. The resulting field voltage is obtained through `model_output` function defined in Listing 5. In addition, the function `update_scrx_input` shown in Listing 4 reads the values from the input ports in Modelica and updates them in the defined C instance.

Listing 3. SCRX Controller Modelica Model.

```
model SCRX
  Modelica.Blocks.Interfaces.RealInput ETERM(
      start = 1);
  Modelica.Blocks.Interfaces.RealInput XADIFD(
      start = 1.325);
  Modelica.Blocks.Interfaces.RealOutput EFD;
```

 10.3384/ECP20772 DOI

```
  SCRX9_DLL scrx9_struct = SCRX9_DLL();
algorithm
  Functions.update(scrx9_struct,time,1,ETERM,0,
      XADIFD,ETERM,0,0);
  EFD:=Functions.model_output(scrx9_struct);
  when terminal() then
    Functions.save_ss_state(scrx9_struct);
  end when;
end SCRX;
```

Listing 4. `update` Function Implementation.

```
__declspec(dllexport) void __cdecl
    update_scrx_input(
    IEEE_Cigre_DLLInterface_Instance* instance,
    double sim_time_input, double vref,double ec
    ,
    double vs,double ifd,double vt,double vuel,
        double voel) {
    MyModelInputs* inputs = (MyModelInputs*)
        instance->ExternalInputs;
    inputs->IFD = ifd; // Field current
    inputs->VT = vt; // Terminal voltage
    ... // Other inputs
    sim_time = sim_time_input;
};
```

Listing 5. `model_output` Function Implementation.

```
__declspec(dllexport) double __cdecl
    model_calculate(
    IEEE_Cigre_DLLInterface_Instance* instance)
    {
    MyModelOutputs* outputs = (MyModelOutputs*)
        instance->ExternalOutputs;
    if (sim_time != pre_sim_time)
    {
        Model_Outputs(instance);
        pre_sim_time = sim_time;
    }
    return outputs->EFD;
};
```

At each time step of the simulation, the program will call `model_calculate` shown in Listing 5 to calculate the output with `Model_Outputs`. The calculation result will return to Modelica as a floating number or a list of floating numbers depend on the type of the controller (multiple input single output or multiple input multiple output).

3.2 Initialization of the External Object

Initializing the RCM requires us to ensure that the simulation starts from a valid equilibrium point. Consequently, this requires sending data to the external object and linking its output to the rest of the system model. In the case of the SCRX RCM, this means passing the measured voltage from the bus bar to the excitation system and returning the field voltage value at the equilibrium condition.

To this end, the C function shown in Listing 6 is called at the termination of each simulation (see Line 10 of Listing 3) to extract the current values of the controller's inputs, outputs, and state variables, storing them in an array for writing to a binary file. For the SCRX controller, the inputs include signals such as `VRef` (reference voltage), `Ec` (measured voltage), `VOEL` (over excitation limit), and others. The output, `EFD`, represents the generated field voltage. Furthermore, the state variables, stored in the `DoubleStates` array (see Line 16) within the instance, are also included. The function opens the file in binary write mode, populates the array with the extracted values, and writes the entire array to the file (see Line 18). This process ensures that all critical data required by the controller are preserved, enabling the analysis and potential reinitialization of the system at the desired state in future simulations.

Listing 6. `save_ss_state` Function Implementation.

```
__declspec(dllexport) void __cdecl
    save_states(
    IEEE_Cigre_DLLInterface_Instance*
    instance)
{
    MyModelOutputs* outputs = (MyModelOutputs
        *)instance->ExternalOutputs;
    MyModelInputs* inputs = (MyModelInputs*)
        instance->ExternalInputs;
    int listSize = 7+1+6; %input+output+
        states
    double list[listSize];
    FILE* file = fopen("list.dat", "wb");
    if (file != NULL)
    {
        list[0] = inputs->VRef;
        list[1] = inputs->Ec;
        .../*More Input states*/
        list[6] = inputs->VOEL;
        list[7] = outputs->EFD;
        for (int i = 0; i < 6; i++){
            list[8+i] = instance->
                DoubleStates[i];
        }
        fwrite(list, sizeof(double), listSize
            , file);
        fclose(file);
    }
}
```

3.3 Illustration with the SCRX Excitation Model

The SCRX excitation model is a simplified control system designed to regulate the field voltage of a synchronous generator, thereby maintaining the machine's AC voltage at a specified reference set-point. This section introduces the excitation controller and illustrates its block diagram and overall system structure shown in Fig.2.

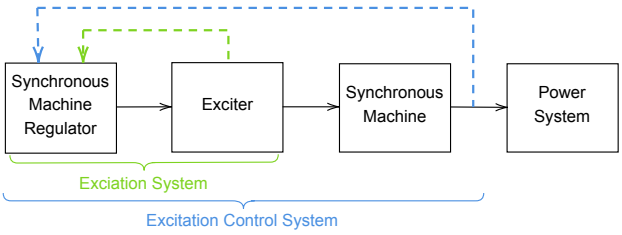


Figure 2. Synchronous Machine Control System(IEEE 2007).

The *Synchronous Machine Regulator* generates control

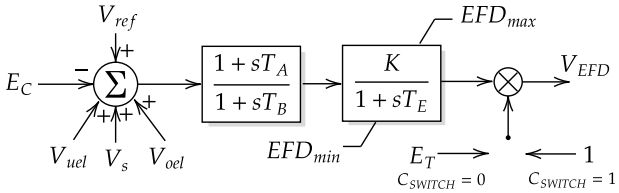


Figure 3. Excitation Control System ("IEEE Standard Definitions for Excitation Systems for Synchronous Machines" 2022).

signals based on the reference voltage (V_REF) and feedback signals from the synchronous machine. It takes these inputs to produce an appropriate control command for the exciter. The *Exciter* modulates the field voltage of the synchronous machine in response to the control commands from the regulator. It serves as an intermediary that translates the regulator's signals into field winding voltage adjustments. The *Synchronous Machine* is the plant of this system that converts rotational mechanical energy into electrical power. It responds to the field voltage adjustments made by the exciter and influences the voltage and stability of the broader power system. The *Power System* represents the electrical grid of which the synchronous machine is part. The primary goal of the excitation system is to maintain the desired voltage levels at the generator terminals.

The SCRX excitation model shown in Figure 3 presents a detailed block diagram and standardized modeling approach that is generally adopted to represent how the control of the field voltage of the synchronous generators is achieved ("IEEE Standard Definitions for Excitation Systems for Synchronous Machines" 2022). The integration of this model into power system simulations allows for extensive testing and validation, ensuring optimal performance under various operational conditions.

Table 1 lists the parameters and the default values required by the SCRX controller, including time constants (TAdTB, TB, TE), controller gain (K), and voltage limits (EMin, EMax), as well as the power source selection switch (CSwitch) and the field resistance ratio (RCdRFD). These parameters are essential for configuring the controller to operate within the desired specifications and to ensure compatibility with the ICDMS.

Table 2 lists the input signals such as the reference voltage (VRef), measured voltage (Ec), stabilizer signal (Vs), field current (IFD), terminal voltage (VT), and excitation limits (VUEL and VOEL), which are used to dynamically adjust the controller performance during simulation. Table 3 defines the output signal (EFD), representing the output machine field voltage.

These tables illustrate the format typically used to define the models of excitation control systems. Note that the ICDMS adopts this formating to specify the parameter, input, and output specifications of all RCMs. This would allow us to use the RCMs in any simulation environment adhering to the ICDMS.

Table 1. SCRX Parameters.

Parameters	*Description*	*Default*
TAdTB	Time Constant	0.1
TB	Time Constant	10
K	Controller Gain	100
TE	Time Constant	0.05
EMin	Min Field Voltage	-10
EMax	Max Field Voltage	10
CSwitch	Power Source Select	1
RCdRFD	Field resistance ratio	10

Table 2. SCRX Input Signals.

Vref	Reference voltage
Ec	Measured voltage
Vs	Stabilizer signal
IFD	Field Current
VT	Terminal Votlage
VUEL	Under Excitation Limit
VOEL	Over Excitation Limit

4 Results

4.1 Testing Power Network Model

The power network model that incorporates the SCRX model is constructed using the OpenIPSL and is shown in Fig.4. This power system model provides a platform for testing both RCM and standard OpenIPSL built-in SCRX9 example controllers.

The power network consists of a synchronous generator connected to an infinite bus through transmission lines, buses, and including a load. The generator is controlled by an SCRX excitation system, which regulates the field voltage (EFD) to maintain the desired power output. A short fault was applied between Bus2 and Bus3 starting at 2 seconds and stopping at 2.15 seconds. This network allows for comprehensive testing and validation of the SCRX controller integrated as an RCM in DLL form, following the ICDMS. By simulating a short fault (i.e., a large disturbance), the power network response can be

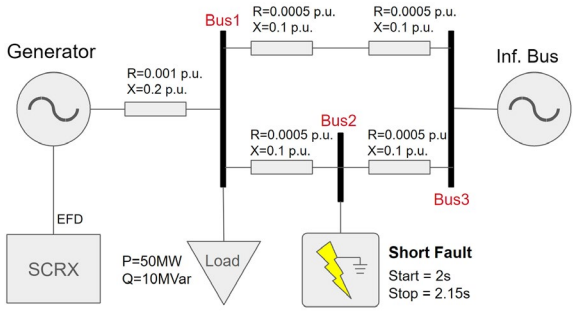


Figure 4. SCRX Controller within a Testing Power Network.

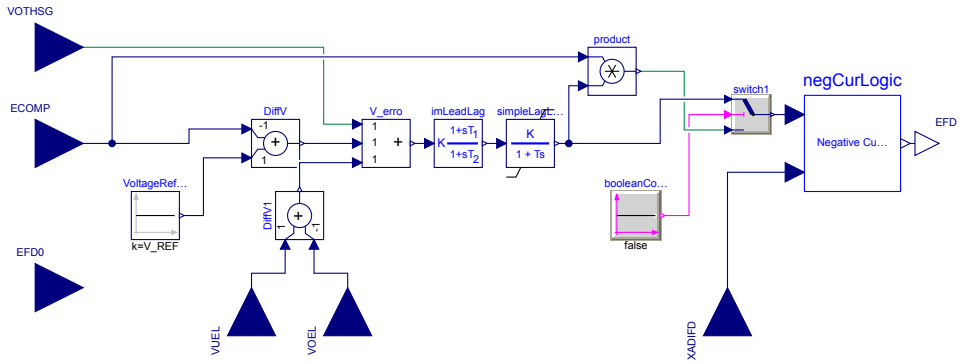


Figure 5. SCRX Controller in OpenIPSL.

Table 3. SCRX Output Signal.

EFD	Output Signal Voltage

used to evaluate the performance of the RCM in handling dynamic events.

4.2 Original SCRX Controller Simulation Result

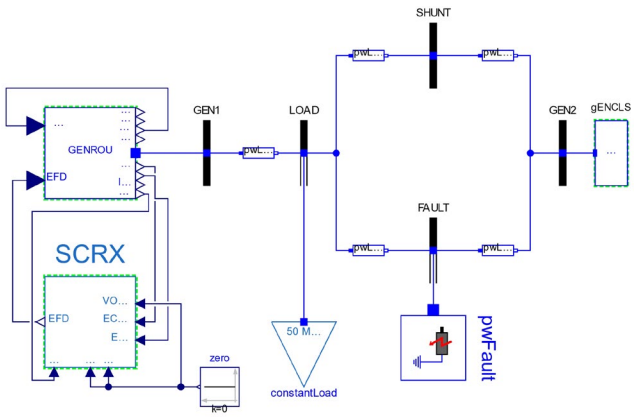


Figure 6. SCRX9 Simulation Example in OpenIPSL.

OpenIPSL contains a model of the SCRX excitation control system, which is shown in Figure 5. In this model, the SCRX is implemented with traditional lead-lag and phase-lag compensators, similar to what is specified in ("IEEE Standard Definitions for Excitation Systems for Synchronous Machines" 2022) and shown in Figure 2. The default parameters in Table 1 are used in this model to compare with the model implemented using the external DLL library. Meanwhile, the grid network is built with OpenIPSL and the generator is controlled by the SCRX RCM, as shown in Figure 6.

Simulating the model in Figure 6 yields the results shown in Figure 7, where two subplots: 1. Generator Voltage (p.u.); 2. SCRX Field Voltage (EFD, Volts). Before the fault occurs, the generator voltage is stable at the reference value of 1.0 p.u., and the field voltage (EFD) is maintained in steady state by the SCRX. When the fault

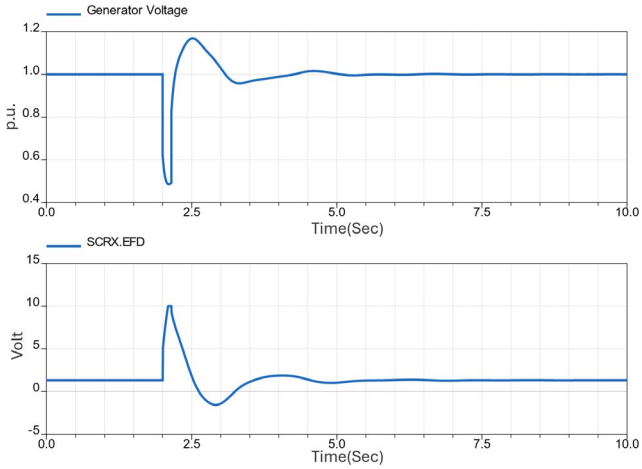


Figure 7. Simulation Result: Bus1 Voltage (Top); SCRX Output Voltage (Bottom).

occurs at 2.0 seconds, there is a significant drop in the generator voltage to approximately 0.4 p.u. The SCRX controller responds by sharply increasing the field voltage to counteract the voltage dip and stabilize the generator. The peak field voltage reaches around 10 Volts (EMAX) shortly after the fault initiation. Once the fault is cleared at 2.15 seconds, the generator voltage initially overshoots about 0.2 p.u. before settling back to the reference value. The SCRX controller adjusts the field voltage accordingly, first reducing it to correct the overshoot and then gradually stabilizing it around the required level to maintain the generator voltage at 1.0 p.u. The performance metrics observed in this simulation can be used as a reference for further testing and comparison with the RCM.

4.3 External Object SCRX Controller Simulation Result

Next, we compare the implementation of the ICDMS using the 'real code' implementation of the SCRX model.

Figure 8 shows the simulation diagram with an excitation controller of the generator replaced with the 'real-code' implementation. The original SCRX controller has 6 inputs. However, 4 of them remain zero during the simulation. Thus, for simplicity of the block, only two feedback ports are preserved, and the input voltage set point is

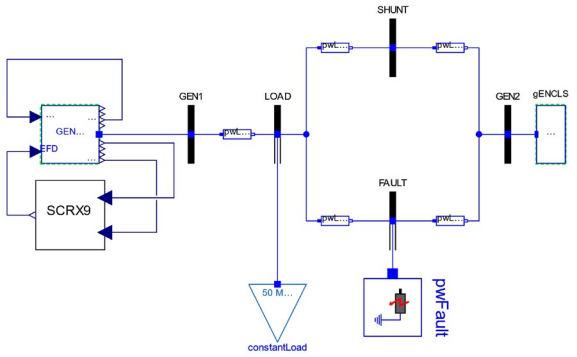


Figure 8. SCRX Controller With Real-Code Implementation.

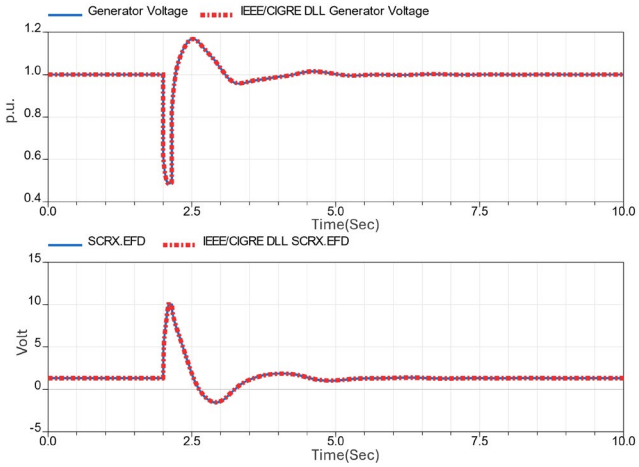


Figure 9. Simulation Result: Bus1 Voltage (Top); SCRX Output Voltage (Bottom).

always set to 1 p.u.

The comparison results shown in Figure 9 indicate that the real code SCRX controller, implemented as a DLL following the ICDMS , performs similarly to the original Modelica-based model from OpenIPSL. This successful integration and matching performance validate the RCM compatibility and robustness within the Modelica simulation environment. Although the comparison of simulation performance (e.g., time required to simulate) was part of our experimental analysis, we observed significant variability in simulation times between different runs. This variability led us to conclude that the operating system's task scheduling had a substantial impact on the simulation time. As a result, we were unable to provide a consistent and meaningful comparison of simulation times between the two approaches.

5 Conclusions and Future Work

By achieving consistent behavior across different implementations, this study confirms that the IEEE/CIGRE DLL Modeling Standard (IDMS) can be implemented in Modelica to support RCMs. These models can be seamlessly integrated with Modelica models of power system components, as shown using the OpenIPSL library, and tested in simulation scenarios. This offers the possibility of performing power system simulations without the need for domain-specific tools, which is valuable for practitioners and researchers who need to develop models that comply with the ICDMS. These results support the broader use of RCMs in power system simulations with Modelica, enhancing the flexibility and reliability of power system simulations and control systems for industrial applications.

The implementation has been tested using the Dymola software. Future work involves releasing the developed code to implement the ICDMS, integrating the examples in this paper into the OpenIPSL library, and conducting tests with OpenModelica.

Although the prototype implementation approach used herein requires one to create treat each RCM individually and, therefore, providing interfacing functions and a Modelica model for each RCM, this process can be automated by developing generic Modelica functions that extract and pass information to a generic DLL. This will be explored in future work.

In addition, future work includes the development of unit testing to assess the performance of the integrated DLLs and determine if additional error handling functions would be required to protect against unexpected DLL numerical errors or other unwanted simulation behavior.

Finally, the authors will explore the potential wrapping of RCMs with FMI and compare the benefits and drawbacks with the approach proposed herein.

Acknowledgements

This paper is in part, based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139.

References

Bogodorova, T. et al. (2013). "A modelica power system library for phasor time-domain simulation". In: *IEEE PES ISGT Europe 2013*, pp. 1–5. DOI: 10.1109/ISGTEurope.2013.6695422.

de Castro, Marcelo et al. (2023). "Version [OpenIPSL 2.0.0] - [iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations]". In: *SoftwareX* 21, p. 101277. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2022.101277. URL: https://www.sciencedirect.com/science/article/pii/S2352711022001959.

IEEE (2007). "IEEE Standard Definitions for Excitation Systems for Synchronous Machines". In: *IEEE Std 421.1-2007 (Revision of IEEE Std 421.1-1986)*, pp. 1–33. DOI: 10.1109/IEEESTD.2007.385319.

"IEEE Standard Definitions for Excitation Systems for Synchronous Machines" (2022). In: *IEEE Std 421.1-2021 (Revision of IEEE Std 421.1-2007)*, pp. 1–45. DOI: 10.1109/IEEESTD.2022.9737077.

Junghanns, Andreas et al. (2021-09-27). "The Functional Mockup Interface 3.0 - New Features Enabling New Applications". In: 14th Modelica Conference 2021. Linköping University

Electronic Press. DOI: 10.3384/ecp2118117. URL: http://dx.doi.org/10.3384/ecp2118117.

Laera, Giuseppe et al. (2022-10). "Guidelines and Use Cases for Power Systems Dynamic Modeling and Model Verification using Modelica and OpenIPSL". In: Proceedings of the American Modelica Conference 2022. Linköping University Electronic Press. DOI: 10.3384/ECP21186146.

Lenord, Oliver et al. (2021-09-27). "eFMI: An open standard for physical models in embedded software". In: 14th Modelica Conference 2021. Linköping University Electronic Press. DOI: 10.3384/ecp2118157. URL: http://dx.doi.org/10.3384/ecp2118157.

Ramasubramanian, Deepak et al. (2024). "Techniques and Methods for Validation of Inverter-Based Resource Unit and Plant Simulation Models Across Multiple Simulation Domains: An Engineering Judgment-Based Approach". In: *IEEE Power and Energy Magazine* 22.2, pp. 55–65. DOI: 10.1109/MPE.2023.3343679.

Rogersten, Robert, Luigi Vanfretti, and Wei Li (2015). "Towards consistent model exchange and simulation of VSC-HVdc controls for EMT studies". In: *2015 IEEE Power & Energy Society General Meeting*, pp. 1–5. DOI: 10.1109/PESGM.2015.7285986.

Schaub, Alexander, Matthias Hellerer, and Tim Bodenmüller (2012-09). "Simulation of Artificial Intelligence Agents using Modelica and the DLR Visualization Library". In: 9th International Modelica Conference. Linköping University Electronic Press. DOI: 10.3384/ecp12076339. URL: http://dx.doi.org/10.3384/ecp12076339.

Vanfretti, Luigi, Tetiana Bogodorova, and Maxime Baudette (2014-03-10). "A Modelica Power System Component Library for Model Validation and Parameter Identification". In: 10th International Modelica Conference. Linköping University Electronic Press. DOI: 10.3384/ecp140961195. URL: http://dx.doi.org/10.3384/ecp140961195.

Vanfretti, Luigi, Wei Li, et al. (2013-01). "Unambiguous power system dynamic modeling and simulation using modelica tools". In: pp. 1–5. DOI: 10.1109/PESMG.2013.6672476.

Zhang, Mengjia et al. (2015-10). "Modelica Implementation and Software-to-Software Validation of Power System Component Models Commonly used by Nordic TSOs for Dynamic Simulations". In: 56th Conference on Simulation and Modelling (SIMS 56). Linköping University Electronic Press. DOI: 10.3384/ecp15119105. URL: http://dx.doi.org/10.3384/ecp15119105.

Decentralised Hydrogen Fuelled Gas Engine CHP Units: A Feasibility Study with Modelica

Florian Andreas Beerlage[1] Naqib Salim[1] Maurice Kettner[1]

[1]Institute of Refrigeration, Air-Conditioning and Environmental Engineering, Karlsruhe University of Applied Sciences, Germany, florian.beerlage, muhamad_naqib.md_salim, maurice.kettner@h-ka.de

Abstract

The use of hydrogen gas as an alternative fuel to power energy systems has been a topic of research over the last few decades and is currently gaining importance, even more due to current circumstances related to decarbonise energy supply. One focus of research is the use of hydrogen gas in combined heat and power gas engines, as this type of energy conversion is known for its high efficiency. For this reason, a cross-border project between France and Germany is developing a living laboratory in the Upper Rhine region to investigate the feasibility of hydrogen gas as an alternative fuel in a holistic decentralised energy system[1]. It consists of several energy components, including a polymer electrolyte membrane electrolyser (PEMEC), gas engine combined heat and power (CHP) unit, photovoltaic (PV) panels, hydrogen storage, thermal and electrical energy storage. To enable and demonstrate multiple what-if scenarios of possible variations of the energy system, a simulation model was developed using Modelica. Users, e.g. local authorities, landlords, businessman etc., of this simulation model could utilize it as a decision support tool for designing a carbon neutral energy system for their own use. This paper describes the development of the model and its application with real measured data from municipal buildings in the city of Offenburg, Germany. The results indicate that the suitability of the model and the use of hydrogen CHPs can be beneficial for this specific use case.

Keywords: Hydrogen, HVAC, CHP, Electrolyser, Gas engine, Cogeneration

1 Introduction

The primary motivation for undertaking these projects is the ambitious objective to reduce greenhouse gas emissions. Germany set goals to reduce these by at least 65% by 2030 and 88% by 2040, compared to 1990 levels (Umweltbundesamt 2023b). These goals align with the Paris Agreement and the Kyoto Protocol, forming part of the climate protection strategies of the EU and the United Nations. Given these targets, hydrogen is likely to play a crucial role in the energy transition due to its potential for carbon-neutral production. On 14^{th} November 2023, Germany's Vice-Chancellor Robert Habeck announced a plan for a 9,700 km hydrogen network, set to start in 2024. This network is part of the European Hydrogen Backbone initiative, comprised of thirty-three energy infrastructure operators with a vision for a climate-neutral Europe supported by a renewable and low-carbon hydrogen market (Reuters 2023; European Hydrogen Backbone 2024). Despite the initiative's early stage, concerns have been raised about the inclusion of small and medium-sized locations, with Offenburg, for example, not being connected to the hydrogen backbone until 2035. Yet, Offenburg aims for carbon neutrality, partly through hydrogen as a green energy carrier. Given that the city already operates gas engine CHP units, an investigation into the feasibility of transitioning them to hydrogen is required. When comparing hydrogen-based gas engine CHP units with fuel cell CHP units, both offer the advantage of no green house gas emissions. While fuel cells have higher electrical efficiency, gas engines often provide better thermal efficiency due to higher combustion temperatures. Additionally, gas engines benefit from shorter startup times and the capability for modulation (Ellamla et al. 2015; Elmer et al. 2015). The purchase, installation and operating costs of gas engine CHP units are also generally lower (see (Danish Energy Agency 2024)). To ensure a precise and practical investigation, measurement data concerning heat and electricity demand from five communal buildings were provided, which are located in Offenburg[2].

This paper is structured as follows: Section 2 describes the model considered in this study and the equations implemented in the newly developed modules. Section 3 presents the validation of the model. Section 4 details the construction, simulation and results of a case study using the data provided by the city of Offenburg. Lastly, section 5 discusses the results and outlines future work.

2 Schematic Model Description

Wherever possible, open-source Modelica libraries compatible with OpenModelica were integrated to ensure the software remains open-source. The *Modelica Buildings library* was utilized for modeling PV systems, batteries, and the grid (Wetter et al. 2014). For the heat pump, an

[1]For more information please visit this website: https://co2inno.com

[2]The software is compatible with OpenModelica and will be published here: https://github.com/IKKUengine

 10.3384/ECP207 80 DOI

empirical approach was adopted based on Ruhnau et al. (2019) for both accuracy and to simplify programming. This methodology allows for the selection between air-, ground-, and water-source heat pumps, as well as between floor heating or radiator heating. Additionally, a simplified thermal energy storage system was implemented to facilitate easier control of the CHP units. This section outlines the modeling methodology for gas engine CHP units, PEMEC, hydrogen storage, compressors, and control strategies.

Table 1. Notation

A_{mem}	m^2	Area membrane
CF		Correction factor
E	V	Operating voltage
E_0	V	Reversible cell voltage
$E_{act,k}$	kJ/mol	Activation energy
F	C/mol	e Faraday constant
I_{cell}	A	Cell current
$J_{0,k}$	A/m^2	Current exchange density at k
J_{cell}	A/m^2	Current exchange density at the anode or cathode
J_0^{ref}	A/m^2	Reference exchange current density
LHV_i	kWh/kg	Lower heating value
M_i	kg/mol	Molar mass of i
P	W	Electrical power
R	$J/(mol \cdot K)$	Universal gas constant
R_{ohm}	Ω	Resistance
SOC		State of Charge of the storage
T	K	Temperature in Kelvin
$\dot{Q}_i$	W	Heat flow
$\dot{V}_i$	Nm^3/h	Volume flow
V_{act}	V	Activation voltage
V_{cell}	V	Cell voltage
V_{con}	V	Transport voltage
V_{ohm}	V	Ohmic voltage
V_{oc}	V	Open circuit voltage
V_{tn}	V	Thermo-neutral voltage
Y		Minimum threshold
Z		Modulation of the CHP plant
m_i	kg	Total mass of fuel needed
$\dot{m}_i$	kg/s	Mass flow rate
n_i		Count
p_i	Pa	Pressure
v_i		Stoichiometric coefficients
α_k		Symmetry factor
σ_{mem}	S/m	Proton conductivity of the membrane
ν		Relative difference
ΔG	kJ/mol	Gibbs free energy
ΔH	kJ/mol	Work enthalpy

2.1 Gas Engine CHP

Two different modelling approaches were carried out. The first one being a gas engine CHP operating with a stationary heat and power output under nominal conditions. Second, an empirical approach was used for a gas engine CHP model that can follow a heat load up to a given maximum and minimum modulation. Both models are designed for heat-driven operation, where sizing and operation are based on the heat demand of the consumer. This is because heat-guided CHPs are the most common (Arbeitsgruppe Erneuerbare Energien-Statistik 2015, pp. 16–17).

One of the most important key performance indices for an gas eninge CHP are the utilisation hours τ. These will help to evaluate the performance of the CHP later on and is defined as:

$$\tau = \frac{E_{CHP,a}}{P_{nom}} \tag{1}$$

where $E_{CHP,a}$ is the energy delivered within one year and P_{nom} is the nominal power of the cogeneration unit. This value can be calculated using either thermal or electrical energy. In this paper only heat energy and power will be considered due to the fact that the CHP is heat guided.

2.1.1 Stationary Gas Engine CHP Model

Normally, gas engine CHP units are running under nominal conditions. Excess heat is stored in a buffer tank. Electricity is either consumed, stored in the battery (BAT) or fed into the grid. When the load is lower, the efficiency of the CHP decreases, so a minimum threshold Y is set as a turn-on condition, which by default is ≥ 50 %:

$$Y = \frac{P_{th,dem}}{P_{th,nom}}, \tag{2}$$

where $P_{th,dem}$ is the thermal heat demand and $P_{th,nom}$ is the thermal heat production of the CHP at nominal conditions. In order to determine the fuel consumption, nominal efficiencies are required. Thereby η_{el} is the ratio of the electrical power P_{el} and the fuel power P_f:

$$\eta_{el} = \frac{P_{el}}{P_f}, \tag{3}$$

and η_{th} – also called heat yield – is the ratio of the useful heat output (thermal power) P_{th} and the fuel power:

$$\eta_{th} = \frac{P_{th}}{P_f}. \tag{4}$$

Since the gas engine is able to be fueled with natural gas, hydrogen, or gas- hydrogen mixture, the fuel power is calculated by:

$$P_f = \sum \dot{m}_i \cdot LHV_i, \tag{5}$$

where $\dot{m}_i$ represents the fuel mass flow rate of i representing CH_4 or H_2 and LHV_i is the lower heating value of

the fuel (compare table 2). The total mass of fuel required can be determined as follows:

$$m_i = \int \dot{m}_i \, dt \tag{6}$$

Table 2. LHV of different fuels (Bender et al. 2020, p. 805)

	value	unit
LHV_{H_2}	33.3	kWh/kg
LHV_{CH_4}	13.9	kWh/kg

2.1.2 Modulation Gas Engine CHP Model

The modulation CHP model is based on an empirical modelling approach based on Berberich et al. (2015) and Höfner (2019). The electrical efficiency is exclusively a function of the nominal electrical power of the CHP plant $P_{el,nom}$ and the modulation Z, which is defined as

$$Z = \frac{P_f}{P_{f,nom}}, \tag{7}$$

with the nominal fuel power $P_{f,nom}$ and Z theoretically ranging between 0 and 1. Within the model, the minimum modulation Z_{min} must be predefined and should always be greater than 0.33 and smaller than 1. P_{el} lies in the range between 50 kW and 18.3 MW according to Berberich et al. (2015). The general empiric relation between Z and the electrical efficiency is defined as:

$$\begin{aligned} \eta_{el} = a_{el} + b_{el} \cdot (Z - Z_{min}) \\ + c_{el} \cdot [\ln(P_{el,nom}) - \ln(P_{el,min})], \end{aligned} \tag{8}$$

where

$$P_{el,min} = P_{el,nom} \cdot Z_{min}, \tag{9}$$

and with the partial derivatives b_{el} and c_{el}. The minimum electrical efficiency a_{el} represents the point from where the tangent plane is spanned. This value can be computed by rearranging the equation and setting in the nominal electrical efficiency $\eta_{el,nom}$ for η_{el} as well as setting Z to 1 (Höfner 2019, p. 16). The parameters b_{el} and c_{el} result out of the research of Berberich et al. (2015) analysing 49 combustion engine CHP plants and are summarized in table 3 (Berberich et al. 2015). In addition to the calculation of the electrical efficiency, the calculation of the thermal efficiency comprises the further variables supply temperature of the heat circuit T_s and the return temperature of the heat circuit T_r. The nominal electrical power $P_{el,nom}$ and the minimum electrical power $P_{el,min}$ need to be replaced in comparison to equation by the corresponding nominal thermal power $P_{th,nom}$ and the minimum thermal power $P_{th,min}$:

$$P_{th,min} = \frac{P_{th,nom}}{P_{el,nom}} \cdot P_{el,min}, \tag{10}$$

which leads to the equation:

$$\begin{aligned} \eta_{th} = a_{th} + b_{th} \cdot (Z - Z_{min}) \\ + c_{th} \cdot [\ln(P_{th,nom}) - \ln(P_{th,min})] \\ + d_{th} \cdot (T_s - T_{s,max}) + e_{th} \cdot (T_r - T_{r,min}) \end{aligned} \tag{11}$$

with the maximum supply $T_{s,max}$ and minimum return temperature $T_{r,min}$. The minimum thermal efficiency equates a_{th} and is reckoned through a rearranging of equation 11. Since the partial derivatives d_{th} and e_{th} have negative signs (compare table 3), a_{th} reaches its minimum if T_s is set to the maximum value and T_r is set to the minimum value. The corresponding terms will be zero. For the calculation of the thermal efficiency η_{th} in equation 11 the modulation Z is needed and redefined as following:

$$Z = \frac{P_{f,\mathrm{tar}}}{P_{f,\mathrm{nom}}} = \frac{P_{th,\mathrm{tar}}}{P_{th,\mathrm{nom}}} \cdot \frac{\eta_{th,\mathrm{nom}}}{\eta_{th}} = X_{th} \cdot \frac{\eta_{th,\mathrm{nom}}}{\eta_{th}}, \tag{12}$$

according to (Berberich et al. 2015, pp. 59–60). Thereby are $P_{f,tar}$ the targeted fuel power and X_{th} the thermal modulation. The equation 11 and equation 12 result in the final equation for η_{th} (Berberich et al. 2015, p. 62):

$$\begin{aligned} \eta_{th} = -\frac{1}{2}\Big\{ & -a_{th} + b_{th} \cdot Z_{min} \\ & - c_{th} \cdot \big[\ln(P_{th,nom}) - \ln(P_{th,min})\big] \\ & - d_{th} \cdot [T_r - T_{r,min}] - e_{th} \cdot [T_s - T_{s,max}] \Big\} \\ & + \Bigg\{ \Bigg(\frac{1}{2}\big(-a_{th} + b_{th} \cdot Z_{min} \\ & - c_{th} \cdot \big[\ln(P_{th,nom}) - \ln(P_{th,min})\big] \\ & - d_{th} \cdot [T_r - T_{r,min}] - e_{th} \cdot [T_s - T_{s,max}] \big) \Bigg)^2 \\ & - b_{th} \cdot X_{th} \cdot \eta_{th,nom} \Bigg\}^{0.5}, \end{aligned} \tag{13}$$

Now the produced electricity P_{el} is determined by:

$$Z = \frac{P_{f,tar}}{P_{f,nom}} = \frac{P_{el}}{P_{el,nom}} \cdot \frac{\eta_{el,nom}}{\eta_{el}}. \tag{14}$$

2.2 Electrolyser

PEMEC are usually selected by the required hydrogen mass flow rates. Mass flow rates from the reaction can be calculated using the following equation with i indicating either water, oxygen or hydrogen:

$$\dot{m}_i = v_i \cdot M_i \cdot \eta_f \cdot N_{cells} \cdot \frac{I_{\mathrm{cell}}}{n \cdot F}, \tag{15}$$

where v_i and M_i are the stoichiometric coefficients and the molar mass, respectively (Sood et al. 2020). These are multiplied by the cell current I_{cell} and divided by the number of moles transferred n and the Faraday constant F

Table 3. Parameters for the calculation of η_{el} and η_{th} according to (Berberich et al. 2015, p. 57)

Parameter	Value	Unit
b_{el}	0.1089	
c_{el}	0.0255	
b_{th}	-0.0746	
c_{th}	-0.0255	
d_{th}	-0.0020	
e_{th}	-0.0017	
Z_{min}	0.33	
Z_{max}	1	
$T_{r,min}$	45	°C
$T_{s,max}$	90	°C

$[9.6485 \cdot 10^4$ C/mol$]$. In this case $n = 2$ and $v_i = 1$ (Sood et al. 2020). η_f represents the Faraday efficiency and N_{cells} the total number of cells within the PEMEC. For calculating I_{cell} losses must be considered. These losses can be expressed by calculating the operating voltage (E) of the PEMEC:

$$E = V_{oc} + V_{act} + V_{ohm} + V_{con}, \tag{16}$$

where V_{oc} represents the open circuit voltage, V_{act} activation voltage, V_{ohm} ohmic voltage and V_{con} the transport voltage. However, V_{con} is negligibly small and do not play a role in this consideration and are therefore not taken into account (Sood et al. 2020). V_{oc} is derived from the Nernst voltage valid for the equilibrium state:

$$V_{oc} = E_0 + \frac{RT}{nF} \cdot \ln\left(\frac{p_{H_2} \cdot \sqrt{p_{O_2}}}{a_{H_2O}}\right), \tag{17}$$

with

$$E_0 = \frac{\Delta G}{nF} = 1.229 \text{ V}, \tag{18}$$

where R stands for the universal gas constant $[8.31447$ J/(mol K)$]$, p_i for the partial pressures of the respective substances involved and T for the temperature in Kelvin. The partial pressures of hydrogen and oxygen are typically determined by the system design. The water activity a_{H_2O} between electrode and membrane corresponds to 1, because water is fed to the cell (Ruiz Diaz 2021). The reversible cell voltage E_0 is then calculated with the Gibbs free energy ΔG $[237.22$ kJ/mol$]$ at standard conditions (Abdin et al. 2015; Ruiz Diaz 2021).

For calculating V_{ohm} the resistance R_{ohm} is needed which mainly includes the resistance due to the membrane and other resistances of the cell components R_{other}:

$$R_{ohm} = \frac{d_{mem}}{\sigma_{mem}} + R_{other}. \tag{19}$$

The quantity d_{mem} denotes the thickness of the membrane. The reference value of $180\,\mu$m was used according to Ojong (2018). R_{other} must be determined experimentally (Sood et al. 2020). The proton conductivity σ_{mem} of the membrane is directly related to the membrane hydration and the operating temperature. For PEM fuel cells, the proton conductivity of the Nafion®- membrane has been studied in detail and can be empirically expressed as a function of membrane hydration and temperature:

$$\sigma_{mem} = (0.005139\lambda - 0.00326) \cdot e^{\left[1268\left(303^{-1} - T^{-1}\right)\right]}, \tag{20}$$

where λ is the hydration number of the membrane, which varies from 14 to 25 (Ojong 2018; Ruiz Diaz 2021; Sood et al. 2020). The degree of hydration of the membrane plays a crucial role in the performance of low-temperature PEM fuel cells. It shows considerable variation, which makes it a critical parameter for determining fuel cell efficiency. On the other hand, for PEM water electrolysis cells, where water is the main transport medium, it is usually assumed that the membrane is always fully hydrated. This is why the hydration number is estimated to be ($\lambda \approx 24$) (Ojong 2018). V_{ohm} can be determined with the use of the cell current I_{cell} or with the cell current density J_{cell} as following:

$$V_{ohm} = \frac{I_{cell}}{R_{ohm}} = \frac{d_{mem}}{\sigma_{mem}} J_{cell}. \tag{21}$$

For the calculation of V_{act}, the exchange current density needs to be obtained. This is typically done with the Butler–Volmer equation:

$$J_{cell} = J_{0,k}\left[e^{\left(\frac{\alpha_k nF}{RT} V_{act,k}\right)} - e^{\left(-\frac{(1-\alpha_k)nF}{RT} V_{act,k}\right)}\right], \tag{22}$$

where α_k is the symmetry factor, $J_{0,k}$ is the current exchange density at k which represents either anode or cathode (Ojong 2018; Ruiz Diaz 2021; Sood et al. 2020). Furthermore J_{cell} can be determined by using the area content of the membrane A_{mem} according to Sood et al. (2020):

$$J_{cell} = \frac{I_{cell}}{A_{mem}}. \tag{23}$$

Lastly the activation voltage $V_{act,k}$ at the anode or cathode can be expressed by the following equation:

$$V_{act,k} = \frac{RT}{F} \sinh^{-1}\left(\frac{J_{\text{cell}}}{2J_{0,k}}\right). \tag{24}$$

$J_{0,k}$ must be determined for cathode and anode using a reference exchange current density J_0^{ref}:

$$J_{0,k} = J_0^{\text{ref}} e^{\left(-\frac{E_{act,k}}{RT}\right)}, \tag{25}$$

where the activation energy $E_{act,k}$ for cathode and anode must be determined experimentally (Ojong 2018; Sood et al. 2020). However, there are publications that use a simplified model using only the exchange current density $J_{0,k}$ without a reference value. **Table 4** gives an overview of

the research results from which the values of Abdin et al. (2015) seem to fit best to this PEMEC model. Other tables can be found in the literature or have to be determined experimentally (Carmo et al. 2013; Ojong 2018; Sood et al. 2020).

Table 4. Overview of electrokinetic parameters. Abdin et al. (2015) parameters were applied.

Parameter	**Abdin et al. (2015)**	Liso et al. (2018)	Marangio et al. (2009)	Ni et al. (2006)
$J_{0,\text{anode}}$	10^{-3}	$5 \cdot 10^{-9}$	10^{-2}	10^{-2}
$J_{0,\text{cathode}}$	10^{3}	10	10	10^{5}
α_{anode}	0.8	1.2	2	0.5
α_{cathode}	0.25	0.5	0.5	0.5

The three important efficiencies to consider are the Faraday efficiency η_f, cell efficiency η_{cell} and the energy efficiency η_e. The Faraday efficiency represents the correlation between the actual and presumed efficiency of the produced hydrogen output and is expressed as:

$$\eta_f = \frac{\dot{m} \cdot n \cdot F}{I \cdot M_{H_2}}. \tag{26}$$

The voltage efficiency η_v is the ratio of the thermoneutral voltage V_{tn}, also called the minimum required voltage, and the actual cell voltage V_{cell}. This requires the work enthalpy ΔH [237.22 kJ/mol] at standard conditions and is expressed as:

$$V_{tn} = \frac{\Delta H}{nF} = 1.48\,\text{V}, \tag{27}$$

$$\eta_v = \frac{V_{tn}}{V_{cell}}. \tag{28}$$

Here the losses due to pressure, mass transport and activation are taken into account (Ruiz Diaz 2021). At this point it is possible to determine the overall efficiency of the cell:

$$\eta_{cell} = \eta_f \cdot \eta_v. \tag{29}$$

Additionally, the energy efficiency is calculated as the ratio of the benefit, measured as H_2 mass flow expressed in generated watts, to the input, the energy balance:

$$\eta_e = \frac{\dot{m} \cdot HHV}{P_{el} - \dot{Q}_{he} + \dot{Q}_{add}}. \tag{30}$$

where HHV is the higher heating value of hydrogen (39.4 kWh/kg) (Bender et al. 2020, p. 805). In addition to the electrical power P_{el}, the heat recovered from the heat exchangers $\dot{Q}_{he}$ and the heat supplied to the system $\dot{Q}_{add}$ may be included in the energy balance.

2.3 Hydrogen storage and compressor

The hydrogen storage dynamics are governed by the equation:

$$p_i = p_0 + CF \cdot \frac{\dot{m}_{H_2} \cdot R \cdot T}{V_{bottle} \cdot n_{bottle} \cdot M_{H_2}}, \tag{31}$$

where p_i is the pressure inside the storage tank, p_0 the initial pressure, CF the correction factor, $\dot{m}_{H_2}$ the mass flow rate of hydrogen, V_{bottle} and n_{bottle} denote the bottle's volume and number of bottles, respectively (Albarghot et al. 2019; Gorgun 2006; Onar et al. 2006). The system is designed with a maximum pressure p_{max} of 80 bar, an initial pressure p_0 of 1 bar, and a bottle volume V_{bottle} of 50 litres, mirroring laboratory setups, and n_{bottle} indicates the count of such bottles. The correction factor (CF), integral to the equation, adjusts for deviations from ideal gas behaviour, essentially a temperature and pressure-dependent ratio of real to ideal gas volumes (Zucker et al. 2019, p. 327). It is equal to one at pressures below 138 bar at ambient temperature, reflecting the model's assumption of a constant room temperature and a slow storage process with a maximum pressure of 80 bar, thus simplifying CF to one for this scenario (McCarty et al. 1981). Using this information the state of charge (SOC) of the storage can be calculated:

$$SOC = \frac{p_i}{p_{max}}. \tag{32}$$

An isothermal compressor has been implemented, assuming an ideal gas as the compression pressures are low. The power of the ideal compressor P_{com} is defined as an integral over the volume flow rate:

$$P_{com} = -\int (p - p_u)\, d\dot{V}, \tag{33}$$

where p is the compression pressure and p_u is the ambient pressure. Using the efficiency η_{com}, the effective compression power required P_{req} can be calculated in terms of electrical power needed:

$$P_{el,req} = \frac{P_{com}}{\eta_{com}}. \tag{34}$$

2.4 Control Strategies

The control strategies play a crucial role in the performance of the energy system. First, a BAT management control sequence has been implemented based on Lu et al. (2019), as the Modelica Buildings library does not offer one. The CHP unit within the heating system plays the most important role. This is because very small or very large heat loads can not be met by the CHP unit, either because it is uneconomical or because the size of the engine does not allow it. The whole control sequence is shown in **Figure 1**, where TES is the thermal energy storage, HP is the heat pump and U is the user. The control system always checks if the TES is charged. If not, the minimum threshold is checked and if this is exceeded, the CHP is switched on. It is switched off when the heat load is no longer required or when the TES is fully charged.

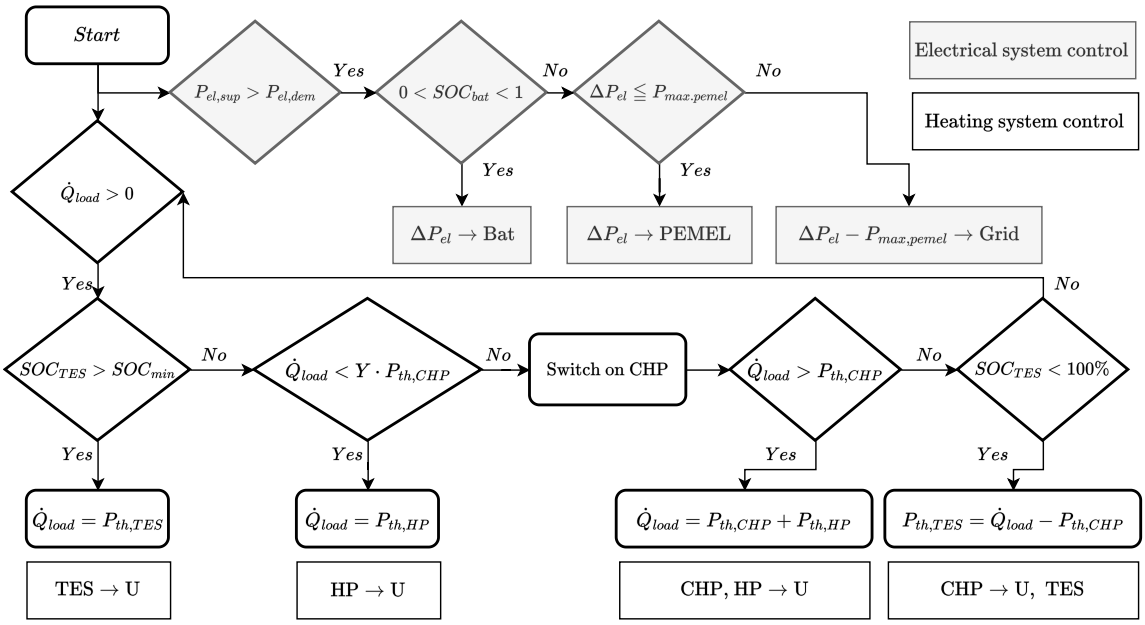


Figure 1. Control sequence of the implemented heating and electrical system

In general, it is possible to operate industrial PEMECs in a grid-connected manner. However, in applications for buildings with smaller PEMECs, low electricity prices are very important to ensure economic viability. For this reason, only self-produced electricity by e.g. PV is used for operation. In case a BAT has been implemented as well, excess energy should first be stored in the BAT before using it for hydrogen production, in order to reduce losses due to lower PEMEC efficiencies. This control sequence is also shown in Figure 1, where ΔP_{el} is the difference between the supplied electrical power $P_{el,sup}$ and the demand $P_{el,dem}$. The power requirement of the compressor is always included when the PEMEC is in operation. If the supply exceeds the PEMEC capacity and the BAT is fully charged, the excess energy is sold to the grid.

Additionally the CHP only runs when sufficient hydrogen is in the tank and is turned off when the hydrogen tank is empty. Similarly to the PEMEC as shown in **Figure 2**.

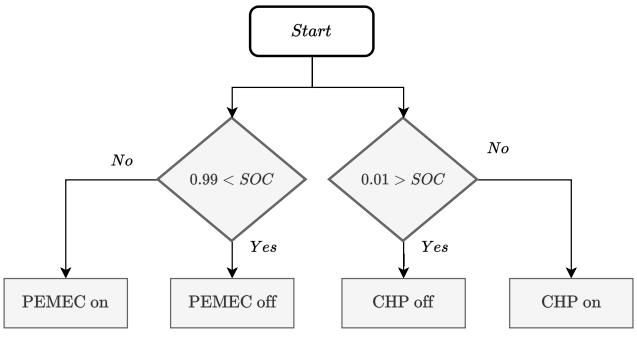


Figure 2. Control sequences of the hydrogen tank.

2.5 CO_2e Calculation

For calculating CO_2e emissions the emission factors need to be known. Sources of CO_2e emissions include combustion of methane gas (202 gCO_2e/kWh (Umweltbundesamt 2023a)) and electricity production, varying by location and energy mix. In 2022, emission factors were 366 gCO_2e/kWh for Germany and 66 gCO_2e/kWh for France (European Environment Agency 2023). Using only green electricity, emission factors differ based on the renewable source, ranging from 4 gCO_2e/kWh for hydro power to 475 gCO_2e/kWh for liquid biomass (Lauf et al. 2022, p. 40). The estimated green emission factors are approximately 66 gCO_2e/kWh for Germany and 31 gCO_2e/kWh for France, reflecting their respective green energy mixes (Arbeitsgruppe Erneuerbare Energien-Statistik 2024; L'Agence ORE et al. 2024).

3 Validation

Laboratory measurements, literature, and manufacturer data were utilized to validate the PEMEC and gas engine CHP models. Technical data are provided in the appendix in **Table 8** and **Table 9**. Initially, the PEMEC's validation involved conducting measurements at different hydrogen outlet pressures (6, 8, and 10 bar) in the university laboratory. **Table 5** is a summary of the measured values compared with the simulation results at a 6 bar outlet pressure, with the relative deviation, ν, calculated using the theoretical value as a reference. This procedure was also applied to data measured at 8 and 10 bar, revealing an average relative deviation of 10%.

Due to its high complexity the modulation model needed to be validated. Höfner (2019) has developed a model specifically for CHP, rather than a general ap-

proach. Their efficiency curves are therefore suitable for comparison and validation, as also shown in **Figure 3**. Furthermore, the university has the manufacturer's specifications for a hydrogen-fuelled CHP unit, which are compared with the simulation results in **Figure 4**. A deviation is present, yet it remains within acceptable limits. Note that both gas engine CHP units are from the same manufacturer (compare **Table 8**) (2G Energy AG 2024).

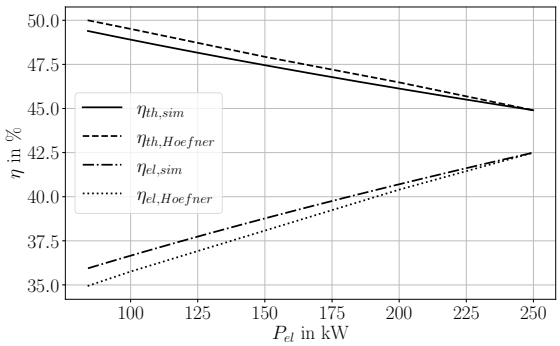


Figure 3. Comparison of Simulation Efficiencies and Höfner (2019) Efficiencies of agenitor 406 Gas Engine CHP

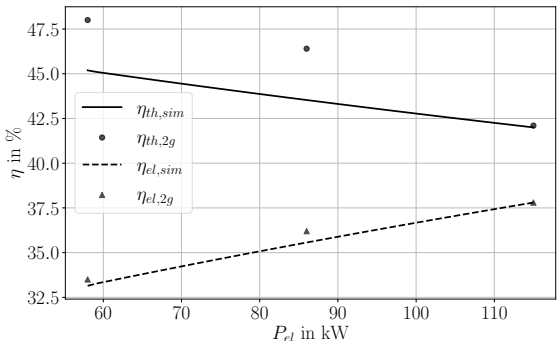


Figure 4. Comparison of Simulation Efficiencies and Manufacturer Specification of agenitor 404 H_2 Gas Engine CHP

Table 5. Comparison of measurements and simulation at 6 bar outlet pressure and $\dot{V}$ in Nm^3/h

P_{el}	$\dot{V}_{H_2,real}$	$\dot{V}_{H_2,sim}$	ν
334.5	0.050	0.054	0.08
555.0	0.100	0.090	0.11
817.7	0.150	0.133	0.13
1138.8	0.200	0.185	0.08
1357.8	0.222	0.220	0.01

4 Case Study

For this project, the city of Offenburg provided hourly data for heat and electricity consumption, used in all simulations. In agreement with project partners, the five buildings were treated as one system to achieve climate neutrality, requiring the system to be designed around the CHP, which needs to be dimensioned first. The CHP aims to cover the base heat demands while the HP operates as supportive heat generator.Here, the classic mode of operation of a gas-fuelled CHP unit is copied for hydrogen-fuelled units .

The dimensioning of a heat-guided CHP plant relies on the descending sorted annual load duration curve of the consumer. The economic optimum for the classical CHP and a peak load boiler, an HP, is sought, aiming for 5,000 to 6,000 full utilization hours or 10% to 30% coverage of thermal heat demand (Arbeitsgruppe Erneuerbare Energien-Statistik 2015; Sokratherm GmbH n.d.; Verbraucherzentrale 20.05.2021). 4,000 full utilisation hours or 15% of nominal thermal capacity at maximum heating demand was chosen as these buildings have no domestic hot water demand and low summer heating demand. The optimal fit would be a CHP with a nominal thermal capacity range $P_{th,nom}$ of 33 to 38 kW. A commercially available hydrogen CHP was chosen, the smallest available being the MAH 33.3 TI 311A from MAMOTEC energy solutions (see technical data in **Table 6**) (MAMotec GmbH 2024).

Table 6. Technical data of hydrogen and natural gas CHP (MAMotec GmbH 2024).

	MAH 33.3 TI 311A
Fuel	Hydrogen
P_{el}	38 kW
P_{th}	53.7 kW
η_{el}	35.5 %
η_{th}	50.2 %
η_{total}	85.7 %

Given that Offenburg will not be connected to the European hydrogen grid until at least 2030, a decentralized, standalone operation is more realistic in the near future. Hydrogen is produced by PEMEC, stored in tanks, and then burned by a CHP when needed. Most of the electricity demand is met by renewable energy sources, with excess energy stored in a BAT and used to operate the PEMEC.

The low density of hydrogen poses storage challenges, with the hydrogen tank being the limiting factor. Large tanks can serve as seasonal hydrogen sinks but require significant space and investment. Therefore, the hydrogen tank size will be investigated through a sweep, keeping the PV size constant. The tank pressure is maintained at 80 bar throughout the study.

Before investigating different tank sizes, the PEMEC size must be determined. Several test simulations indicated that a 500 kW PEMEC is optimal due to the uniformity of hydrogen production, resulting in high full load hours for the PEMEC. Diversifying the electricity mix

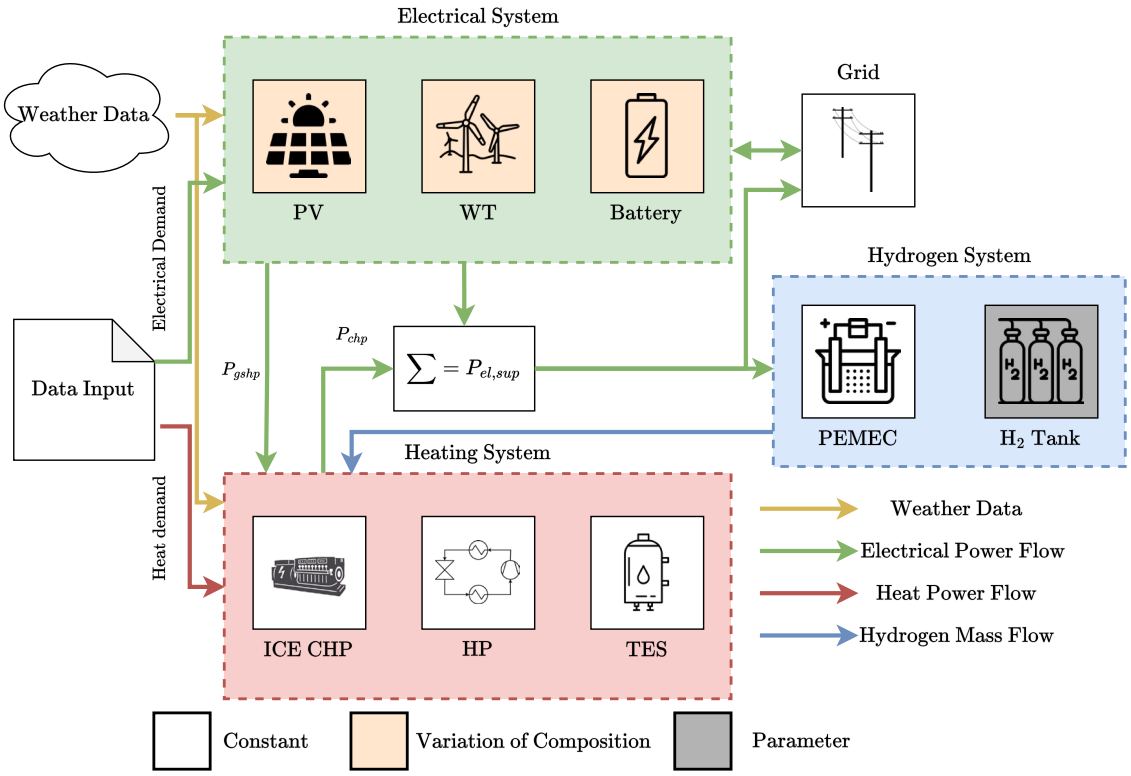


Figure 5. Schematic set up of the decentralised stand-alone operation scenario with a modulation CHP

with PV and Wind Turbines (WT) is prudent. Reducing the PV area to provide the required amount of renewable energy is cost-efficient, while reducing the size of a WT is more challenging form a technical point of view.

Figure 5 shows the schematic structure of this scenario. The Modelica model was implemented identically to the schematic setup. The orange blocks indicate the variation combination, the grey block indicates the parameter for the sweep, and the white blocks indicate constants throughout the sweeps. Only the modulation operation strategy of the CHP is investigated, based on previous results. These showed that with this described control strategy and under the condition that hydrogen can be supplied at any time, the modulation CHP needed 900 kg less hydrogen per year than a stationary CHP (13,700 kg). The downside is a worse efficiency and less heat coverage of only 56% compared to 59%. As the production of hydrogen is very expensive, it should only be used sparingly. For this reason, the modulation was chosen.

5 Results

A total of six scenarios are considered and CO_2e emissions are calculated for comparison as this is the target to be reduced. The results are summarised in **Figure 6**. The labels indicate the composition of the scenario according to **Table 7**.

Figure 6 shows the emissions in tonnes of CO_2e for different storage capacities. Sources of emissions during operation are from natural gas combustion or depending on the electricity demand emission factor (see section 2.5). Since only hydrogen is burned, all emissions are due to the electricity required from the grid. In this scenario, only green electricity was consumed, using the estimated green emission factor of 66 gCO_2e/kWh for Germany. It demonstrates that the combination of PV, WT, and BAT leads to the lowest emissions. Nevertheless, due to drought periods where no electricity is produced, this strategy is unable to entirely eliminate emissions. It also states that the WT generated by this facility is insufficient for its intended purpose. Consequently, it would be necessary to expand the plant. However, the geographical location is not optimal for the generation of wind power.

Furthermore an increase in the size of the hydrogen storage tank has a negligible impact on emissions from a volume of 500 m^3, particularly when WT and PV are com-

Table 7. Dimensioning and composition of the decentralised energy system with the MAH 33.3 TI 311A CHP.

Description	PV	PV and WT
CHP	38 kW_{el}, 53.7 kW_{th}	38 kW_{el}, 53.7 kW_{th}
PV	1.8 MWp	0.9 MWp
BAT	500 kWh	500 kWh
WT	-	0.5 MWp
HP	197 kW	197 kW

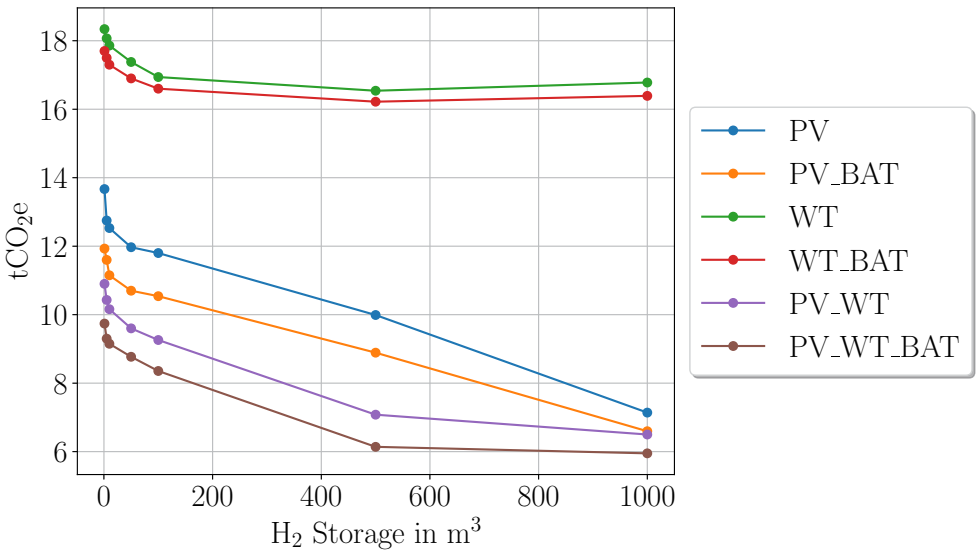


Figure 6. CO_2e emissions depending on hydrogen storage size (at 80 bar)

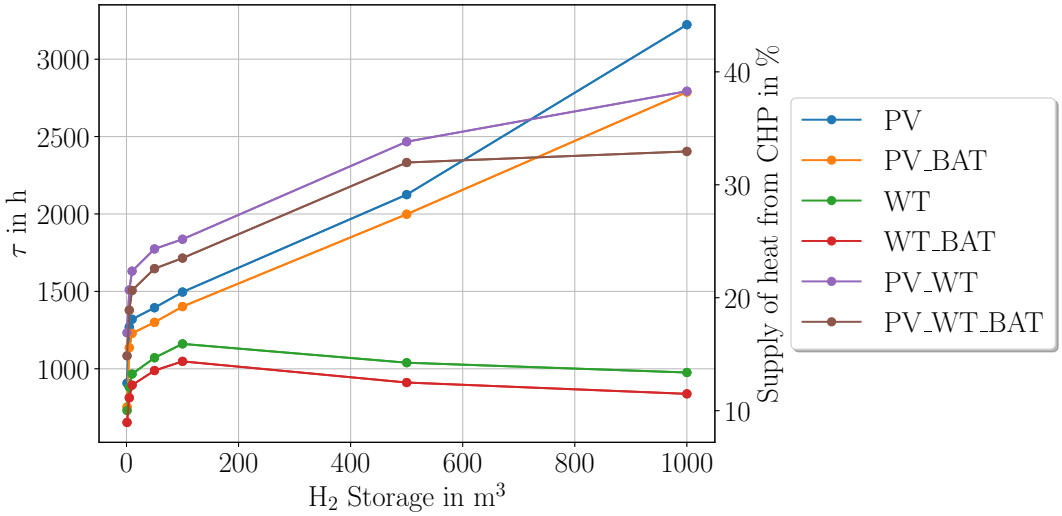


Figure 7. Full utilisation hours and heat coverage of the CHP.

bined. However, tank sizes larger than 50 m^3 are unrealistically large. Therefore, the feasibility of such a system needs to be investigated using other control strategies.

The full utilization hours of the CHP depend on hydrogen production. This depends on the amount of electrical energy available, the size of the PEMEC, and the size of the hydrogen tank. For this reason, the **Figure 7** shows that the full utilization hours also increase as the tank size increases. The only exceptions are the two scenarios where wind power is the sole source of electricity. In these cases, the full load hours decrease as the tank size increases due to the minimum threshold of the control system, which means the tank can only be emptied when a certain amount is available. Here, the electricity production of the WT is clearly too low. Conversely, the highest full utilization hours can be achieved with pure PV and a larger hydrogen storage tank.

6 Discussion and Outlook

In this paper, the feasibility of a hydrogen-powered gas engine CHP unit in a decentralised energy system has been investigated using a real use case with data from the city of Offenburg in Germany. The ultimate goal is to reduce emissions in order to achieve carbon neutrality or come close to this target.

The results of the simulation models indicate several

promising benefits of hydrogen CHP units. In particular, this system can significantly reduce carbon emissions when integrated into decentralised energy systems. However, hydrogen storage requires a lot of space, which is questionable in real-world conditions and probably not realistic. Smaller storage sizes already reduce CO_2e, but other operating strategies may be more efficient in terms of CO_2e emissions and need to be compared while considering costs as well. Therefore, a new control strategy needs to be investigated, where the CHP covering only peak heat demand.

A cost calculation has already been added, calculating investment and operating costs, as well as the levelized costs of electricity and hydrogen. In the future, the calculation of CO_2 emissions will be improved by including emissions from the manufacture of the equipment used. In addition, improvements to the empirical approach are being considered with the CHP units available in the laboratory, as well as improvements to all other models using measured data from the university laboratory where possible. For example, by implementing a hydrogen tank with higher storage pressures. For a continuation of this project, an optimisation tool could be used to optimally dimension the system.

Please refer to the link for the latest version of the model: `https://github.com/IKKUengine/CO2InnO-H2-CHP-Demonstrator`.

Acknowledgements

This work was co-funded by Interreg through the CO2InnO project. The authors would like to thank Lukas Stahl and René Behmann for their early support in implementing the software. Special thanks to Natalie Miller and Yamit Ibarrasuarez from the City of Offenburg for their cooperation.

References

2G Energy AG (2024). *agenitor | 75-450 kW | Der globale Effizienzmaßstab : 2G Energy*. URL: https://2-g.com/de/produkte/agenitor (visited on 2024-02-15).

Abdin, Z., CJ. Webb, and E.MacA. Gray (2015). "Modelling and simulation of a proton exchange membrane (PEM) electrolyser cell". In: *International Journal of Hydrogen Energy* 40.39, pp. 13243–13257. ISSN: 0360-3199. DOI: 10.1016/j.ijhydene.2015.07.129.

Albarghot, Mohamed M. et al. (2019). "Sizing and Dynamic Modeling of a Power System for the MUN Explorer Autonomous Underwater Vehicle Using a Fuel Cell and Batteries". In: *Journal of Energy* 2019, pp. 1–17. ISSN: 2356-735X. DOI: 10.1155/2019/4531497.

Arbeitsgruppe Erneuerbare Energien-Statistik (2015). *BHKW-Fibel: Wissen in kompakter Form*. Essen: energieDRUCK Verlag für sparsamen und umweltfreundlichen Energieverbrauch. URL: https://asue.de/sites/default/files/asue/themen/blockheizkraftwerke/2015/broschueren/asue_050315_bhkw_fibel.pdf.

Arbeitsgruppe Erneuerbare Energien-Statistik (2024-03-08). *Erneuerbare Energien in Zahlen*. Umweltbundesamt. Publisher: Umweltbundesamt. URL: www.umweltbundesamt.de/themen/klima-energie/erneuerbare-energien/erneuerbare-energien-in-zahlen#uberblick (visited on 2024-04-23).

Bender, Beate and Göhlich Dietmar, eds. (2020). *Dubbel Taschenbuch für den Maschinenbau 1: Grundlagen und Tabellen*. 26th ed. Vol. 1. Berlin: Springer. ISBN: 978-3-662-59710-1. DOI: 10.1007/978-3-662-59711-8.

Berberich, Magdalena et al. (2015). *SOLAR-KWK – Entwicklung multifunktionaler Systeme zur solar unterstützten Kraft-Wärme-Kopplung – solare Fernwärme und saisonale Wärmespeicher für die Energiewende*. Stuttgart.

Carmo, Marcelo et al. (2013). "A comprehensive review on PEM water electrolysis". In: *International Journal of Hydrogen Energy* 38.12, pp. 4901–4934. ISSN: 0360-3199. DOI: 10.1016/j.ijhydene.2013.01.151.

Danish Energy Agency (2024). *Technology Catalogues*. The Danish Energy Agency. URL: www.ens.dk/en/our-services/technology-catalogues (visited on 2024-04-10).

Ellamla, Harikishan R. et al. (2015-10-20). "Current status of fuel cell based combined heat and power systems for residential sector". In: *Journal of Power Sources* 293, pp. 312–328. ISSN: 0378-7753. DOI: 10.1016/j.jpowsour.2015.05.050.

Elmer, Theo et al. (2015-02). "Fuel cell technology for domestic built environment applications: State of-the-art review". In: *Renewable and Sustainable Energy Reviews* 42, pp. 913–931. ISSN: 13640321. DOI: 10.1016/j.rser.2014.10.080.

European Environment Agency (2023). *Greenhouse gas emission intensity of electricity generation*. Greenhouse gas emission intensity of electricity generation. URL: www.eea.europa.eu/data-and-maps/daviz/co2-emission-intensity-14/#tab-googlechartid_chart_41 (visited on 2024-04-23).

European Hydrogen Backbone (2024). *The European Hydrogen Backbone (EHB) initiative | EHB European Hydrogen Backbone*. URL: www.ehb.eu (visited on 2024-01-04).

Gorgun, H. (2006). "Dynamic modelling of a proton exchange membrane (PEM) electrolyzer". In: *International Journal of Hydrogen Energy* 31.1, pp. 29–38. ISSN: 03603199. DOI: 10.1016/j.ijhydene.2005.04.001.

Höfner, Peter (2019). *Vergleich strom- und wärmegeführter Betriebsweise eines BHKW im Nahwärmenetz mit Langzeitwärmespeicher*. Vienna.

L'Agence ORE et al. (2024-03-08). *Panorama de l'électricité renouvelable*. URL: www.assets.rte-france.com/prod/public/2023-07/2023-07-19-panorama-energies-renouvelables-2022.pdf (visited on 2022-12-31).

Lauf, Thomas, Michael Memmler, and Sven Schneider (2022-12-09). *Emissionsbilanz erneuerbarer Energieträger 2021*. Umweltbundesamt. 170 pp. URL: www.umweltbundesamt.de/publikationen/emissionsbilanz-erneuerbarer-energietraeger-2021 (visited on 2024-04-23).

Liso, Vincenzo et al. (2018). "Modelling and Experimental Analysis of a Polymer Electrolyte Membrane Water Electrolysis Cell at Different Operating Temperatures". In: *Energies* 11.12. ISSN: 1996-1073. DOI: 10.3390/en11123273.

Lu, Xing et al. (2019). "An Open Source Modeling Framework for Interdependent Energy-Transportation-Communication Infrastructure in Smart and Connected Communities". In: *IEEE Access* 7. Conference Name: IEEE Access, pp. 55458–55476. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2913630.

MAMotec GmbH (2024). *Gasmotoren Überciht*. URL: https://mamotec-online.de/gasmotoren-uebersicht/ (visited on 2024-06-03).

Marangio, F., M. Santarelli, and M. Calì (2009). "Theoretical model and experimental analysis of a high pressure PEM water electrolyser for hydrogen production". In: *International Journal of Hydrogen Energy* 34.3, pp. 1143–1158. ISSN: 0360-3199. DOI: 10.1016/j.ijhydene.2008.11.083.

McCarty, RD., J. Hord, and H. M. Roder (1981). *Selected properties of hydrogen (engineering design data)*. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/MONO/nbsmonograph168.pdf.

Ni, Meng, Mkh Leung, and Y. C. Leung (2006). "Electrochemistry Modeling of Proton Exchange Membrane (PEM) Water Electrolysis for Hydrogen Production". In: *Semantic Scholar*. URL: https://api.semanticscholar.org/CorpusID:45236790.

Ojong, Tabu Emile (2018). *Characterization of the Performance of PEM Water Electrolysis Cells operating with and without Flow Channels, based on Experimentally Validated Semi-empirical Coupled-Physics Models*. Ed. by Fraunhofer. DOI: 10.24406/publica-fhg-282794.

Onar, O. C., M. Uzunoglu, and M. S. Alam (2006). "Dynamic modeling, design and simulation of a wind/fuel cell/ultra-capacitor-based hybrid power generation system". In: *Journal of Power Sources* 161.1, pp. 707–722. ISSN: 03787753. DOI: 10.1016/j.jpowsour.2006.03.055.

Reuters (2023-11-14). "Wasserstoff: Robert Habeck kündigt fast 10.000 Kilometer langes Netz an". In: *Der Spiegel*. ISSN: 2195-1349. URL: www.spiegel.de/wirtschaft/unternehmen/habeck-kuendigt-fast-10-000-kilometer-langes-wasserstoffnetz-an-a-2821473f-6e07-47dd-a1f0-15c9ab45f6a6 (visited on 2023-11-22).

Ruhnau, Oliver, Lion Hirth, and Aaron Praktiknjo (2019-10-01). "Time series of heat demand and heat pump efficiency for energy system modeling". In: *Scientific Data* 6.1. Publisher: Nature Publishing Group, p. 189. ISSN: 2052-4463. DOI: 10.1038/s41597-019-0199-y.

Ruiz Diaz, Daniela Fernanda (2021). *Mathematical Modeling of Polymer Electrolyte Membrane Water Electrolysis Cell with a Component-level Approach*. Ed. by UC Irvine. URL: https://escholarship.org/uc/item/8cv660cn.

Sokratherm GmbH (n.d.). *Dimensioning of CHP units up to 2 MWel*. URL: https://www.sokratherm.de/wp-content/uploads/auslegungsgrundsaetze-08-1-wm-eng.pdf.

Sood, Sumit et al. (2020). "Generic Dynamical Model of PEM Electrolyser under Intermittent Sources". In: *Energies* 13.24. ISSN: 1996-1073. DOI: 10.3390/en13246556.

Umweltbundesamt (2023a-10-31). *Kohlendioxid-Emissionsfaktoren für die deutsche Berichterstattung atmosphärischer Emissionen*. URL: https://view.officeapps.live.com/op/view.aspx?src=https%3A%2F%2Fwww.umweltbundesamt.de%2Fsites%2Fdefault%2Ffiles%2Fmedien%2F361%2Fdokumente%2Fco2_ef_liste_2024_brennstoffe_und_industrie_final.xlsx&wdOrigin=BROWSELINK (visited on 2024-05-08).

Umweltbundesamt (2023b-05-02). *Treibhausgasminderungsziele Deutschlands*. Umweltbundesamt. Publisher: Umweltbundesamt. URL: www.umweltbundesamt.de/daten/klima/treibhausgasminderungsziele-deutschlands (visited on 2023-11-08).

Verbraucherzentrale (20.05.2021). *Kleine Blockheizkraftwerke: Die Heizung, die auch Strom liefert*. URL: https://www.verbraucherzentrale.de/wissen/energie/heizen-und-warmwasser/kleine-blockheizkraftwerke-die-heizung-die-auch-strom-liefert-6007.

Wetter, Michael et al. (2014). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: https://doi.org/10.1080/19401493.2013.765506.

Zucker, Robert D. and Oscar Biblarz (2019). *Fundamentals of gas dynamics*. 3rd ed. Hoboken, NJ: Wiley. ISBN: 978-1-119-48170-6.

Appendix

Table 8. Technical data of the gas engine CHP units (2G Energy AG 2024; Höfner 2019)

	agenitor 404c H_2	agenitor 404	agenitor 406
P_{el}	115 *kW*	100 *kW*	250 *kW*
P_{th}	129 *kW*	130 *kW*	264 *kW*
η_{el}	0.377	0.384	0.425
η_{th}	0.423	0.499	0.449
η_{total}	0.80	0.883	0.874

Table 9. Technical data PEMEC

	Value	Unit
n_{cell}	10	-
n_{stack}	1	-
$P_{\text{el,max}}$	1900	W
p	0 - 10	bar
V_{op}	230	V
$\dot{V}_{H_2,\text{max}}$	0.3	Nm^3/h

FMI-3.0 export for models with clock in a signal flow diagram environment

Masoud Najafi Ramine Nikoukhah

Altair Engineering, France {masoud,ramin}@altair.com

Abstract

The FMI-3.0 standard, recently released, introduces several promising features, such as clocks and arrays. FMI-3.0 supports various clock types, including time-based clocks, triggered input and triggered output clocks. **Altair Twin Activate** (TA), as a modeling and simulation environment, inherently supports hybrid systems combining continuous-time and discrete-time models. The discrete-time part is typically activated by events and clocks. The clock types provided by FMI-3.0 however may differ from those in TA. In the paper (Najafi and Nikoukhah 2022), we explained how different clocks defined in FMI-3.0 can be successfully imported into TA. Building upon this, our current paper aims to demonstrate how various clocks used in TA can be used in the export of a subsystem in both FMI-3.0 and FMI-2.0 formats. Specifically, we will explain the way input periodic clocks and input triggered clocks are exported.

Keywords: FMI, Synchronous clock, Signal based tool, Modelica tool

1 Introduction

The Functional Mock-up Interface (FMI) (Modelica Association 2022) has become the de facto tool-independent standard for the exchange of dynamic models and co-simulation. The FMI-3.0 version (Specification 2022) introduces numerous new features that enable more advanced modeling and support for co-simulation algorithms. Clocks facilitate the synchronization of events between Functional Mock-up Units (FMUs) and the simulator (importer). Additionally, several new data types and multi-dimensional arrays are now supported (Junghanns et al. 2021).

Altair Twin Activate is a modeling and simulation tool developed by Altair Engineering, built on the open-source academic simulation software **Scicos** (INRIA n.d.). The TA environment allows users to create models of dynamical systems using signal-based block diagrams. Basic blocks, such as FMUs, can be interconnected to construct complex models. This approach is very similar to the way diagrams are created in the SSP (System Structure and Parametrization) standard [1].

TA can also be used to create Modelica diagrams (Nikoukhah and Furic 2009). The process begins with aggregating Modelica components to create a Modelica program, which is then processed by the Modelica compiler[2]. In TA, the Modelica compiler generates an FMU block that replaces the Modelica components in the original model. The resulting FMU for Modelica supports both ModelExchange or CoSimulation.

Due to this FMI-based integration of Modelica in TA, the tool offers FMU import support via a TA FMU block. More generally, this block can be used to import FMUs from other vendors (Nikoukhah, Najafi, and Nassif 2017).

With FMI-3.0 and the introduction of clocks, activation, and synchronization, FMU import and export in TA presents new challenges. Although activation signals and synchronization have been integral parts of the TA semantics from the beginning, slight semantic differences between FMI-3.0 and TA formalism prevent FMUs from being imported or exported like other native blocks in TA. This issue also existed, to a lesser extent, with FMI-2.0, as discussed in (Nikoukhah, Najafi, and Nassif 2017). The challenges and solutions for FMI-3.0 import have been presented in (Najafi and Nikoukhah 2022).

This paper addresses the difficulties and proposed solutions for providing extended support for FMI-3.0 export in TA. It begins with an overview of how TA handles activations (clocks) and discusses the differences with FMI-3.0's clock handling. Then, it presents solutions for exporting models as FMI-3.0 in TA, focusing on periodic and triggered input clock types.

1.1 Activation signals in TA

Activation signals in TA control the execution of block functions and can be explicitly manipulated, providing powerful modeling capabilities within the simulation environment. These signals are associated with red links connected to ports typically located at the top and bottom of blocks, as illustrated in Figure 1.

Activation signals are used to specify the activation times of the blocks to which they are connected. The most common usage involves the activation of blocks at a fixed frequency using signals generated by a `SampleClock` block. This block produces a series of isolated activations, known as events, which are regularly spaced in time. These events correspond to the clock ticks in FMI-3.0.

In TA, events can be explicitly manipulated: they can be conditionally subsampled, and unions and intersections of

[1] `https://ssp-standard.org/`

[2] The Maplesim Modelica compiler is used in TA

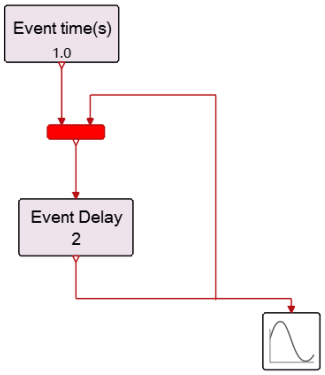


Figure 1. Event Delay model

events can be constructed. Blocks can generate delayed events, enabling operations such as event delaying. In the model shown in Figure 1, the output activation port of an event delay block is fed back to its input activation port. This setup creates a sequence of events where the time spacing between successive events corresponds to the value of the delay.

A first event generated by the `Event time(s)` block initiates the cycle, producing its first (and only) event at 1.0 seconds. The union of this activation signal and the activation signal fed back from the `EventDelay` block is generated by the red "Event Union" block, which then activates the `EventDelay` block at 1.0 seconds. At this point, the `EventDelay` block creates an event delayed by 2.0 seconds, so the next event will occur at 3.0 seconds. The simulation result of the model in Figure 1 is shown in Figure 2. Since the `EventDelay` block's activation output triggers itself, it continues to create events every two seconds for the remainder of the simulation. This combination of blocks mimics the behavior of an EventClock block, and indeed, the `EventClock` and `SampleClock` blocks are constructed with the same principle in mind.

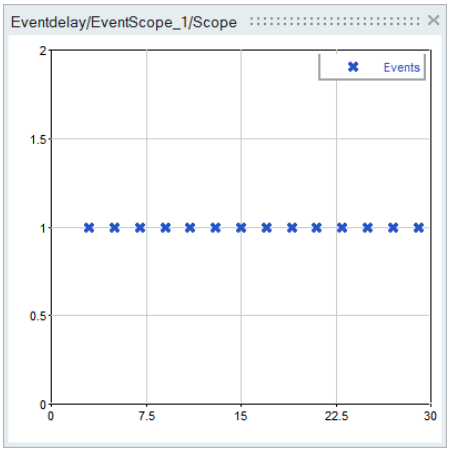


Figure 2. Event Scope results

Output events are defined by their time instants. Based on how the time instant of an event is determined, there are two different types of events in TA: predictable and unpredictable events.

Predictable or programmed events: When activated at any event time instant, a block can schedule another **event** on its activation output ports either at the current time instant or at any future time. The block specifies the event firing delay, *i.e.,* the duration after the block execution when the event should occur, for each of its output activation ports.

The block can also schedule initial output events. For instance, the block **Event time(s)** only schedules initial events and remains inactive during simulation.

The programmed events can be considered as similar to time-based clocks in FMI-3.0, in particular, `changing` and `countdown` time-based clock types.

Unpredictable or zero-crossing events: Activation signals may also be produced by blocks activated in continuous-time. If something happens inside the block, the block can program an event immediately or in the future. The `EdgeTrigger` block is a good example that produces an event based on a zero-crossing test. It generates an event when a condition occurs, such as when a variable reaches a threshold value. Figure 3 represents the simple model of a thermostat and its results (Figure 4).

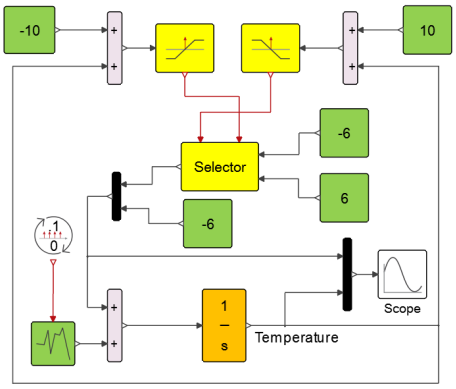


Figure 3. Simple Thermostat model

Two yellow `EdgeTrigger` blocks are used to activate the heater or the cooler when the temperature falls below -10 or rises above 10. These events trigger the `SelectInput` block, which, depending on the activation port through which it is activated, copies its first or second input (values -6 or +6) to its output. This output represents the heat flow added to a random signal and fed to an integrator, the output of which represents the temperature. The simulation results illustrate how the thermostat functionality is implemented by the zero-crossing blocks. This kinds of events are similar to the `triggered output` clock type in FMI-3.0.

The activation signals encountered so far are series

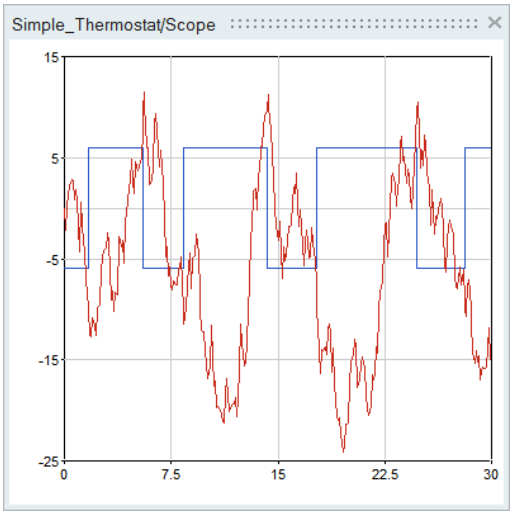


Figure 4. Results from Simple Thermostat model

of events, which are isolated activation signals in time, *i.e.*, discrete events. However, activation signals can be more general and include time intervals. The simplest activation signal of this type is the *always active* activation signal. Many basic blocks in TA palettes, such as the `SineWaveGenerator` or the `Integral` block, are "always active" by default, *i.e.*, they are (implicitly) activated by the *always active* block; they are active in continuous-time. There is no similar clock or activation type in FMI-3.0.

Another activation signal is the "initial activation". Some blocks are only activated once at the initialization phase, just before the start of the simulation. For example, the `Constant` block is declared initially active.

In the model in Figure 1, a sequence of events firing at regular intervals was created using the `EventDelay` block. This was achieved by programming an event on a regular basis. The resulting activation signal resembles the signal produced by the `SampleClock` block, but it is not of the same type. The one produced by the `SampleClock` block is of type *periodic*. The compiler recognizes this signal as periodic, which contains events firing periodically and synchronously with all other `SampleClock` blocks in the model. When a block is activated by a periodic signal, it has access to the period and offset information at compile time. This allows the block to adapt its behavior by computing specific block simulation parameters. For instance, the `SampledData` block computes the discrete-time linear system matrices corresponding to the discretization of a continuous-time linear system for the operating frequency. This frequency, which is the inverse of the sampling (activation) period, is available at compile time. `SampledData` block cannot be activated with non-periodic clock. The regular time-spaced events is similar to periodic time-based clocks in FMI-3.0, particularly fixed and constant clock types.

1.2 Synchronous vs. asynchronous activations

Activation signals are characterized by time periods or time instances. An event, for example, defines an isolated point in time specifying the time instant when the blocks receiving the event should be activated. However, the time of the event does not fully characterize the event, especially its relationship with other events. Two events may have identical times (simultaneous) but not be synchronous.

When two blocks are activated by the same event, the compiler must compute the order in which they should be activated depending on their connections and direct dependencies between inputs and outputs (port feedthrough properties) of blocks. If a block requires the value on one of its inputs to compute its output and this input is connected to the output of another block, then the latter block should be executed first. Generally, for any activation signal, the compiler computes an execution order of blocks. This order includes the blocks receiving the activation signal directly, or indirectly through inheritance.

Each "distinct" activation source has its own list of blocks and is treated independently of other activation sources. Even if two events produced by two "distinct" activation sources happen to have identical times, they are treated as independent events. At runtime, the two events are treated sequentially. Two "distinct" activation sources produce asynchronous activation signals. In general, any output activation port on a TA is considered a distinct activation source. However, there are two exceptions: basic blocks with direct event input-output dependencies are the conditional blocks `IfThenElse` and `SwitchCase`.

Consider, for example, the `IfThenElse` block, which represents conditional constructs similar to the **if-else** statement in classical programming languages such as C. The `IfThenElse` block has one activation input port and two activation output ports. Depending on the value of the signal on its regular input port, the block redirects its input activations to one of its output activation ports. In this case, the output activation signal is synchronous with the input activation signal. So, the compiler does not treat the output activation ports of the `IfThenElse` block as "distinct" activation sources. In other words, the origin of the output clocks is the same, making them synchronous. The `SwitchCase` block is the counterpart of the **switch-case** statements in classical languages. Other blocks, such as `Subsample`, built on top of these two basic blocks, also provide synchronous outputs. Note that all `SampleClock` blocks, even having periods and offset values, produce synchronous activations or events.

2 Code Generation and FMU export

Code generation is utilized to create C code from a TA superblock, capturing its dynamic behavior. The generated code serves various purposes, including creating new blocks to replace the original superblock, ensuring intellectual property protection, or exporting to other simula-

tion environments.

Two distinct code generation technologies are available in TA:

1. **Standard Code Generator**: This technology closely mirrors the behavior of the TA simulator, relying on the same libraries used by the simulator, particularly the libraries containing the simulation functions of the blocks. The generated code essentially replicates the actions performed by the simulator, resulting in performance comparable to simulation. However, the generated code is not intended for inspection or direct use and has dependencies on TA libraries. Therefore, when exported, the FMUs produced using this code generation technology contain several shared libraries.

2. **Inline Code Generator**: Unlike the standard code generator, the inline code generator does not rely on TA libraries for block simulation functions. Instead, it generates and inlines a specific code based on the types and sizes of the block input and output signals. The code is customized and highly optimized using for example by constant propagation and threshold based loop rolling. As a result, the generated code is more efficient and simpler. Additionally, all memory used by the code can be statically allocated. This code generator supports both discrete-time and continuous-time dynamics and, to some extent, multiple synchronous clocks. Various targets can be selected for code generation, such as a native TA block, a Python block, or an FMU block.

Both code generators support nested FMUs, enabling the export of Modelica models. In the standard code generator, the Modelica model is converted into an FMI-2.0 for ModelExchange, while in the inline code generator, it is converted to an FMI-2.0 for CoSimulation.

2.1 Events and clocks in FMU export

In general, the TA model may contain continuous-time and discrete-time states. Continuous-time states are, in general, always active and are invoked by the numerical solver. These states may be reinitialized or experience discontinuities at event times. Events can be triggered by either a clock or an external event. Discrete-time states are usually activated by clocks or external activations.

During the code generation process, the periods and offsets of all `SampleClocks` blocks are used to compute a base frequency which is used as parameter of a unique periodic clock. This clock drives the periodic part of the model directly or through subsampling. Thus, the final generated code is activated by one clock and possibly several external activation sources. The clock and external activations execute their own tasks at activation. These tasks may or may not have intersections or common variables.

There are several clock and event types in FMI-2.0 and FMI-3.0. Most of these event and clock types are successfuly imported in Activate (Najafi and Nikoukhah 2022). In this section, we examine the inverse problem: the way events and clocks defined in TA can be exported to FMUs.

2.2 FMI export for FMI-2.0

Synchronous clocks or external clocks are not supported in FMI-2.0. FMI-2.0 for CoSimulation does not support events. In FMI-2.0 for ModeExchange, events can be either time-event, input-event, or state-event. Among these event kinds, time-event looks appropriate to be used for export of clocks used in TA. In FMI-2.0, at each time event, the time instant of the next time-event is retrieved and the FMU is called at that time instant. This is the exact counterpart of the way events are generated in TA. The events can be either periodic or aperiodic; see, for example, the model in Figure 1.

During the FMU export, the initial event time is used at the very first time the FMU enters the Event-Mode. For the later event times, the next event time is programmed by the FMU and delivered to the FMU importer. The code snippet for handling the event inside the FMU is as follows:

```
if (fabs(comp->eventInfo.nextEventTime - currentTime)<Tolerance){
  updateOutput(x,xd,ins1,outs1,outs2);
  updateState(x,xd,ins1,outs1,outs2);
  comp->eventInfo.nextEventTimeDefined = fmi2True;
  comp->clock_tick++;
  comp->eventInfo.nextEventTime = comp->start_time+
     (comp->clock_tick) *ClockPeriod;
}
```

The `Tolerance` and `ClockPeriod` values are defined by the code generator. The problem with events in FMI-2.0 lies in event classification inside the FMU. When an event occurs, the FMU must distinguish whether it occurs due to a time-event, state-event, input-event, or if no particular event has occurred and the importer has simply pushed the FMU into event mode. To determine if the event is indeed a time event, the expected time-event time instant is compared with the current time of the FMU set by the API `fmi2SetTime`. While a good importer usually sets the current time precisely at the expected time event, the FMU should consider the general case and take into account numerical round-off errors by using an error tolerance in the comparison. This error tolerance may be problematic in many cases, which is why clocks were introduced in FMI-3.0 to eliminate uncertainties.

2.3 FMI export for FMI-3.0

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 TA, such as arrays. However, despite blocks in TA having activation (clock) inputs and outputs, the semantic differences between FMI-3.0 clocks and TA activations do not allow for a simple mapping of FMI-3.0 clocks into TA activation signals.

In FMI-3.0, besides the legacy time event already available in FMI-2.0, several clock types have been introduced. There is no exact one-to-one correspondence between TA clocks and FMI-3.0 clocks. The way FMI-3.0 clocks are imported in TA has been explained in (Najafi and Nikoukhah 2022).

2.3.1 Periodic Clock: Example of Clocked Counter

The simplest and most basic clock type in TA is the `SampleClock`, which is defined by offset and period values. Consider the sample clock shown in Figure 5, which activates a `counter`. The counter increaments its output on each clock tick and once reaches five resets to zero.

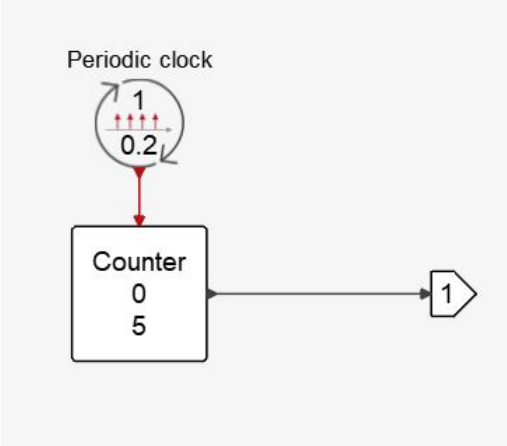


Figure 5. Clocked-counter

The sample clock is mapped to the time-based periodic clock with `intervalVariability="constant"`. The `intervalDecimal` is set to the basic period of the final clock, and the `shiftDecimal` or the clock offset is always set to zero in TA, as its value has been taken into account in computing the basic period of the clock. The clocks in TA cannot be exported with `intervalDecimal` fixed or tunable. The `clocks` attribute of the variable `Output` indicates the dependency on the clock, i.e., `"2"`.

```
<Clock name="SampleClock" causality="input"
valueReference="2"   variability="discrete"
intervalVariability="constant"
intervalDecimal="0.2" shiftDecimal="0"
description="Constant periodic input clock: 1, nevprt=1" >
</Clock>

<Int32 name="Output"  valueReference="3" variability="discrete"
clocks="2"   causality="output"   description="" >
</Int32>
```

This code snippet is generated for the above model. At the clock tick, at first, the model is evaluated, then the internal states are updated.

```
fmi3Status fmi3SetClock(fmi3Instance instance,
    const fmi3ValueReference valueReferences[],
    size_t nValueReferences,
    const fmi3Clock values[])  {
...
for (i=0;i<nValueReferences;i++)
  if (valueReferences[i]==2) {
    ...
    comp->Clk[k]=values[i];
  }
 }
...
}

fmi3Status fmi3UpdateDiscreteStates(...) {
 ...
 if (comp->Clk[0]) {
  updateOutput_clock_1 (outs1);
  updateState_clock_1 (outs1);
  ....
 }
...
}
```

The exact timing of the clock ticks is computed by the importer. At each clock time instant, the importer sets the corresponding clock and informs the FMU that the clock is enabled. So there's no place for uncertainty or error tolerance.

2.4 Triggered Input Clock: Incrementing the Counter

The next basic event source type in TA is the external activation. If a superblock is activated by an external activation, the compiler has no information about the periodicity of the events. The model part is executed when the event happens. This event type is the exact counterpart of the triggered event in FMI-3.0. Consider the model in Figure 6 where a counter is activated by an external unknown source.

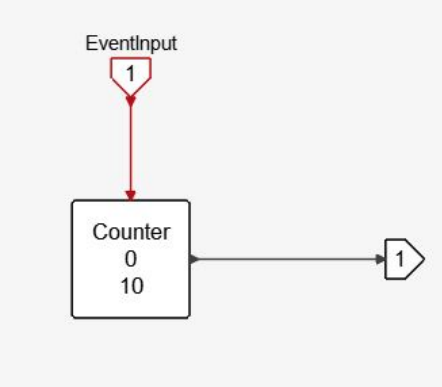


Figure 6. Triggered-counter

```
<Clock name="TriggeredClock1"  valueReference="2"
variability="discrete" intervalVariability="triggered"
causality="input"
description="External triggered input clock: 1, nevprt=1" >
</Clock>
<UInt8 name="Output"  valueReference="3"
variability="discrete"   clocks="2"
causality="output"  >
</UInt8>
```

2.5 Multiple Variable Access: Clocked Counter and Reset

In TA, several events and clocks can be used to access and update a single variable. For instance, the output of the block `Selector` in Figure 3, is updated by two input events. Another example is the `Counter with reset` block. Consider the model in Figure 7 where the

counter is incremented at activation time instants of the `SampleClock`.

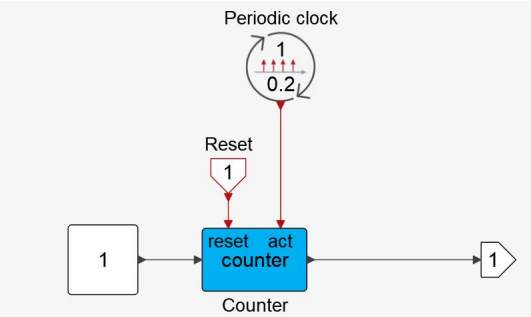


Figure 7. Counter-with-reset

Counter is reset to zero whenever the event of the external activation is fired.

reset	act	ouput
0	0	do nothing
0	1	increament by one
1	0	resent to zero
1	1	resent to zero

In this model, the counter output variable is accessed and updated by two different events.

```
<Clock name="TriggeredClock1"  valueReference="2"
variability="discrete"   intervalVariability="triggered"
causality="input"
description="External triggered input clock: 1, nevprt=1" >
</Clock>
<Clock name="SampleClock"  valueReference="3"
variability="discrete"   intervalVariability="constant"
intervalDecimal="0.2"   shiftDecimal="0"   causality="input"
description="Constant periodic input clock: 2, nevprt=2" >
</Clock>
<Float64 name="Output" valueReference="4" variability="discrete"
clocks="2 3"   causality="output"   description="" >
</Float64>
```

Note that the `clocks` attribute of the variable `Output` lists the dependency of the two clocks, i.e., `"2,3"`.

2.6 Synchronism Issue

Unlike in FMU, in TA, events can happen at the same time (simultaneous) but be asynchronous. Due to this difference, several situations should be considered to be handled. Consider, for example, a superblock having two external input events. In this case, the following table is considered to handle three possible different tasks in these situations.

Event-1	Event-2	Task
0	0	do nothing
0	1	task-1
1	0	task-2
1	1	task-3

For instance, if only Event-1 is activated, task-1 should be run. If Event-1 and Event-2 are activated synchronously, task-3 is run. This distinction between tasks is important in some situations where there is a common variable activated by two events. If no common variables are activated by both events, the execution of task-3 would have the same result as the execution of task-1 and task-2 in any order. Consider, for example, the counter in Figure 8.

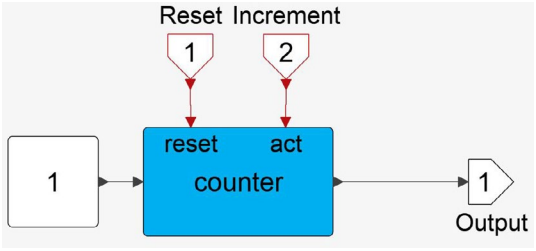


Figure 8. Counter with two-external-activations

If at a time instant both the reset and increment events are activated synchronously, in TA the output of the counter will be zero. If event ports are activated simultaneously, the order of execution is important. If the increment event input is activated after the reset event input is activated, the result will be different.

In the FMI-3.0 standard, when the FMU enters the event-mode, the importer should inform the FMU about the activated clocks by calling the API `fmi3SetClock`. With this API, the importer can enable the clocks one by one and then call the API `fmi3UpdateDiscreteStates` to execute the tasks corresponding to each clock of the FMU. The other possibility is to activate all clocks at once and then call the API `fmi3UpdateDiscreteStates`.

Actually, since there is no way in FMI-3.0 to indicate if input clocks are synchronous, *i.e.*, should tick together, the result of the simulation may be different in different importers. The only way to avoid this situation is to avoid using variables activated by different clocks. In this case, the order of execution does not matter. But this becomes a limitation for exporting a tool independent FMU.

2.7 Periodic Input Clock Connections

In TA, every clock source, dependent or independent, defines the information flow toward other input clock ports. No clock source can be connected to other clock sources. If the union of two clock sources is needed, a `Union` block can be used. For example, in Figure 9, the counter is incremented whenever each of the clock sources ticks.

In FMI-3.0, the `causality` attribute of periodic clocks is `"input"`, which should be interpreted as if the clock source is coming from the importer. This makes an open gate for arbitrary interpretation from FMU importers. For example, what should happen if two periodic clocks from two FMUs are connected together and connected to the triggered input clock of another FMU. In TA, this connection raises an error, but other tools may interpret it as the union (OR operation) or the intersection

 10.3384/ECP20791 DOI

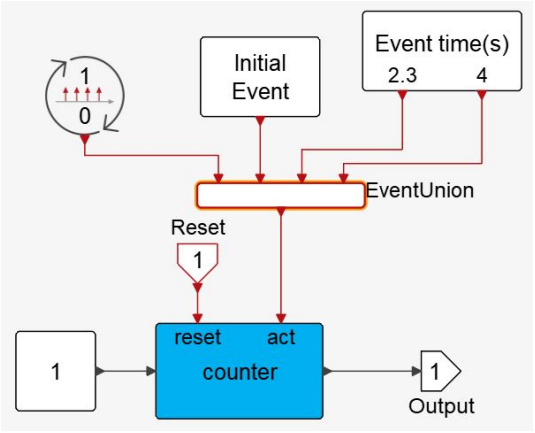


Figure 9. Example of clock unions (**EventUnion**)

(AND operation) of periodic clock ticks. This lack of definition would result in the FMU export of a model with such a connection being tool-dependent.

2.8 FMU for Cosimulation: Solver inlining

The way the clock is handled in FMI-3.0 is independent of the FMU implementation, *i.e.*, the FMU can be either Model-Exchange or Co-Simulation[3]. The FMI-3.0 FMUs work almost identically for both FMU implementations in handling clocks. The difference between the two implementations is in handling of the continuous-time dynamics. The introduction of clocks in FMI-3.0 has offered TA the opportunity of exporting the internal dynamics in a new way, *i.e.*, solver inlining.

In the FMU export for Co-Simulation, with the inline code generation, a variable-step or fixed-step solver is chosen to be used to simulate the continuous-time part of the model. Besides the classical solver linking, e.g., linking an Euler or RK4 solver, TA supports solver inlining which is a transformation method for embedding a numerical solver within the generated code. This transformation can be applied to a general model or part of it, to turn it into a purely discrete-time synchronous model with the resulting discrete-time (super) block behavior matching as closely as possible that of the original super block.

The model transformation for solver inlining, done during the model compilation process, is achieved by embedding a fixed-step numerical solver for discretizing the dynamics of the continuous-time components of the system. The exported model can be considered both as a pure discrete-time block and a Co-simulation component. The main usage of the solver inlining is for models exported by the inline code generator for embedded applications.

The basic idea behind the embedded solver is the conversion of the differential equations associated with the dynamics of blocks with internal continuous-time states to time difference equations. Difference equations are, in turn, can be implemented by discrete-time blocks running on a single base clock. This, however, does not work for variable step-solvers. The construction of the discrete-time version of the model in that case requires complex transformations. To see this difference, consider the following simple system, which can be implemented in TA by two blocks: an integrator block and a memoryless block realizing the function f.

$$y' = f(y)$$

The Euler solver uses a first order discrete approximation of this system:

$$y_{k+1} = y_k + h.f(y_k)$$

where

$$y_k = y(t_k)$$

and

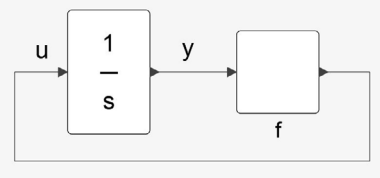
$$t_{k+1} = t_k + h$$

. The time instances when the state is updated correspond to a fixed frequency sampling of time t, with period h. The differential equation in this case is trivially translated into a difference equation. The system can be represented as a block diagram by separating the system into an integrator block and a memoryless block by noting that it can be rewritten as follows

$$y' = u$$

$$u = f(y)$$

The corresponding model can be constructed as follows.



In this case, the Euler discretization yields

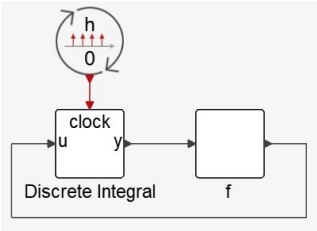
$$y_{k+1} = y_k + h.u_k$$

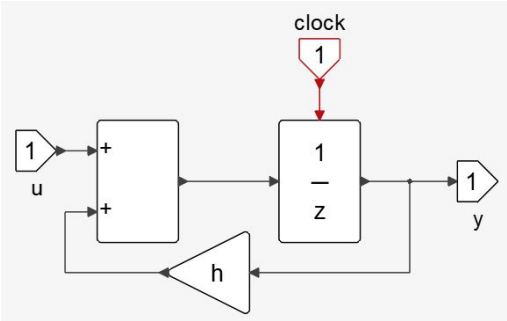
$$u_k = f(y_k)$$

So, the discrete-time version of the model is obtained by simply replacing the integrator block by a discrete block (Discrete Integral super block). The content of the Discrete Integral superblock is also shown.

The new model which is activated by a `SampleClock` block with period h is a purely discrete-time model. The original model is simply transformed by replacing the integrator with the Discrete Integral block and a `SampleClock` block. In a more general model with multiple integrator blocks, each integrator block can be replaced by its discrete-time equivalent, activated the `SampleClock` block. The stateless blocks of the model,

[3]The Scheduled Execution FMU type has not been considered in this paper





represented here by the f block, are not modified. For higher order approximations of the derivative however, the computations at each time step cannot be realized by single activations of discrete blocks. To see this, consider a fourth order Runge-Kutta (RK4) solver algorithm for the same system:

$$y_{k+1} = y_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = h.f(y_k)$$

$$k_2 = h.f(y_k + k_1/2)$$

$$k_3 = h.f(y_k + k_2/2)$$

$$k_4 = h.f(y_k + k_3)$$

The computation of the of next discrete state y requires four evaluations of the function f with different arguments. To embed this solver in the continuous-time model to obtain a discrete-time model, the RK4 solver equations can be implemented as a more complex Discrete Integral superblock as shown in Figure 10.

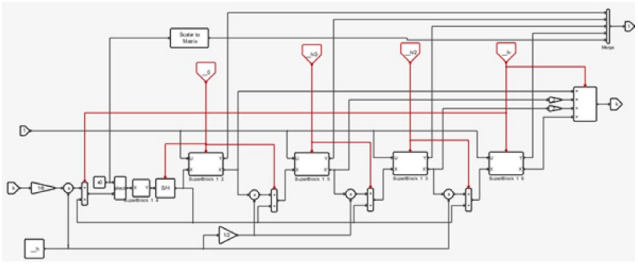


Figure 10. Discreteization of an integrator block using RK4 solver

Once the transformation is done, a C code is generated for the purely discrete model activated by the sample clock with period h.

The generated C code, which is activated by a periodic clock with a known constant period, can naturally be exported to an FMU. For FMI-2.0, a periodic regular time-event with the time interval equal to h will be used in the FMU. For FMI-3.0, the export is more natural; an input clock with the `intervalVariability="constant"` and `intervalDecimal` equal to h will be used.

The embedded solver transformation converts the continuous-time dynamics part of the model to discrete-time. In other words, the FMU has no continuous-time dynamics and the whole dynamics has been discretized and activated by a `periodic clock`. As a result, disregarding the discretization error, the FMU can be exported in the same way for both ModelExchange and CoSimulation in FMI-3.0.

3 Conclusion

The introduction of clocks in FMI-3.0 has provided the possibility of exporting more general models with continuous-time and discrete-time dynamics, particularly from TA. This paper has explored the integration of clocks within the context of **Altair Twin Activate** for FMU exports. Different clock and activation types are considered and the way they are exported to FMI-3.0 has been presented. The introduction of periodic clocks in FMI-3.0 has allowed the inlining of the numerical solver within the FMU, making it possible to achieve identical discrete dynamics in FMU export for both ModelExchange and CoSimulation.

References

Gomes, Claudio et al. (2021). "The FMI 3.0 Standard Interface for Clocked and Scheduled Simulations". In: *Proceedings of the 14th International Modelica Conference.*

INRIA (n.d.). URL: http://www.scicos.org.

Junghanns, Andreas et al. (2021). "The FMI 3.0 Standard Interface for Clocked and Scheduled Simulations." In: *Proceedings of the 14th International Modelica Conference.*

Modelica Association, FMI Website (2022). URL: https://fmi-standard.org.

Najafi, Masoud and Ramine Nikoukhah (2022). "Importing FMU-3.0: challenges in proper handling of clocks". In: *Proceedings of Asian Modelica Conference 2022, Tokyo, Japan.*

Nikoukhah, Ramine and Sebastien Furic (2009). "Towards a full integration of Modelica models in the Scicos environment". In: *Proceedings of the 7th International Modelica Conference.*

Nikoukhah, Ramine, Masoud Najafi, and Fady Nassif (2017). "A Simulation Environment for Efficiently Mixing Signal Blocks and Modelica Components". In: *Proceedings of the 12th International Modelica Conference.*

Specification, FMI-3.0 (2022). URL: https://fmi-standard.org/docs/3.0.

Event support for simulation and sensitivity analysis in CasADi for use with Modelica and FMI

Joel Andersson[1] James Goppert[2]

[1]Freelance software developer and consultant, USA, joel@jaeandersson.com
[2]School of Aeronautics and Astronautics, Purdue University, USA, jgoppert@purdue.edu

Abstract

CasADi is an open-source framework that can be used to efficiently solve optimization problems involving user-defined ODE/DAE models. Supported solution methods include so-called shooting methods, where solvers for initial-value problems in ODEs or DAEs are referenced inside in nonlinear programming (NLP) formulations. In order to solve such NLP formulations with gradient-based algorithms, CasADi implements a fully automatic sensitivity analysis. This analysis includes forward sensitivity analysis, adjoint sensitivity analysis as well as the calculation of higher-order sensitivities for the ODE/DAE models. Because of the variational (differentiate-then-integrate) approach used, the numerical solution can be performed with variable-step size, variable-order integrators such as those from the SUNDIALS suite.

In this work, we present a generalization of the sensitivity analysis support in CasADi to systems with events, as are common in real-world cyber-physical models. In particular, the event extension enables us to formulate and solve optimization problems with such event systems, without a priori knowledge of the number and ordering of events. Ultimately, we expect the proposed approach to be compatible with general cyber-physical models formulated in Modelica or available as model-exchange FMUs.

We demonstrate the proposed approach for two proof-of-concept examples; the classical bouncing ball written in CasADi directly and a simple hybrid DAE describing a breaking spring formulated in Modelica and imported symbolically into CasADi. In the examples, we show that the forward sensitivities calculated to high precision using the proposed approach are consistent with a cruder finite-difference approximation and provide an example of how they can be embedded into optimization formulations. We discuss how the approach can be extended to handle standard FMUs, adhering to FMI 2 or FMI 3, as well as non-trivial Modelica models imported via a symbolic interface based on the emerging Base Modelica standard.

Keywords: Hybrid DAEs, sensitivity analysis, CasADi, Modelica, FMI

1 Introduction

Dynamic models describing cyber-physical systems often include events that are triggered when some conditions are met. These events can arise both from the need to faithfully capture the physics, e.g. an object transitioning from being stationary to starting to slide, or to capture the modes in control systems. Physical modeling environments, such as those based on Modelica, allow events to be efficiently described and transformed into a canonical form compatible with numerical solvers. For Modelica, the corresponding form is a *hybrid differential-algebraic equation* (DAE) as described by the language specification (Modelica Association 2021). For a hybrid DAE in a standard form, events are generally triggered by zero-crossing conditions for a set of event indicators, which are evaluated along with the DAE. Certain numerical solvers such as those from the SUNDIALS suite (Hindmarsh et al. 2005) used in this work, are able to monitor the event indicators for zero-crossings and stop the integration prematurely if an event is detected. At the detected event, the system is then updated according to the finite state machine semantics described in the hybrid DAE representation before the DAE integration is resumed.

1.1 Events in dynamic optimization

The handling of events using zero-crossing events and event transitions is the standard approach for hybrid DAE simulation. For dynamic optimization problems, i.e. optimization problems where the hybrid DAE enters as constraints in the formulation, the standard approach is instead to partition the time horizon into multiple *stages* (or *phases*) with events happening between the different stages but not within them. The time durations between events then become additional decision variables of the optimization problem. To illustrate, if we have a single event at (a priori unknown) time T, the physical time variable t is substituted in the first stage with a dimensionless time τ according to $t = T\,\tau$. We can then proceed to solve the optimization problem as if the event times were known with stage durations added as an additional optimization variables. While this approach has proven useful in numerous applications, it is not as general as the hybrid DAE representation used for simulation. In particular, it requires a priori knowledge of the number of stages, which is often not available.

Using the approach proposed here, this reformulation to a multi-stage problem, with associated restrictions, is no longer needed. Instead, we are able to embed the hybrid

DAEs directly in the dynamic optimization formulations and still get the exact first and second order sensitivity information needed by gradient-based numerical optimization methods.

1.2 CasADi

CasADi (J. A. E. Andersson et al. 2019) is an open-source software package for C++, Python, MATLAB and Octave. It offers versatile environent that in particular can be used to solve a range of different numerical optimization problems, using different methods and solvers. In particular, CasADi can be used to efficiently solve numerical optimal control problems, i.e. optimization problem constrained by differential equations. At the core of CasADi is a symbolic framework implementing *algorithmic differentiation* (AD) in both forward and reverse (adjoint) modes. In addition to AD, symbolic expressions can be used for efficient evaluation, either in virtual machines or in generated, self-contained C code. Importantly, the symbolic expressions can embed calls to user-defined, differentiable *function objects*. Such function objects can be defined in number of different ways, including from other symbolic expressions, by linear or nonlinear systems of equations or user defined code. In (Joel Andersson 2023), it was shown how differentiable CasADi function objects could be created from *functional mock-up units* (FMUs) adhering to the *functional mock-up interface* (FMI) standard. In this work, we present an extention of another imporant type of function objects in CasADi, *Integrator* instances, which are used to simulate and perform sensitivity analysis for differential equations. A relatively comprehensive and up-to-date description of this functionality is presented in Section 2. In Section 3, we will show how the integrators were extended to support general events handling, while still retaining efficient and accurate differentiability.

1.3 Related work

Sensitivity analysis and numerical optimization for hybrid dynamic systems have been performed previously, in particular in the Julia environment, using integrators formulated in the *DifferentialEquations.jl* package (Rackauckas and Nie 2017). For models available as expressions, derivatives can be calculated analytically by differentiating the entire algorithm, giving a integrate-then-differentiate approach. It is our understanding that this approach, unlike the variational approach presented here, can not be readily used with models formulated in Modelica or provided as FMUs. We refer to (Corner, C. Sandu, and A. Sandu 2019) for a recent overview of methods for hybrid sensitivity analysis.

2 Simulation and sensitivity analysis in CasADi

CasADi can be used to solve initial value problems (IVP) in ordinary differential equations (ODEs) or differential-algebraic equations (DAEs) with fully automatic sensitivity analysis. This support, which has existed since early versions of CasADi, has been extended and improved over the years. In the following, we provide a description of the current algorithm, which largely corresponds to the refactoring of the functionality which enabled the use of FMI models, as described in (Joel Andersson 2023). In Section 3, we will show how this formulation can be extended to support events, while still maintaining efficient, analytic differentiability.

The dynamic systems considered, as of CasADi 3.6, are semi-explicit DAEs with quadratures:

$$\begin{cases} \dot{x}(t) = f_{\text{ode}}(t, x(t), z(t), p, u(t)) \\ 0 = f_{\text{alg}}(t, x(t), z(t), p, u(t)) \\ \dot{q}(t) = f_{\text{quad}}(t, x(t), z(t), p, u(t)), \end{cases} \tag{1}$$

where $t \in \mathbb{R}$ is time (or some other independent variable), $x(\cdot) \in \mathbb{R}^{n_x}$ is a state vector, $z(\cdot) \in \mathbb{R}^{n_z}$ is a vector of algebraic variables, $q(\cdot) \in \mathbb{R}^{n_q}$ is a state vector that does not appear in the right-hand-side, $p \in \mathbb{R}^{n_p}$ is a (tunable) parameter and $u(\cdot) \in \mathbb{R}^{n_u}$ is a control input, which is assumed to be piecewise constant. If piecewise constant control inputs are too restrictive for a particular application, piecewise polynomial approximations can be handled by adding additional state variables (e.g. defined by $\dot{x}_{\text{piecewise linear}} = u_{\text{piecewise constant}}$). The quadrature states in this context are especially important for calculating integral terms but are also used in adjoint sensitivity analysis.

We assume that any DAE is of index-1, i.e. in particular that the Jacobian of f_{alg} with respect to z exists and is invertible. For ODEs, z and $f_{\text{alg}}(\cdot)$ have dimension zero. If the index-1 assumption does not hold, an index reduction should be performed prior to simulation, which has been implemented both in CasADi natively (for models given as symbolic expressions) and in coupled modeling environents, such as those based on Modelica.

To solve IVPs, the CasADi user creates *Integrator* instances. These are formed from a given the initial time t_0, an output time grid $[t_1, \ldots, t_N]$ as well as symbolic expressions of the form (1), or more generally, a (differentiable) CasADi function object that calculates f_{ode}, f_{alg} and f_{quad} from given values for t, x, z, p and u:

$$\begin{aligned} &f : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_q} \\ &(t, x, z, p, u) \mapsto (f_{\text{ode}}, f_{\text{alg}}, f_{\text{quad}}) \end{aligned} \tag{2}$$

For the typical usage, an Integrator instance is a (differentiable) CasADi function object that given $x(t_0)$, p, the $u(t)$ trajectory and a *guess* for $z(t_0)$, calculates $x(t_k)$, $z(t_k)$ and $q(t_k)$ at all output times, $k = 1, \ldots, N$. If the DAE has quadratures, $q(t_0)$ is assumed zero. We can write the function object defined by the integrator instance as follows:

$$\begin{aligned} &F : \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_u \times N} \to \mathbb{R}^{n_x \times N} \times \mathbb{R}^{n_z \times N} \times \mathbb{R}^{n_q \times N} \\ &(x_0, z_0, p, u) \mapsto (x, z, q) \end{aligned} \tag{3}$$

Solving the initial value problem

The actual calculation of (3) takes places in the CasADi Integrator class, which relies on successive calls to one of the solver *plugins*. The top level solver algorithm is illustrated in Algorithm 1, where the functions `RESET` and `ADVANCE` are implemented in the specific solver plugin. These two functions correspond to initializing the (forward) integration at some given time, providing the necessary data, and advancing the solution to some given time point, respectively. Algorithm 1 also includes the helper function `NEXT_STEP` which checks the provided control input and determine when the next step change in the control occurs. If there are no more input step changes, the end of the simulation (N) is returned. The algorithm will ensure that the integration is stopped at such input change times. The stopping times are also used to prevent a solver plugin from taking *internal* time steps past the stopping times during the `ADVANCE` step (omitted in Algorithm 1 for simplicity).

Algorithm 1 Integration in CasADi without sensitivity analysis or events handling

procedure SIM($x_0 \in \mathbb{R}^{n_x}$, $z_0 \in \mathbb{R}^{n_z}$, $p \in \mathbb{R}^{n_p}$, $u_k \in \mathbb{R}^{n_u}$, $k = 0,\ldots,N-1$)
  $k_{\text{step}} := 0$ ▷ Index of the next input step change
  for $k = 0,\ldots,N-1$ **do** ▷ Forward integration
    if $k = k_{\text{step}}$ **then** ▷ Input step change
      $k_{\text{step}} :=$ `NEXT_STEP`$(k, u_{k+1}, \ldots, u_{N-1})$
      `RESET`(t_k, x_k, z_k, p, u_k)
    end if
    $(x_{k+1}, z_{k+1}, q_{k+1}) :=$ `ADVANCE`(t_{k+1})
  end for
  return $x_k \in \mathbb{R}^{n_x}$, $z_k \in \mathbb{R}^{n_z}$, $q_k \in \mathbb{R}^{n_q}$, $k = 1,\ldots,N$
end procedure

As of this writing, there were four solver plugins available; two CasADi native fixed-step integrators implementing explicit and implicit Runge-Kutta, respectively, as well as interfaces to the SUNDIALS solvers CVODES and IDAS (Hindmarsh et al. 2005). For latter two solvers, the default algorithm is a variable-order variable-step size backward differentiation formula (BDF) method that takes successive steps *past* the given output time and then evaluates the polynomial representation available for the last integrator step at the given output time. All the interfaced solvers rely on CasADi to automatically generate any derivative information needed, including sparse Jacobians, and use sparse linear algebra for the step size computation.

The remainder of this section details how Algorithm 1 is extended internally in CasADi to be able to efficiently calculate forward and adjoint sensitivities and requires some familiarity with algorithmic differentiation. A reader mainly interested in using the framework in applications may choose to skip these parts as they are not essential for using the code.

Forward sensitivity analysis

The CasADi integrators support analytic forward sensitivity analysis via a variational approach (J. Andersson 2013; J. A. E. Andersson et al. 2019), i.e. an augmented set of DAEs are formed corresponding to the forward sensitivity equations. The forward sensitivity analysis is implemented both symbolically and numerically. In the symbolic implementation, which is the older implementation, a new DAE for the augmented system is created which is solved as any other DAE, exploiting only the sparsity of the augmented DAE system. This symbolic differentiation can be done repeatedly, to get analytic derivatives to any order, assuming sufficiently smooth DAEs.

To better exploit the specific structure of the forward sensitivity equations, a numeric implementation of forward sensitivity analysis was added (Joel Andersson 2023). The numeric implementation is implementated by supporting multiple columns in (3), corresponding to different forward seeds/sensitivities, i.e. perturbations with respect to different combinations of inputs in Algorithm 1. For N_f forward sensitivities that are calculated along with the original (undifferentiated) trajectory, the generalized definition of F can be written:

$$\begin{aligned}&\tilde{F} : \mathbb{R}^{n_x\times(1+N_f)} \times \mathbb{R}^{n_z\times(1+N_f)} \times \mathbb{R}^{n_p\times(1+N_f)} \times \mathbb{R}^{n_u\times(1+N_f)N} \\ &\to \mathbb{R}^{n_x\times(1+N_f)N} \times \mathbb{R}^{n_z\times(1+N_f)N} \times \mathbb{R}^{n_q\times(1+N_f)N} \\ &(x_0, z_0, p, u) \mapsto (x, z, q)\end{aligned} \tag{4}$$

Note that the multiple right-hand-sides are usually hidden from the user, who typically embeds the undifferentiated F from (3) in some optimization formulation, and the sensitivity equations are generated automatically to provide a gradient-based optimizer with the required derivative information.

Algorithm 1 continues to be valid when forward sensitivity equations are included in the calculation, with the only change that calculation of x_k, z_k and q_k is now done with $(1+N_f)$ columns at a time instead of one column at a time. It is up to the solver interfaces, i.e. the implenentation of `RESET` and `ADVANCE` to exploit the sensitivity structure. In the SUNDIALS interfaces, this exploitation is done by providing SUNDIALS with structure-exploiting linear algebra routines. These linear algebra routines use second order derivative information – calculated via forward-over-forward algorithmic differentiation of the DAE function – to exactly and efficiently solve the augmented linear system. Note that we do not use SUNDIALS native forward sensitivity support.

Adjoint sensitivity analysis

Similar to forward sensitivity analysis, the CasADi integrators support adjoint sensitivity analysis via a variational approach (J. Andersson 2013; J. A. E. Andersson et al. 2019). These equations define a terminal-value problem coupled to the regular forward integration. Because the coupling of the terminal-value problem to the initial value problem is in one direction only, the combined prob-

lem can be solved with a forward integration, recording the integrator steps, followed by a backward integration.

The original implementation of adjoint sensitivity analysis in CasADi supported a general backward differential equation, as long as it was affine in the "backward states", and was implemented symbolically. Because of the specific structure, the integrator could be differentiated repeatedly, giving analytical sensitivities to any order, noting that adjoint-over-adjoint sensitivities can be reformulated as forward-over-adjoint sensitivities.

In CasADi 3.6, a restriction of the formulation was imposed, requiring that the terminal value problem to always be the adjoint sensitivity equations corresponding to the forward integration. The adjoint equations may in turn have forward sensitivity equations, which is important to be able to efficiently calculate second order derivative information, e.g. for numerical optimization. With N_a adjoint sensitivities and N_f forward sensitivities, (4) is further generalized as follows:

$$\begin{aligned}\hat{F} : &\mathbb{R}^{n_x\times(1+N_f)} \times \mathbb{R}^{n_z\times(1+N_f)} \times \mathbb{R}^{n_p\times(1+N_f)} \times \mathbb{R}^{n_u\times(1+N_f)N} \\ &\times \mathbb{R}^{n_x\times(1+N_f)N_aN} \times \mathbb{R}^{n_z\times(1+N_f)N_aN} \times \mathbb{R}^{n_q\times(1+N_f)N_aN} \\ \to &\mathbb{R}^{n_x\times(1+N_f)N} \times \mathbb{R}^{n_z\times(1+N_f)N} \times \mathbb{R}^{n_q\times(1+N_f)N} \\ &\mathbb{R}^{n_x\times(1+N_f)N_a} \times \mathbb{R}^{n_z\times(1+N_f)N_a} \\ &\times \mathbb{R}^{n_p\times(1+N_f)N_a} \times \mathbb{R}^{n_u\times(1+N_f)N_aN} \\ &(x_0, z_0, p, u, \lambda_x, \lambda_z, \lambda_q) \mapsto (x, z, q, \lambda_{x_0}, \lambda_{z_0}, \lambda_p, \lambda_u),\end{aligned} \tag{5}$$

where $\lambda_x, \lambda_z, \lambda_q$ correpond to *adjoint (and forward-over-adjoint) seeds* and $\lambda_{x_0}, \lambda_{z_0}, \lambda_p, \lambda_u$ correspond to *adjoint (and forward-over-adjoint) sensitivities*. Note that since z_0 is a *guess*, λ_{z_0} is going to be trivially zero, but is kept in the function signature to get a consistent function signature (that can easily be embedded into symbolic expressions). The function signature (5), which is the most complex of any of the CasADi core classes, remains the same with the addition of event support, which we will present in Section 3.

In Algorithm 2 we show the generalization of Algorithm 1 to handle forward and adjoint sensitivities, which in addition to `RESET` and `ADVANCE` mentioned earlier also includes two additional methods, `IMPULSE` to provide an additive contribution to the adjoint states at a given time and `RETREAT` to integrate the system backwards to a given time point. `NEXT_IMPULSE` is a helper function, similar to `NEXT_STEP` to find the next output time where an `IMPULSE` call is needed. Note that whenever there is a step change in a control input, the forward integration is repeated starting at the beginning of the previous step change (or initial time t_0).

As in the case of forward sensitivity analysis, the addition of numerical adjoint (and forward-over-adjoint) sensitivity analysis in CasADi 3.6 enabled significantly better structure exploitation in the integrator interfaces, specifically in the SUNDIALS interfaces. In particular, it allowed an arbitrary number of forward, adjoint and forward-over-adjoint sensitivities to be calculated along with the original simulation without increasing the size of the linear system needing to be factorized inside the interfaced ODE/DAE integrators. Similar to the forward sensitivity analysis, the forward-over-adjoint sensitivity analysis uses a matrix-free second order correction, implemented via forward-over-adjoint directional derivatives to exactly solve the augmented linear system.

Algorithm 2 Integration in CasADi with forward and adjoint sensitivity analysis but without events handling

procedure SIM_S($x_0 \in \mathbb{R}^{n_x\times(1+N_f)}$, $z_0 \in \mathbb{R}^{n_z\times(1+N_f)}$, $p \in \mathbb{R}^{n_p\times(1+N_f)}$, $u_\bullet \in \mathbb{R}^{n_u\times(1+N_f)}$, $\lambda_{x_\bullet} \in \mathbb{R}^{n_x\times(1+N_f)N_a}$, $\lambda_{z_\bullet} \in \mathbb{R}^{n_z\times(1+N_f)N_a}$, $\lambda_{q_\bullet} \in \mathbb{R}^{n_q\times(1+N_f)N_a}$)
  $k_{\text{step}} := 0$ ▷ Index of the next input step change
  for $k = 0, \ldots, N-1$ **do** ▷ Forward integration
    if $k = k_{\text{step}}$ **then** ▷ Input step change
      $k_{\text{prev}} := k$ ▷ Also keep track of old k_{prev}
      $k_{\text{step}} := \texttt{NEXT\_STEP}(k, u_\bullet)$
      $\texttt{RESET}(t_k, x_k, z_k, p, u_k)$
    end if
    $(x_{k+1}, z_{k+1}, q_{k+1}) := \texttt{ADVANCE}(t_{k+1})$
  end for
  $\lambda p := 0, \quad \lambda_{u_\bullet} := 0$ ▷ Initialize to zero
  for $k = N-1, \ldots, 0$ **do** ▷ Backward integration
    if $k < k_{\text{prev}}$ **then**
      $k_{\text{prev}} :=$ <retrieve saved value>
      $\texttt{RESET}(t_{k_{\text{prev}}}, x_{k_{\text{prev}}}, z_{k_{\text{prev}}}, p, u_k)$
      $\texttt{ADVANCE}(t_{k+1})$
    end if
    if $k < k_{\text{step}}$ **then**
      $\texttt{IMPULSE}(\lambda_{x_{k+1}}, \lambda_{z_{k+1}}, \lambda_{q_{k+1}})$
      $k_{\text{step}} := \texttt{NEXT\_IMPULSE}(k, \lambda_{x_\bullet}, \lambda_{z_\bullet}, \lambda_{q_\bullet})$
    end if
    $[\tilde{\lambda}_x, \tilde{\lambda}_p, \tilde{\lambda}_u] := \texttt{RETREAT}(t_k)$
    $\lambda_p := \lambda_p + \tilde{\lambda}_p, \quad \lambda_{u_k} := \lambda_{u_k} + \tilde{\lambda}_u$
  end for
  $\lambda_{x_0} := \tilde{\lambda}_x$
  $\lambda_{z_0} := 0$
  return $x_\bullet \in \mathbb{R}^{n_x\times(1+N_f)}$, $z_\bullet \in \mathbb{R}^{n_z\times(1+N_f)}$, $q_\bullet \in \mathbb{R}^{n_q\times(1+N_f)}$, $\lambda_{x_0} \in \mathbb{R}^{n_x\times(1+N_f)N_a}$, $\lambda_{z_0} \in \mathbb{R}^{n_z\times(1+N_f)N_a}$, $\lambda_p \in \mathbb{R}^{n_p\times(1+N_f)N_a}$, $\lambda_{u_\bullet} \in \mathbb{R}^{n_u\times(1+N_f)N_a}$
end procedure

3 Event support in CasADi

In order to implement event support in the CasADi integrators, we add a zero-crossing output to the DAE function (2) resulting in the generalized formulation:

$$\begin{aligned}f : &\mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_q} \times \mathbb{R}^{n_e} \\ &(t, x, z, p, u) \mapsto (f_{\text{ode}}, f_{\text{alg}}, f_{\text{quad}}, f_{\text{zero}})\end{aligned} \tag{6}$$

The zero-crossing component calculates n_e separate *smooth* trajectories which are monitored for zero crossings, as of this writing only from strictly negative to strictly positive values (this restriction may be removed in the future). The smoothness property is essential, and will

be used for finding the exact event time as described as in Section 3.2 below. Furthermore, the smoothness property is necessary to properly calculate forward and adjoint sensitivities as described in Section 3.3 and Section 3.4, respectively.

When a zero crossing occurs, an optional *reinit* function is called. This is a separate user-provided function which has the signature:

$$\begin{aligned} &E : \mathbb{I} \times \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \\ &(j, t, x^-, z^-, p, u) \mapsto (x, z) \end{aligned} \tag{7}$$

where x^- and z^- are the values of x and z immediately before the event, i.e. $x^-(t) = \lim_{\tau \to t} x(\tau)$ and $z^-(t) = \lim_{\tau \to t} z(\tau)$, respectively. In other words, the function E explicitly defines a the new state vector and a new guess for the algebraic variables. If a reinit function is not provided, the identity mapping is assumed.

A differentiable function with the signature (7) can be created in various ways in CasADi. In particular, we may want to create n_e different functions of the form:

$$\begin{aligned} &E_j : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \\ &(t, x^-, z^-, p, u) \mapsto (x, z), \qquad j = 0, \ldots, n_e - 1, \end{aligned} \tag{8}$$

and then use a *Switch* function in CasADi to combine them into a single function with the signature of (7). Also note that we can use an implicit definition of E or E_j e.g. by using a *Rootfinder* function in CasADi.

With the addition of the zero-crossing output in the DAE function and the new reinit function, the DAE formulation (1) becomes generalized as follows:

$$\begin{cases} (x(t), z(t)) = E(j, t, x^-(t), z^-(t), p, u(t)) \\ \qquad \text{if } \exists j : f_{\text{zero}}^{(j)}(t, x^-(t), z^-(t), p, u(t)) = 0 \\ \begin{cases} \dot{x}(t) = f_{\text{ode}}(t, x(t), z(t), p, u(t)) \\ 0 = f_{\text{alg}}(t, x(t), z(t), p, u(t)) \\ \dot{q}(t) = f_{\text{quad}}(t, x(t), z(t), p, u(t)) \end{cases} \\ \qquad \text{otherwise} \end{cases} \tag{9}$$

3.1 Generalized simulation algorithm

In Algorithm 3 we show the generalization of Algorithm 2 to also include event handling as described above. During the forward integration, the main generalization comes from allowing the `ADVANCE` step to terminate before the desired output time, in which case it will return the corresponding time and the index of the triggered root-zero crossing component. When this happens, a reinit function called `REINIT` in the algorithm is called. For simulation without sensitivities, the `REINIT` function is essentially a call to E from (7). We will show in Section 3.3 below how this function generalizes to forward sensitivity analysis. Following an event, the solver plugin needs to be reset, similarly as for the case of changing inputs. To simplify the presentation, we assume that `REINIT` returns the actual algebraic variable $\tilde{z}$. In the actual implementation, `REINIT` will just return a *guess* for the algebraic state and the actual values will be calculated during the algorithm to find consistent initial conditions inside the following `RESET`. For each event, we record x and z both before and after the event transition, along with information such as the zero crossing index and time. This will be used for the backward integration.

For the backward integration, two generalizations are necessary. Firstly, before the call to progress the backwards integration to the beginning of the interval (t_k), there is a for-loop to first visit all events that were recorded for the specific interval, in reverse order. After the adjoint integration has progressed to a specific event, the adjoint of the event transition function is called. This function is discussed in Section 3.4. Following the event, during the backward integration, we need to redo the forward integration starting at the previous event *or* input step (whichever is encountered first), denoted by the `PREVIOUS_EVENT` helper function.

3.2 Event detection algorithm

In order to determine the time of zero crossing event with high precision, the current algorithm is based on linearizing the zero-crossing algorithm in the time direction. Note that we currently do not use the zero-crossing detection capabilities of the interfaced solvers, although we may switch to doing so in a future version of the code, as discussed in Section 6.3.

Consider the zero-crossing function as a function of t, including the indirect dependencies via x, z and u:

$$e(t) = f_{\text{zero}}(t, x(t), z(t), u(t)) \tag{10}$$

We can linearize this function with respect to time as follows, assuming known values for $\dot{x}(t)$ and $\dot{z}(t)$:

$$\begin{aligned} \dot{e}(t) = &\tfrac{\partial f_{\text{zero}}}{\partial t}(t, x(t), z(t), p, u(t)) \\ &+ \tfrac{\partial f_{\text{zero}}}{\partial x}(t, x(t), z(t), p, u(t))\, \dot{x}(t) \\ &+ \tfrac{\partial f_{\text{zero}}}{\partial z}(t, x(t), z(t), p, u(t))\, \dot{z}(t), \end{aligned} \tag{11}$$

which can be efficiently calculated using a forward directional derivative of f_{zero}. Note that there are no partial derivatives w.r.t. p and u as these are constant during the interval. As of this writing, we obtain $\dot{x}(t)$ from evaluating the ODE right-hand-side, i.e. f_{ode} in (9) and did not consider zero crossing functions depending on algebraic variables. In a future iteration, we expect to obtain $\dot{x}(t)$ and $\dot{z}(t)$ from the specific integrator interface, e.g. by linearizing the DAE equations with respect to time or by evaluating an exiting polynomial representation of the $x(t)$ and $z(t)$ trajectories.

The event detection algorithm used consists of three parts:

- At the beginning of the (now generalized) `ADVANCE` function, we predict using linear extrapolation whether a zero-crossing event is expected before the given output time. If this is the case, the forward integration will be done only to this time and not to

Algorithm 3 CasADi integration, with events handling

procedure SIM_E($x_0 \in \mathbb{R}^{n_x \times (1+N_f)}$, $z_0 \in \mathbb{R}^{n_z \times (1+N_f)}$, $p \in \mathbb{R}^{n_p \times (1+N_f)}$, $u_\bullet \in \mathbb{R}^{n_u \times (1+N_f)}$, $\lambda_{x_\bullet} \in \mathbb{R}^{n_x \times (1+N_f) N_a}$, $\lambda_{z_\bullet} \in \mathbb{R}^{n_z \times (1+N_f) N_a}$, $\lambda_{q_\bullet} \in \mathbb{R}^{n_q \times (1+N_f) N_a}$)
  $k_{\text{step}} := 0$ ▷ Index of the next input step change
  $t := t_0, \quad i := 0$ ▷ Current time, event index
  for $k = 0, \ldots, N-1$ **do** ▷ Forward integration
    if $k = k_{\text{step}}$ **then** ▷ Input step change
      $k_{\text{prev}} := k$ ▷ Also keep track of old k_{prev}
      $k_{\text{step}} := \texttt{NEXT\_STEP}(k, u_\bullet)$
      $\texttt{RESET}(t_k, x_k, z_k, p, u_k)$
    end if
    while $t < t_{k+1}$ **do** ▷ Integrate until t_{k+1}
      $(\tilde{x}, \tilde{z}, \tilde{q}, t, j) := \texttt{ADVANCE}(t_{k+1})$
      while $j \geq 0$ **do** ▷ Event transition(s)
        Save $\tilde{x}$, $\tilde{z}$ (pre-call), t, j for event i
        $(\tilde{x}, \tilde{z}) := \texttt{REINIT}(j, t, \tilde{x}, \tilde{z}, p, u_k)$
        $\texttt{RESET}(t, \tilde{x}, \tilde{z}, p, u_k)$
        Save $\tilde{x}$, $\tilde{z}$ (post-call) for event i
        $i := i + 1$
        $j :=$ <chained event, if any>
      end while
    end while
    $x_{k+1} := \tilde{x}, \quad z_{k+1} := \tilde{z}, \quad q_{k+1} := \tilde{q}$
  end for
  $\lambda p := 0, \quad \lambda_{u_\bullet} := 0$ ▷ Initialize to zero
  for $k = N-1, \ldots, 0$ **do** ▷ Backward integration
    if $k < k_{\text{prev}}$ **then**
      $k_{\text{prev}} :=$ <retrieve saved value>
      $[t, \tilde{x}, \tilde{z}] = \texttt{PREVIOUS\_EVENT}(k, i)$
      $\texttt{RESET}(t, \tilde{x}, \tilde{z}, p, u_k)$
      $\texttt{ADVANCE}(t_{k+1})$
    end if
    if $k < k_{\text{step}}$ **then**
      $\texttt{IMPULSE}(\lambda_{x_{k+1}}, \lambda_{z_{k+1}}, \lambda_{q_{k+1}})$
      $k_{\text{step}} := \texttt{NEXT\_IMPULSE}(k, \lambda_{x_\bullet}, \lambda_{z_\bullet}, \lambda_{q_\bullet})$
    end if
    for all events i in interval k in reverse order **do**
      $[\tilde{\lambda}_x, \tilde{\lambda}_p, \tilde{\lambda}_u] := \texttt{RETREAT}(t^{(i)})$
      $[\tilde{\lambda}_x, \tilde{\lambda}_z, \tilde{\lambda}_p^E, \tilde{\lambda}_u^E] := \texttt{ADJ\_REINIT}(i, \tilde{\lambda}_x, \tilde{\lambda}_z)$
      $[t, \tilde{x}, \tilde{z}] = \texttt{PREVIOUS\_EVENT}(k, i)$
      $\texttt{RESET}(t, \tilde{x}, \tilde{z}, p, u_k)$
      $\texttt{ADVANCE}(t_{k+1})$
      $\lambda_p := \lambda_p + \tilde{\lambda}_p + \tilde{\lambda}_p^E$
      $\lambda_{u_k} := \lambda_{u_k} + \tilde{\lambda}_u + \tilde{\lambda}_u^E$
    end for
    $[\tilde{\lambda}_x, \tilde{\lambda}_p, \tilde{\lambda}_u] := \texttt{RETREAT}(t_k)$
    $\lambda_p := \lambda_p + \tilde{\lambda}_p, \quad \lambda_{u_k} := \lambda_{u_k} + \tilde{\lambda}_u$
  end for
  $\lambda_{x_0} := \tilde{\lambda}_x$
  $\lambda_{z_0} := 0$
  return $x_\bullet \in \mathbb{R}^{n_x \times (1+N_f)}$, $z_\bullet \in \mathbb{R}^{n_z \times (1+N_f)}$, $q_\bullet \in \mathbb{R}^{n_q \times (1+N_f)}$, $\lambda_{x_0} \in \mathbb{R}^{n_x \times (1+N_f) N_a}$, $\lambda_{z_0} \in \mathbb{R}^{n_z \times (1+N_f) N_a}$, $\lambda_p \in \mathbb{R}^{n_p \times (1+N_f) N_a}$, $\lambda_{u_\bullet} \in \mathbb{R}^{n_u \times (1+N_f) N_a}$
end procedure

the output time. If there are multiple zero crossing events predicted, only the soonest one will be considered. Also, ommitted in the algorithm for ease of presentation, if a zero-crossing event is predicted before the next input change, the stopping time for internal time stepping will be updated accordingly.

- If after this initial integration, the zero crossing functions and their derivatives w.r.t. time indicate that a zero crossing event has occurred or is still predicted to occur before the desired output time, a rootfinding iteration will start. The algorithm is an Newton method, with a fallback to bisection if $\dot{e}$ has the wrong sign. This fallback can e.g. happen if $\dot{e}_j$ is non-positive, even though the sign of e_j indicates that a zero crossing from negative-to-positive has occured, or if the predicted event crossing happens before the start of the integration interval. During the rootfinding iterations, the solver interfaces will be responsible for updating the state to a given time (which may require small steps backwards in time).

- When the zero crossing iteration has reached a given tolerance, or hit a user-selected maximum number of iterations, the corresponding values for x, z and q along with time and zero-crossing index are returned to the user.

We do not include specific handling of the case where the event time is explicitly given, e.g. as a function of p, u and non-changing components of x, but note that the above algorithm will find the exact time of such events in a single iteration since $e(t)$ is linear in t.

3.3 Forward sensitivity analysis

For the forward sensitivity analysis, the function `REINIT` in Algorithm 3 needs to be generalized. To get the correct sensitivity propagation through the event, we must take into consideration that the event time t may depend on the state. We can handle this at the event considering the time t to be implicitly defined by the corresponding zero crossing function:

$$f_{\text{zero}}^{(j)}(t, x, z, p, u_k) = 0 \quad \Leftrightarrow \quad t = G(x, z, p, u_k) \tag{12}$$

We can propagate forward sensitivities through this function using the implicit function theorem, similar to how forward sensitivities for CasADi's `Rootfinder` class implemented. Since it is a scalar function, the propagation can easily be calculated:

$$\begin{aligned} \hat{t} &:= \frac{\partial t}{\partial x}\hat{x} + \frac{\partial t}{\partial z}\hat{z} + \frac{\partial t}{\partial p}\hat{p} + \frac{\partial t}{\partial u}\hat{u} \\ &= -\frac{1}{\dot{e}_j}\left(\frac{\partial f_{\text{zero}}}{\partial x}\hat{x} + \frac{\partial f_{\text{zero}}}{\partial z}\hat{z} + \frac{\partial f_{\text{zero}}}{\partial p}\hat{p} + \frac{\partial f_{\text{zero}}}{\partial u}\hat{u},\right) \end{aligned} \tag{13}$$

where $\hat{t}$ are the forward sensitivities of t and the corresponding forward seeds are $\hat{x}$, $\hat{z}$, $\hat{p}$ and $\hat{u}$, respectively.

With $\hat{t}$ for each sensitivity direction calculated, we are able to propagate the forward sensitivities through the reinit function:

$$\hat{x}_E := \frac{\partial E_x}{\partial t}\hat{t} + \frac{\partial E_x}{\partial x}\hat{x} + \frac{\partial E_x}{\partial z}\hat{z} + \frac{\partial E_x}{\partial p}\hat{p} + \frac{\partial E_x}{\partial u}\hat{u}, \quad (14)$$

where E_x is the calculation of x using E in (7). This calculation is performed using a forward directional derivative applied to the reinit function (7). Since the reinit function will only provide a guess for z (the exact value being determined by the DAE), no derivative propagation is needed.

Finally, we need to consider that sensitivity of $\hat{t}$ needs to be propagated to the duration of the subsequent interval. For example, if a small perturbation Δp in an input parameter p leads to the event happening a time Δt later, the subsequent integration interval will be Δt shorter. We can account for this by using $\dot{x}$ obtained from (9) and the known sensitivity in duration length ($-\hat{t}$).

$$\hat{x}_{\text{REINIT}} := \hat{x}_E - \hat{t}\dot{x} \quad (15)$$

3.4 Adjoint and forward-over-adjoint sensitivity analysis

Algorithm 3 also include a tentative implementation of adjoint sensitivity analysis and second order (forward-over-adjoint) sensitivity analysis. During the backward integration, there is no need to detect zero crossings. Instead we will simply keep records of the events (times and corresponding event indices) during the forward integration and then visit the same events in reverse order during the backward integration. Second order derivatives are handled by allowing all variables to have multiple right-hand-sides.

As of this writing, the extension of the adjoint sensitivity support to support events is still ongoing. The parts in Algorithm 3 related to adjoint and forward-over-adjoint sensitivity analysis therefore reflects the planned implementation.

4 Examples

4.1 Forward sensitivities for a bouncing ball

In our first example, we perform an analytical forward sensitivity analysis for a bouncing ball and compare the results with a finite-difference approximation. The system has two states corresponding to height h and velocity v, i.e. the state vector is $x = [h; v]$. The corresponding ODE is:

$$\dot{h} = v, \quad \dot{v} = -9.81 \quad (16)$$

When the ball hits the ground at $h = 0$, defined by $f_{\text{zero}}(x) = -h$, an event will be triggered defined by:

$$x = \begin{bmatrix} 0 \\ -0.8\,v^{-}, \end{bmatrix} \quad (17)$$

where v^{-} is the velocity immediately before the event.

In the leftmost figures of Figure 1, we show the event simulation, over 7 s for a ball starting at rest at $h = 5$, using SUNDIALS/CVODES as the interfaced solver. The remaining figures show the sensitivities of h and v with respect to perturbations in $h(0) = h_0$ and $v(0) = v_0$, respectively. The results are compared to a basic finite differencing perturbation of the whole simulation trajectory.

To understand the results in the lower right subplot, which may seem counter-intutitive, it can be shown that for a ball starting at rest, the derivative of the time of the first bounce T_{bounce} with respect to initial velocity can be written:

$$\frac{dT_{\text{bounce}}}{dv_0} = \frac{1}{g}. \quad (18)$$

Therefore, the first derivative of the ball velocity at impact $v_{\text{impact}} = v_0 - g\,T_{\text{bounce}}$ with respect to initial velocity is zero:

$$\frac{\partial v_{\text{impact}}}{\partial v_0} = \frac{dv_0}{dv_0} - g\frac{dT_{\text{bounce}}}{dv_0} = 1 - \frac{g}{g} = 0. \quad (19)$$

The first order sensitivity of the ball velocity *after* the bounce with respect to initial velocity, is therefore just due to how much time has elapsed since the bounce:

$$\frac{dv(t; h_0, v_0)}{dv_0} = -\frac{dT_{\text{bounce}}}{dv_0}(-g) = 1. \quad (20)$$

This theoretical result, which holds in the *almost everywhere* sense, is confirmed with the analytical forward sensitivities (blue line). The result repeats itself at subsequent bounces. For the corresponding finite difference approximation (red line), in contrast, the numerical error will grow for every bounce.

4.2 Parameter estimation for a breaking spring

As a second example, we consider the a simple model of a spring formulated in Modelica. When the spring is extended too far, an event corresponding to the spring "breaking" is triggered:

```
model BreakingSpring
    input Real m(start = 1)
      "PARAMETER:Mass";
    output Real v(start = -5, fixed = true)
      "velocity";
    output Real x(start = -1, fixed = true)
      "displacement";
    input Real k(start = 2)
      "PARAMETER:spring constant";
    input Real c(start = 0.1)
      "PARAMETER:damping constant";
    input Real d(start = 0) "disturbance";
    Real f "spring force";
    Boolean b "Is the spring broken?";
initial equation
  b  = false;
equation
    der(x) = v;
    f = if not b then -k * x + d else 0;
```

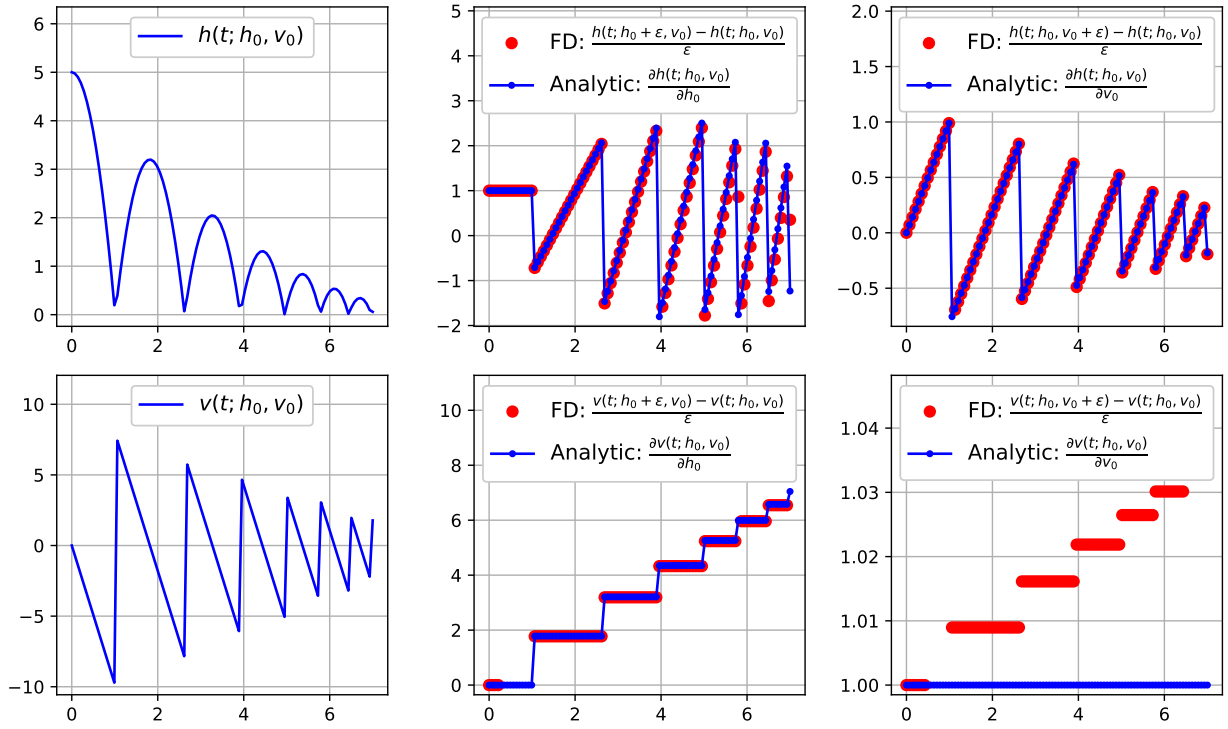


Figure 1. Forward sensitivity analysis for a bouncing ball, comparison with finite differences (FD)

```
    m * der(v) + c * v = f;
    when x>2 then
      b = true;
    end when;
end BreakingSpring;
```

Compared to the bouncing ball model, the breaking spring model includes the following:

- A free input parameter `d`, corresponding to u in (1)
- Three tunable parameters, `m`, `k` and `c`, corresponding to p in (1). To ensure derivative information is available after compiling the model (e.g. into an FMU), we will model tunable parameters as *controls*, using a `Parameter:` prefix in the description string to distinguish them from regular controls.
- A boolean state `b`, which is updated discontinuously at events. Since CasADi does not have the concept of discrete states, we will model discrete states as real-valued states with zero time derivative, i.e. `b` is a component of x (say, index i_b) with $\dot{x}[i_b] = 0$.

Using OpenModelica 1.24, we compile the above model into an XML file, containing a symbolic representation of the problem, using the approach described in (Shitahun et al. 2013). This model is then imported into a CasADi `DaeBuilder` instance, which in turn is used to generate an analytically differentiable integrator object in CasADi, again using SUNDIALS/CVODES as the interfaced solver.

CasADi integrator instances can be embedded into expression graphs corresponding to different optimization formulations. In Figure 2 (left), we show the result of solving a parameter estimation problem using the hybrid integrator. The problem corresponds to finding the parameter values m, k and c that minimize a sum-of-squares cost function:

$$\underset{m,k,c}{\text{minimize}} \quad \sum_{k=1}^{N} (x_k - \tilde{x}_k)^2, \tag{21}$$

subject to the hybrid dynamical equation and bounds of the parameter. To generate simulated measurements $\tilde{x}_k$, we add Gaussian noise to the simulation result corresponding to known values of the parameters. The optimization is done for a known disturbance vector d, but again with random noise added, as shown in Figure 2 (right). The problem is solved using a single-shooting discretization, using IPOPT as an optimizer.

5 Summary

In this work, we have shown an extension of the DAE simulation routines in CasADi to handle systems with events. This includes the efficient calculation of analytical sensitivity information, as needed by gradient-based optimization algorithms, also in the presence of events. We provided details of the forward sensitivity implementation, illustrated with two examples, as well as details on the ongoing work to implement adjoint and forward-over-adjoint sensitivity analysis with events. While we have thus far relied on relatively simple toy examples available as CasADi

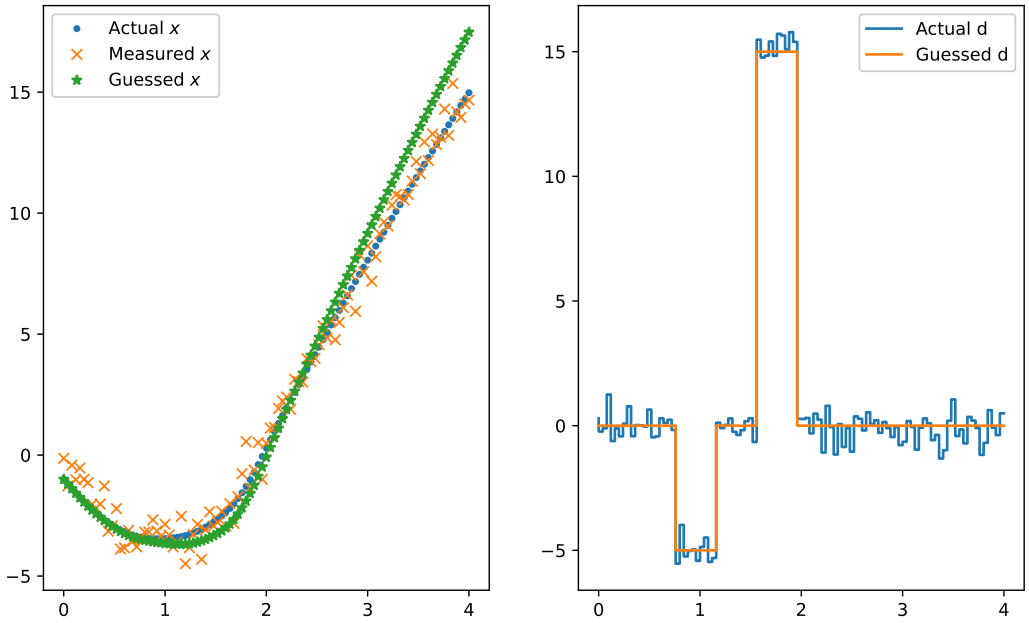


Figure 2. Parameteter estimation for a breaking spring, with generated measurement values.

symbolic expression graphs, the intention is to use this feature to implement dynamic optimization for challenging cyber-physical systems, including but not limited to systems implemented in Modelica. We will discuss the path to handle such systems in the following section.

6 Outlook

The work presented in this paper is in active development, with additional features being added as they become required by applications. In the following, we discuss some of the most important extensions planned.

6.1 Event support for models provided as standard FMUs

The ultimate goal of this work is to enable the formulation and solution real-world optimization problems with event dynamics, in particular those formulated in Modelica. In our initial experiments, presented in Section 4, we used a symbolic coupling based on a legacy XML-based symbolic coupling between OpenModelica and CasADi. This coupling is neither well maintained, nor generic enough to handle realistic systems. It is also restricted to a single exporting tool (OpenModelica).

A recent addition to CasADi is the import of general FMUs adhering to FMI 2, as described in (Joel Andersson 2023). In pre-release versions of CasADi, this support has since been extended to FMI 3, including the interface to adjoint derivatives of model equations. Our plan is to use the FMI interface together with the event support in the CasADi integrators to be able to efficiently and conveniently solve optimization problems for real-world Modelica models. Note that by relying on FMI, the structure of the underlying Modelica model becomes irrelivant as long as it conforms with the FMI standard and has the prerequisite smoothness properties for numerical optimization. It is also possible to use models that include variables that cannot be represented in CasADi, for example records or string-valued expressions, as long as these variables are not manipulated by the optimizer.

Since the FMI format, as written, does not natively conform to the required formulation (9), some reformulations of the Modelica models may be needed prior to FMU generation. In particular:

- Event indicator expressions will need to be linked to differentiable model outputs. That means that the argument of *when*-constructs in Modelica may need to be assigned to additional model outputs, following some naming convention. This convention ensures that derivative information is available for the zero-crossing functions.

- The reinit equations need to consist of simple *outputs-to-states* mappings. This means that at events, the differential state should be assigned to some of variable with output causality. Each event indicator should uniquely map to an assignment, which may require the addition of additional output variables. This convention ensures that derivative in-

formation is available for the reinit functions.

- We may need to reformulate free parameters as inputs (as in Section 4.2) to ensure that analytic derivative information with respect to these parameters is included in the FMU. Alternatively, we can rely on tool-specific extensions, such as using the annotation `"evaluate = false"` in Dymola to ensure that the parameter can be manipulated by the optimizer.

6.2 A standardized symbolic interface based on *Base Modelica*

A symbolic model interface, such as the XML-based interface used in Section 4.2 will always have some fundamental advantages over a "black box" binary interface. This is especially true when the model dimensions are small or when higher order derivative information is needed. To be able to take advantage of the fundamental advantages of a symbolic interface, we plan to replace the XML interface with a new symbolic interface based on a ANTLR4-based parser for the emerging *Base Modelica* standard (Kurzbach et al. 2023). This interface builds on our previous work with Pymoca and Cymoca, cf. `https://github.com/pymoca/pymoca` and `https://github.com/jgoppert/cymoca`, respectively.

Since Base Modelica is intended to become a standard, with ongoing work to export models in this format from different Modelica compilers, the approach should be compatible with multiple tools. The hope is also that since Base Modelica is in essence a small subset of the full Modelica language, implementing and maintaining a parser should be possible with a reasonable effort.

6.3 Event detection in interfaces

In Section 3.2, we presented an approach to locate events based on an algorithm implemented in the integrator base class. An alternative to this algorithm would be to use the solver's native event-finding algorithm, such as the Illinois algorithm (Hiebert and Shampine 1980) used in SUNDIALS. This algorithm has proven efficient and robust for numerous applications. There is also value in using the same event finding algorithm as the modeler uses for hybrid simulation.

6.4 Algebraic variables in the zero-crossing functions and reinit functions

The implementation of the proposed approach was done in a way that was generic for both ODEs and DAEs, although it had yet to be tested with DAEs as of this writing. The implementation would also need an extension to be able to handle the case when algebraic variables (z) explicitly enter in the zero-crossing functions or reinit functions.

Disclaimer

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

Acknowledgements

This material is based on research sponsored by DARPA under agreement number FA8750-24-2-0500. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The authors also want to acknowledge the helpful remarks by the anonymous reviewers.

References

Andersson, J. (2013-10). "A General-Purpose Software Framework for Dynamic Optimization". PhD thesis. Arenberg Doctoral School, KU Leuven.

Andersson, Joel (2023-10). "Import and Export of Functional Mockup Units in CasADi". In: *Proceedings of the 15th International Modelica Conference*. Vol. 2855, pp. 321–326.

Andersson, Joel A E et al. (2019). "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1, pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

Corner, Sebastien, Corina Sandu, and Adrian Sandu (2019). "Modeling and sensitivity analysis methodology for hybrid dynamical system". In: *Nonlinear Analysis: Hybrid Systems* 31, pp. 19–40. ISSN: 1751-570X. DOI: https://doi.org/10.1016/j.nahs.2018.07.003. URL: https://www.sciencedirect.com/science/article/pii/S1751570X1830058X.

Hiebert, Kathie L. and Lawrence F. Shampine (1980-02). *Report SAND80-0180: Implicitly Defined Output Points for Solutions of ODEs*. Tech. rep. Sandia National Laboratory.

Hindmarsh, Alan C et al. (2005). "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software (TOMS)* 31.3, pp. 363–396. DOI: 10.1145/1089014.1089020.

Kurzbach, Gerd et al. (2023-10). "Design proposal of a standardized Base Modelica language". In: *Proceedings of the 15th International Modelica Conference*. Vol. 2855, pp. 469–478.

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

Rackauckas, Christopher and Qing Nie (2017). "DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia". In: *Journal of Open Research Software* 5.1.

Shitahun, Alachew et al. (2013). "Model-Based Dynamic Optimization with OpenModelica and CasADi". In: *IFAC Proceedings Volumes* 46.21. 7th IFAC Symposium on Advances in Automotive Control, pp. 446–451. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20130904-4-JP-2042.00166. URL: https://www.sciencedirect.com/science/article/pii/S1474667016384117.

Steady-state Optimization of Modelica Models and Functional Mockup Units with Pyomo

Jesse Gohl[1] Hubertus Tummescheit[1] Robin Andersson[1]
Matthew Stuber [2]

[1]Modeon Inc, USA,
[2]Dept. of Chemical & Biomolecular Engineering, University of Connecticut
jesse.gohl@modelon.com

Abstract

This paper describes two ways on how to interface Functional Mockup Units (FMUs) and Modelica models through the Pyomo's foreign function interface with Pyomo. Pyomo is a Python-based, open-source optimization modeling language with a diverse set of optimization capabilities. Modelica has arguably much better modeling capabilities than Pyomo, but Pyomo integrates excellent optimization solvers, such as Ipopt (Wächter et al. 2006), and provides a good optimization infrastructure. The Interface has been developed in the context of a NAWI, (National Alliance Water Innovation) Hub project in collaboration with the University of Connecticut and Sandia National Labs. The optimization has been set up and tested within Modelon's Modelica platform Modelon Impact. An unpublished, detailed multi-effect desalination plant developed by Prof. Matt Stuber in the context of (Stuber et al., 2015) has been used to demonstrate the capabilities, as well as simple test models, and design models from Modelon's commercial Libraries.

Keywords: Modelica, Functional Mockup Interface, FMI, Steady-state Optimization, Design Optimization

1 Introduction

There is a growing list of options to perform optimization studies involving Modelica models. A number of simulation tools support optimization natively within their simulation environment (OpenModelica, Ansys Twin Builder®, System Modeler, the Modelica Optimization Library, etc.). The models can also be exported as Functional Mockup Units (FMU) and imported to specialized optimization tools (modeFrontier®, Optimus®, etc.). A couple of dynamic optimization (Bryson 1999) methods (OpenModelica and JModelica) rely on CasADi (Andersson 2011; Bachmann 2012; Ruge 2014). The tools transfer the Modelica model to CasADi for automatic differentiation and optimization. Originally this was done via an XML file format for both tools (Magnusson 2015), but the Optimica Compiler Toolkit (OCT) has evolved from JModelica to support more comprehensive coverage of the Modelica language, transferring large parts of the language into a native CasADi problem (Modelon 2024). This is done automatically but both methods rely on sufficiently restricted models to avoid unsupported constructs by CasADi. The Optimica Compiler Toolkit also includes support for derivative free optimization using the Nelder-Mead simplex method (Nelder and Mead 1965; Fletcher 1987) for static optimization. In addition, OpenModelica and OCT support the Optimica® language for the description of the optimization problem. This language is an extension to the Modelica language. An alternative to the above methods that is explored in this paper, is to connect the FMU to a solver through the Pyomo Python toolbox for optimization and solution through its connection to IPOPT.

The Functional Mockup Interface (FMI, Modelica Association 2024) is a standardized, widely accepted API implemented by more than 200 simulation tools for executable simulation models. However, it has been designed for transient simulations, not steady-state (design-oriented) simulation models. It can be used to compute stationary points for transient system models, but the API lacks functions to compute sensitivities with respect to decision variables in an optimization problem symbolically. It is possible, but less accurate and performant to approximate the derivates by finite differences. Modelon Impact offers to convert Modelica model parameters to inputs when translated into an FMU. This allows using the symbolic derivatives with respect to inputs in the standard FMI-interface in the solution process. While this is helpful, and allows to optimize arbitrary Modelica-models, it is not sufficient to robustly optimize large and highly non-linear problems.

This paper will present two different interfaces between Modelica models and Pyomo's optimization algorithms:

1. An interface based on generic FMUs, enhanced with Modelon Impact's capability to convert parameters to inputs for improved accuracy of Jacobian computations.

2. Modelon Impact's internal interface for solving large non-linear systems for steady-state design problems.

Modelon's interface is similar to FMI but adds the ability to couple a Modelica model to an external steady-state solver. This is in spirit like a model exchange FMU.

The first interface allows to couple FMUs for any FMI-compliant tool to Pyomo, the second interface is more robust, able to solve larger models, but is only available for Modelica models in Modelon Impact.

2 Requirements for efficient gradient-based optimization

A typical constrained static optimization problem can be written as follows:

$$\begin{gathered}\text{Minimize } f(x) \\ \text{Subject to } g_j(x) \leq 0, j = 1, \dots, p \\ h_i(x) = 0, i = 1, \dots, m\end{gathered} \tag{1}$$

where $g_j, h_i: \mathrm{R}^n \rightarrow \mathrm{R}$ are inequality and equality constraints respectively, and $f(x)$ is the objective function. In the context of models coming from Modelica, usually all equality constraints come from the model equations, and the objective function and inequality constraints must be handled outside of the standard Modelica language. For this prototype interface, we have chosen to not use the Optimica Modelica extension proposed by (Åkesson et. al., 2011). A graphical Modelica user environment such as Modelon Impact can allow those to be entered in the definition of the optimization experiment.

With the optimization of a model in Modelica, there are two fundamentally different ways of solving a steady-state optimization problem: 1) the nonlinear solver is the same that is used for steady-state initialization in a Modelica simulator, and the optimization solver is "wrapped" around that, i.e. nested solvers, or 2) the optimization solver handles everything, and treats the model equations as equality constraints ($h_i(x)$), i.e. a single solver. With generic FMUs, a nested solver is the only option. With Modelon Impact's steady-state interface, the second option is possible, and has demonstrably been the faster and more robust option in our testing.

It should be noted that the dimension m of $h_i(x)$ can be large, several thousand equations, which can lead to excessive solution times. However, so called tearing of equation systems (Baharev et al., 2016) can dramatically reduce the dimension, is generally used by Modelica tools, and is also used by us in this interface to reduce the dimension of the $\boldsymbol{h}(\boldsymbol{x})$ vector equations that are exposed to the solver. Also note that, even after the dimensionality reduction through tearing, the resulting subset of the equations is still large enough to justify the use of sparse solver interfaces.

There are further important numerical requirements for efficiency and robustness, some of which are on the model-side of the interface, and some on the optimization solver side of the interface. Pyomo offers methods to compute first and second derivatives for constraints and objective function. On the model side, these can be approximated numerically, or computed analytically. For standard, transient FMUs, they must be approximated numerically. Modelon's dedicated steady-state, FMU-like interface allows to provide even analytic Jacobians to the Pyomo external function interface. Scaling of variables is also important for robustness and can be achieved by making use of the Modelica and FMI feature of nominal values for variables. Note that scaling is much more important than in the simulation case.

3 The Pyomo/PyNumero External Function Interface

The core of both interfaces of this work, mentioned in the introduction, extends from the `ExternalGreyBoxModel` class of PyNumero (Rodriguez et al. 2024). This class allows users to use external models with Pyomo. The class translates the external model, usually written in Python code, into a Pyomo model by wrapping a set of standard methods, necessary to define an optimization problem. Examples of these methods are `evaluate_outputs`, `evaluate_equality_constraints`, `evaluate_jacobian_outputs`, and `evaluate_jacobian_equality_constraints`. The second derivatives can also be defined using the `evaluate_hessian_outputs` and `evaluate_hessian_equality_constraints` methods. The new classes from this effort, wrap (evaluate) the FMI interface methods from a PyFMI FMU object within these PyNumero methods.

The main class constructor of this work accepts a PyFMI FMU object or a steady state FMUProblem[1] object from Modelon's steady-state interface class. The methods provided by these objects are evaluated within the PyNumero methods during the solution of the optimization or static problem. The constructor also accepts lists of relevant variables for inputs, outputs, and constraints, as well as modifiers to FMU parameters and some additional options specific to this interface. The input, output, and constraint lists identify the relevant

[1] The steady state interface in PyFMI follows to the largest extent possible the FMI-Standard 2.0, but it defines a steady-state problem, but a transient one as "normal" FMUs.

10.3384/ECP207109 DOI

FMU variables for the optimization or steady state problem.

4 The Interfaces

In general, FMUs define a mathematical representation of an engineering problem. Frequently, this is defined as an experiment that simulates a transient, progression in time, as a differential-algebraic equation (DAE) system. The *GenericFMU* class from the new Python package simulates an FMU from the initial time to the final time to supply the values needed by an optimization solver. This includes the outputs, constraint residuals, and their derivatives. Because many FMUs support these methods, this allows the new interface to support the widest range of FMUs, regardless of the generation tool.

Static problems do not include a dependence on time because time and derivatives with respect to time do not appear in the DAE system. This means that the mathematical representation of the problem is reduced to a set of nonlinear equations with one or more solutions (hopefully!) that needs to be solved by a nonlinear programming (NLP) solver. Modelon's steady state problem interface is an efficient translation of these types of problems to pass to an NLP solver. Evaluating the outputs and constraints does not require stepping forward in time and is the simplest and most efficient case because, since there are no dynamic states (no time derivatives), the outputs only depend on the inputs in continuous-time mode (as defined by the FMI specification). This means the solver can update the inputs and the outputs can be evaluated without additional method calls. This is also the most efficient case for the evaluation of derivatives and can take advantage of the directional derivatives defined by the FMU. This applies when the inputs to the optimization problem are also the inputs to the FMU (v_{known} of section 2.1.9 of the FMI specification version 2.0.4). The methods of the *StaticFMU* class avoid full simulations to compute the values needed by the optimization solver.

The static interface also supports other cases that include FMUs with event indicators, optimization inputs that are not also FMU inputs (usually this means causality = parameter), and co-simulation FMUs. The static interface checks for these cases and defines the appropriate evaluation methods depending on the situation. For the case of FMUs with event indicators an event handling loop is activated if an event is detected by the indicators. The allowed modes for modifying parameter values are more restricted. For example, normal parameters (e.g. causality = parameter and variability = fixed), can only be modified before exiting initialization mode. Therefore this version of the interface package resets the FMU, sets the updated parameter values, then initializes the FMU to ensure consistency. Co-simulation type FMUs use a similar strategy of setting values and initializing.

5 Test Models and Results

PyNumero includes an example of a problem based on a continuously stirred reactor as described in section 7.4.4 of Bynum et al., (2021). The problem is to find the incoming flow rate of an inlet stream with a single species to a chamber with competing reactions, that maximizes the concentration of a specific species in the outlet stream.

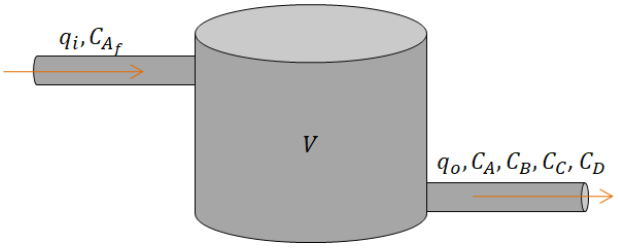


Figure 1. Continuously stirred reactor diagram

The reactions within the vessel are governed by the rates k_1, k_2, and k_3 through the following sequences.

$$A \overset{k_1}{\rightarrow} B \overset{k_2}{\rightarrow} C \tag{2}$$

$$2A \overset{k_3}{\rightarrow} D \tag{3}$$

This can be represented by the following model that describes the rates of change of the outlet concentrations C_A, C_B, C_C, and C_D.

$$\dot{C}_A = \frac{q}{V} C_{A_f} - \frac{q}{V} C_A - k_1 C_A - 2k_3 C_A^2 \tag{4}$$

$$\dot{C}_B = -\frac{q}{V} C_B + k_1 C_A - k_2 C_B \tag{5}$$

$$\dot{C}_C = -\frac{q}{V} C_C + k_2 C_B \tag{6}$$

$$\dot{C}_D = -\frac{q}{V} C_D + k_3 C_A^2 \tag{7}$$

$$\frac{q}{V} = sv \tag{8}$$

At steady state, the rates of change of concentrations are zero and the inlet and outlet flow rates are equal. This is equivalent to a statics problem that does not involve time.

$$\dot{C}_X \rightarrow 0 \tag{9}$$

$$q_o \rightarrow q_i \tag{10}$$

Notice the steady state problem involves a quadratic term for C_A. Depending on the tool, this can require iteration to solve numerically. The example in PyNumero encodes this problem in Python code to demonstrate the external interface to a Pyomo model. The `ExternalGreyBoxModel` of this example appears as in

the following image that shows the method definitions from the base class:

```
class ReactorConcentrationsOutputModel(ExternalGreyBoxModel):
    def input_names(self):…

    def output_names(self):…

    def set_input_values(self, input_values):…

    def finalize_block_construction(self, pyomo_block):…

    def evaluate_outputs(self):
        sv = self._input_values[0]
        caf = self._input_values[1]
        k1 = self._input_values[2]
        k2 = self._input_values[3]
        k3 = self._input_values[4]
        ret = reactor_outlet_concentrations(sv, caf, k1, k2, k3)
        return np.asarray(ret, dtype=np.float64)

    def evaluate_jacobian_outputs(self):…
```

Figure 2. PyNumero reactor model example Python class definition

The `finalize_block_construction` method is the location for specifying the Pyomo variable attributes like bounds and starting values. Notice the variable bounds are specified separately from the constraints. In this example a lower bound of zero is defined for all the concentrations. Evaluation of the residuals occurs in the `_model` function, defined in the function `reactor_outlet_concentrations`. This is where equations (4) – (8) are defined in the Python model. As the problem is written, solution of this system requires an iterative approach to drive the four residual values toward zero. This relies on *fsolve* from scipy.optimize to minimize the residuals. Whenever the optimization solver needs the values of the outputs (or to estimate the Jacobian as described next), the external NLP solver, *fsolve*, must be called to find the values of `ca`, `cb`, `cc`, and `cd` that result in sufficiently small residual values. This structure can also be used with the FMU based approach if the FMU requires iteration to resolve nonlinear systems of equations. Alternatively, the variables and residual values can be exposed to the optimization solver and the residuals handled as equality constraints (with equality zero). This is possible with the steady-state FMU interface package and is usually considered more efficient and robust.

```
def reactor_outlet_concentrations(sv, caf, k1, k2, k3):
    def _model(x, sv, caf, k1, k2, k3):
        ca, cb, cc, cd = x[0], x[1], x[2], x[3]

        # compute the residuals
        r = np.zeros(4)
        r[0] = sv*caf + (-sv-k1)*ca - 2*k3*ca**2
        r[1] = k1*ca + (-sv-k2)*cb
        r[2] = k2*cb - sv*cc
        r[3] = k3*ca**2 - sv*cd

        return r

    concentrations = \
        fsolve(lambda x: _model(x, sv, caf, k1, k2, k3), np.ones(4), xtol=1e-8)

    # Todo: check solve status
    return concentrations
```

Figure 3. PyNumero reactor model output evaluation function

The Jacobian is provided by the `evaluate_jacobian_outputs` method of the reactor class. The matrix is returned in the sparse coordinate format as an instance of a *coo_matrix* class from the scipy.sparse package. The PyNumero example estimates the matrix using finite differences, but it could also have been analytically computed from the model equations.

The Pyomo code to define the optimization problem to maximize the concentration of species B appears as in the following:

```
def maximize_cb_outputs(show_solver_log=False):
    # in this simple example, we will use an external grey box model representing
    # a steady-state reactor, and solve for the space velocity that maximizes
    # the concentration of component B coming out of the reactor
    m = pyo.ConcreteModel()

    # create a block to store the external reactor model
    m.reactor = ExternalGreyBoxBlock(
        external_model=ReactorConcentrationsOutputModel()
    )

    # The reaction rate constants and the feed concentration will
    # be fixed for this example
    m.k1con = pyo.Constraint(expr=m.reactor.inputs['k1'] == 5/6)
    m.k2con = pyo.Constraint(expr=m.reactor.inputs['k2'] == 5/3)
    m.k3con = pyo.Constraint(expr=m.reactor.inputs['k3'] == 1/6000)
    m.cafcon = pyo.Constraint(expr=m.reactor.inputs['caf'] == 10000)

    # add an objective function that maximizes the concentration
    # of cb coming out of the reactor
    m.obj = pyo.Objective(expr=m.reactor.outputs['cb'], sense=pyo.maximize)

    solver = pyo.SolverFactory('cyipopt')
    solver.config.options['hessian_approximation'] = 'limited-memory'
    results = solver.solve(m, tee=show_solver_log)
    pyo.assert_optimal_termination(results)
```

Figure 4. PyNumero reactor model optimization problem statement in Pyomo code

Notice that an instance of a Pyomo *ConcreteModel* class, `m`, is passed to an instance of the *SolverFactory* class, solver. This will be the same when using the new FMU/Pyomo interface package. Also notice that the values of `k1`, `k2`, `k3`, and `caf` are handled as equality constraints. Only `sv` is free to iterate during optimization. Pyomo hands this problem to cyipopt to solve the problem and produces the following print out of the results:

```
inputs : Size=5, Index=reactor._input_names_set
    Key : Lower : Value                : Upper : Fixed : Stale : Domain
    caf :  None :              10000.0 :  None : False : False :  Reals
     k1 :  None :    0.8333333333333334 :  None : False : False :  Reals
     k2 :  None :    1.6666666666666667 :  None : False : False :  Reals
     k3 :  None : 0.0001666666666666666 :  None : False : False :  Reals
     sv :     0 :    1.3438109305149577 :  None : False : False :  Reals
outputs : Size=4, Index=reactor._output_names_set
    Key : Lower : Value              : Upper : Fixed : Stale : Domain
     ca :     0 :   3874.25779977848 :  None : False : False :  Reals
     cb :     0 :   1072.43720049758 :  None : False : False :  Reals
     cc :     0 : 1330.0943532862536 :  None : False : False :  Reals
     cd :     0 : 1861.6053232188908 :  None : False : False :  Reals
```

Figure 5. PyNumero reactor model solution results log

This shows that the solver converged to a solution of 1.34 with 1072 as the maximum concentration of species B. Alternatively, the problem can be written in another language, from which an FMU can be generated. For example, in Modelon Impact the reactor model definition could be written in Modelica as:

```
model CSTR_static
  "Reactor example based on the PyNumero example"

  parameter Real sv(min=0)=1;
  parameter Real caf(min=0)=1;
  parameter Real k1(min=0)=1;
  parameter Real k2(min=0)=1;
  parameter Real k3(min=0)=1;

  Real ca;
  Real cb;
  Real cc;
  Real cd;

equation
  0 = sv*caf + (-sv-k1)*ca - 2*k3*ca^2;
  0 = k1*ca + (-sv-k2)*cb;
  0 = k2*cb - sv*cc;
  0 = k3*ca^2 - sv*cd;

end CSTR_static;
```

Figure 6. Static continuously stirred reactor model in Modelica

Notice that the variables include attributes like `min`, `max`, `start`, `nominal`. These can be included in the FMU's variable attributes and the new FMU/Pyomo interface package will use these to automatically define the required Pyomo attributes. The interface also allows the attributes to be provided in the problem statement for cases when the FMU does not include these.

It is also worth noting that an added benefit to encoding this model in Modelica is that the Modelica translator can "tear" (Baharev et al., 2016) the system of equations to minimize the size of equation blocks. In this case the block is torn to a size 1 system around variable `ca`.

With this model the solution space of can be explored by plotting the solution of `cb` as a sweep of values of `sv`. Notice the location and maximum of this curve correlates to the solution from Ipopt shown in Figure 5.

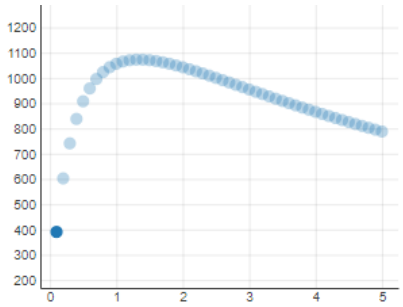


Figure 7. Sweep of outlet concentration, `cb`, (vertical axis) versus inlet flow rate, `sv` (horizontal axis)

A similar optimization problem can now be defined using the new FMU/Pyomo interface package. The package includes helper functions `static_pyomo_model` and `generic_pyomo_model` that return an instance of the PyNumero *ExternalGreyBoxBlock*, "`block`", as an element of an instance of a Pyomo *ConcreteModel*. The block contains an `external_model` that is an instance of one of either of the new FMU/Pyomo classes, *StaticFMU* or *GenericFMU*, as described in section 3. The methods in these classes are optimized for the type of FMU that is being used. An example of this is shown in the following code where `m` is the instance of a Pyomo *ConcreteModel* class that includes an instance of the *StaticFMU* class.

```
51  fmu = load_fmu('CSTR_static.fmu')
52
53  # create a block to store the external reactor model
54  m = static_pyomo_model(
55      problem=fmu,
56      input_list=['sv'],
57      output_list=['cb'],
58      csv_log_file=csv_log_file,
59      modifiers={'caf':10000.0, 'k1':0.83333, 'k2':1.6666, 'k3':0.00016666}
60  )
61
62  # add an objective function that maximizes the concentration
63  # of cb coming out of the reactor
64  m.obj = pyo.Objective(expr=m.block.outputs['cb'], sense=pyo.maximize)
65
66  solver = pyo.SolverFactory('cyipopt')
67  solver.config.options['hessian_approximation'] = 'limited-memory'
68  results = solver.solve(m, tee=show_solver_log)
69  pyo.assert_optimal_termination(results)
```

Figure 8. Optimization problem statement using the CSTR_static FMU.

By default the FMU's inputs and outputs will be used for the problem inputs and outputs, or they can be specified directly for FMUs without explicit I/O variables, as in the above code that specified `sv` and `cb` as the inputs and outputs respectively. Notice that the code that defines the optimization problem, i.e. the objective, solver, and its settings on lines 64 – 69 in the above code, is the same as lines 38 – 43 shown in Figure 4. Calling the `pprint` method of the Pyomo *ConcreteModel* instance, `m` prints the following information to the screen. This demonstrates that the found solution matches the solution shown in Figure 5.

```
inputs : Size=2, Index=block._input_names_set
    Key : Lower   : Value              : Upper  : Fixed : Stale : Domain
     sv :     0.0 : 1.3437710786561108 : 1e+100 : False : False :  Reals
outputs : Size=1, Index=block._output_names_set
    Key : Lower   : Value              : Upper  : Fixed : Stale : Domain
     cb : -1e+100 : 1072.4690498456466 : 1e+100 : False : False :  Reals
```

Figure 9. Solution log for the CSTR system using the static FMU/Pyomo interface

At this point it is important to note a difference in solver statistics between the two cases. In the original PyNumero example, Ipopt reports 13 iterations required to solve the problem. With the FMU based approach the solution required 17 iterations. The exact cause for this difference has not yet been determined but some differences can be noted. One difference with the previous example is the initialization. The original PyNumero example initializes the problem with `sv` = 5 and `ca` and `cb` both equal to 1 whereas the above FMU based example initialized at `sv` = 1 and `ca` and `cb` equal to 0. This can be changed in the problem statement by adding the following lines.

```
mdl = m.block.get_external_model()

mdl.input_vars['sv'].initial = 5
mdl.input_vars['ca'].initial = 1
mdl.output_vars['cb'].initial = 1

# This needs to be called again in order to ensure the bounds and
# initial conditions are set
mdl.finalize_block_construction(mdl.pyomo_block)
```

Figure 10. Applying consistent initial conditions to the problem statement

This reduces the number of iterations to 16 but still not to 13. This result is unexpected since the FMU based approach has both reduced the number of iteration variables and provides an analytic Jacobian. Resolution of this discrepancy would likely require low-level review of the verbose Ipopt log.

The above FMU based approach relied on the steady-state interface of the new FMU/Pyomo interface as described in the Introduction section. This allows the Ipopt solver to resolve the nonlinear block for the concentration of species A as described above. An alternative approach relies on the generic FMU/Pyomo interface described in the Introduction section. This approach supports any FMU, not just FMUs generated by Modelon Impact's steady-state interface. This is demonstrated with a generic FMU based on the model shown in Figure 6 and uses the *generic_pyomo_model* method of the FMU/Pyomo interface. Notice that the `ca` variable is no longer an input but is now an internal variable that will be resolved by the methods within the FMU.

```
# create a block to store the external reactor model
m = generic_pyomo_model(
    problem=fmu,
    input_list=['sv'],
    output_list=['cb'],
    final_time=1,
    csv_log_file=csv_log_file,
    modifiers={'caf':10000.0, 'k1':0.83333, 'k2':1.6666, 'k3':0.00016666}
)

# Not strictly necessary but just to show they can be changed if desired
mdl = m.block.get_external_model()

mdl.input_vars['sv'].initial = 5
mdl.output_vars['cb'].initial = 1

# This needs to be called again in order to ensure the bounds and
# initial conditions are set
mdl.finalize_block_construction(mdl.pyomo_block)
```

Figure 11. Comparable problem statement for the CSTR system using the generic FMU/Pyomo interface.

Because generic FMUs can have time varying equations, the *simulate* method of the PyFMI FMU object is used to evaluate the variables. For model exchange FMUs this will attach an external DAE (differential algebraic equation) solver to integrate forward in time from zero to some user defined final time. For co-simulation FMUs, the DAE solver is internal so the simulate method requests the solver step forward from zero to the final time. The values returned to the optimization solver (e.g. Ipopt) are the values at the end of the simulation, even if steady-state conditions have not been reached. This also means that the Jacobian must be computed numerically.

Passing this problem to the solver results in a failed solution because the maximum number of iterations is reached. Adding a constraint on the upper bound of the iteration variable, `sv`, helps resolve this issue.

```
mdl.input_vars['sv'].initial = 5
mdl.input_vars['sv'].maximum = 10
mdl.output_vars['cb'].initial = 1

# This needs to be called again in order to ensure the bounds and
# initial conditions are set
mdl.finalize_block_construction(mdl.pyomo_block)
```

Figure 12. Defining the upper bound on the iteration variable to resolve failed solutions.

With an upper limit on the iteration variable, the solution now converges to the same solution as before but requires more than 2000 iterations.

Notice the iteration variable, `sv`, is on the order of 1.0 while the output concentration variable, `cb`, is on the order of 1000. This can significantly affect the accuracy of the Jacobian, especially when it is computed through the finite difference. Instead of applying a bound on the iteration variable, we can define better nominal values of the variables. The FMU/Pyomo interface will then scale the inputs, outputs, and Jacobian elements based on these nominal values.

```
mdl.input_vars['sv'].initial = 5
mdl.output_vars['cb'].initial = 1
mdl.output_vars['cb'].nominal = 1000
```

Figure 13. Defining a nominal attribute for the output variable of the CSTR

In this case the solver converges to the solution much faster, although still not as fast as the case when an analytic Jacobian is used.

The last example using the CSTR model demonstrates the same problem but with a transient model. In this example the species concentrations are states of the differential equation system that evolve in time.

```
model CSTR_transient
  "Reactor example based on the PyNumero example"

  parameter Real sv(min=0)=1;
  parameter Real caf(min=0)=1;
  parameter Real k1(min=0)=1;
  parameter Real k2(min=0)=1;
  parameter Real k3(min=0)=1;

  Real ca(start=caf);
  Real cb(start=0);
  Real cc(start=0);
  Real cd(start=0);

equation
  der(ca) = sv*caf + (-sv-k1)*ca - 2*k3*ca^2;
  der(cb) = k1*ca + (-sv-k2)*cb;
  der(cc) = k2*cb - sv*cc;
  der(cd) = k3*ca^2 - sv*cd;

end CSTR_transient;
```

Figure 14. Modelica definition of the CSTR system that includes transient effects.

Simulating this for 1 second demonstrates these trajectories.

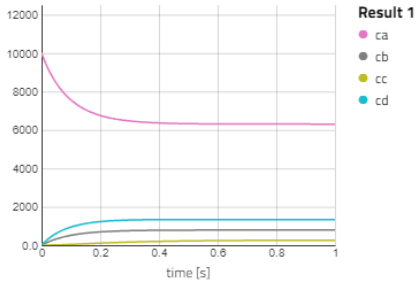


Figure 15. Species concentrations (vertical axis) transient response of the CSTR system versus time (horizontal axis)

Notice that the `generic_pyomo_model` function is used to define the problem statement. We also allow the simulation to proceed longer, to ensure that steady state conditions have been reached. Alternatively, we could have elaborated on the Modelica model to terminate the simulation after acceptable steady-state conditions have been reached. The problem statement remains the same as the other cases and the results are also comparable to the static cases.

```
50    fmu = load_fmu('CSTR_transient.fmu')
51
52    # create a block to store the external reactor model
53    m = generic_pyomo_model(
54        problem=fmu,
55        input_list=['sv'],
56        output_list=['cb'],
57        final_time=100,
58        csv_log_file=csv_log_file,
59        modifiers={'caf':10000.0, 'k1':0.83333, 'k2':1.6666, 'k3':0.00016666}
60    )
61
62    # Not strictly necessary but just to show they can be changed if desired
63    mdl = m.block.get_external_model()
64
65    mdl.input_vars['sv'].initial = 5
66    mdl.output_vars['cb'].initial = 1
67    mdl.output_vars['cb'].nominal = 1000
```

Figure 16. Transient CSTR optimization problem statement

Now that we have demonstrated the use of the FMU/Pyomo interface package with the CSTR model, we can use this for larger models. The results are shown for two different models. The first is a model from Modelon's Jet Propulsion Library that simulates a steady-state operating gas turbine engine. The sizing of the engine is designed to meet the requirements for an example aircraft under the conditions of a rolling-take-off (RTO), top-of-climb (TOC), and cruise (CRZ) conditions. These requirements are constraints within the model and simultaneous evaluation of these three cases relies on three instances of the engine component within a single simulation model. This is based on the multi-point methodology defined by Kyprianidis et al. (2014). The optimization problem is to minimize the specific fuel consumption under cruising conditions.

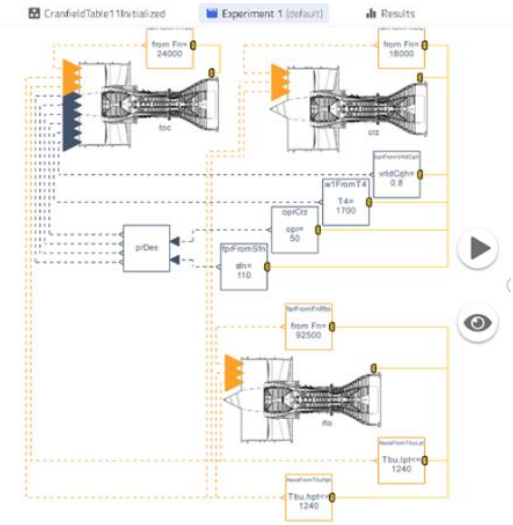


Figure 1. The Steady-state gas-turbine design model used for testing.

Sweeping some of the variables of interest and plotting the specific fuel consumption for the cruising case shows the local minimum (the solution that the solver will search for) occurs between 0.94 and 0.96 of the swept variable.

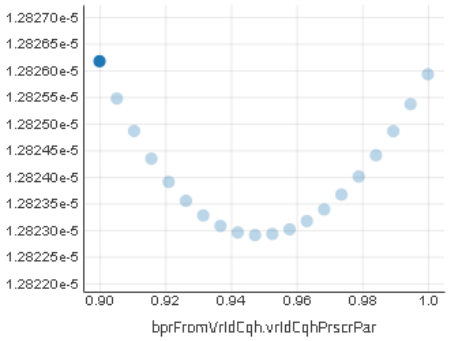


Figure 17. Specific fuel consumption (vertical axis) versus swept by-pass ratio constraint on the velocity ratio tuning variable (horizontal axis)

Defining the optimization problem is the same as for the CSTR system but with specifics for the FMU and its variables. The variables to tune are the cold-to-hot air velocity ratio, the overall pressure ratio of the engine, and the specific thrust. The nominal values for these (used for the normalization) are 1, 50, and 100 respectively.

```
name = 'GearedTurbofan_MultiPoint_Cranfield_Table11Initialized.fmu'
fmu = load_fmu(name)
m = static_pyomo_model(
    problem=fmu,
    input_list=[
        "bprFromVrIdCqh.vrIdCqhPrscr",
        "oprCrz.oprPrscr",
        "fprFromSfn.sfnPrscr"
    ],
    output_list=["crz.summary.SFC"],
    csv_log_file=csv_log_file
)
if modify_attributes: ···

m.obj = pyo.Objective(
    expr=m.block.outputs['crz.summary.SFC'],
    sense=pyo.minimize
)
```

Figure 18. Gas turbine specific fuel consumption minimization problem statement

The total problem involves 112 variables. Most of the iteration variables are the constraints needed to solve the design problem that requires the engine to meet the specific performance requirements. The solver converges to the solution within about 60 iterations.

The last model described in this paper defines an open-cycle parallel double-effect absorption heat pump (DEAHP) coupled to a multi-effect distillation (MED) system for brine desalination. In this system, the DEAHP acts as a steam recompression unit that improves the coupled system's overall efficiency proportionally to the coefficient of performance of heating (COPh).

The optimization problem is to maximize the COPh of the DEAHP by varying the strong and weak mass fraction concentrations of the absorption fluid `Xstrong` and `Xweak`, respectively.

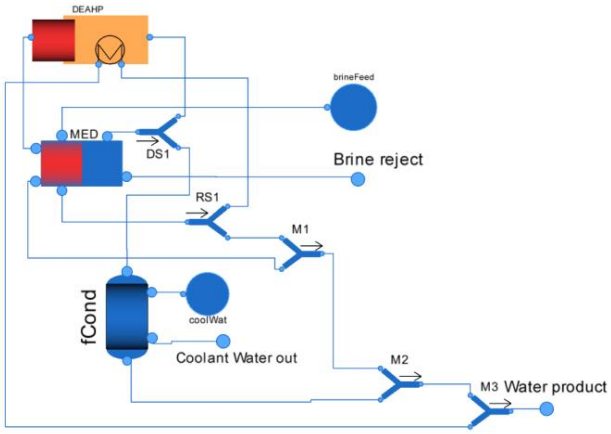


Figure 2. The Steady-state design model for a double effect absorption heat pump from (Stuber et al., 2015)

Using the new FMU/Pyomo interface package to define an optimization problem to solve with Ipopt results in convergence to a solution of 0.43 and 0.25 for `Xstrong` and `Xweak`, respectively with a COPh of 2.66. In this test, the starting values for `Xstrong` and `Xweak` were 0.588 and 0.549, respectively.

Sweeping the concentrations in simulation and observing the COPh can be used to verify the solution. Notice there is a local maximum in `Xstrong` and a maximum at the lower boundary of `Xweak`. This is consistent with the solution found by Ipopt.

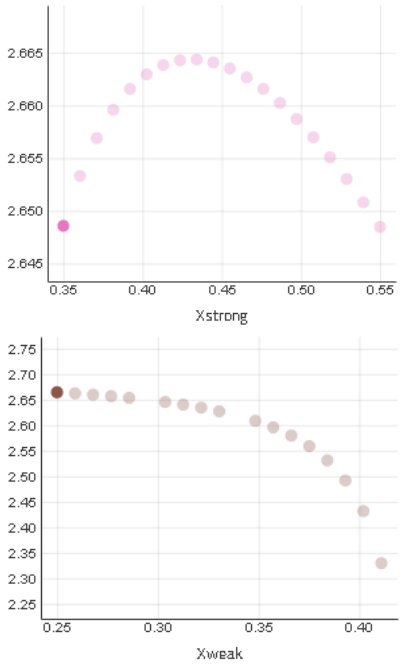


Figure 19. Example sweep of the strong and weak concentrations (horizontal axes) and the resulting coefficient of performance (vertical axes) that demonstrates the optimization problem's solution.

Table 1 lists some statistics related to the different models described in this paper. The key for the rows is shown in Table 2.

Table 1. Statistics per model.

CSTR Python	*CSTR FMU*	*Gas Turbine*	*DEAHP*
4	4	2382	621
4	1	109	30
1	1	3	2
13	16	60	24

Table 2. Row key for Table 1

1	Model implementation
2	Variable count
3	Equality constraint count
4	Optimization (tuner) variable count
5	Optimization iteration count

6 Conclusions

The Pyomo interface package has been used to solve both small and large optimization problems. There are two additional findings from this work. The first is the sensitivity of success to converge to a solution, on the

 10.3384/ECP207109 DOI

Jacobian. As described in the **Test Models and Results** section, small changes in the method used to estimate or compute the Jacobian had a significant effect on the ability of the solver to find a solution. This was demonstrated with a small example problem but is especially true as the models become larger.

Another significant finding was the importance of normalizing the variables. Even small models can benefit from this. Using the nominal attributes to normalize variables balances the sensitivity of individual variables so they do not dominate the convergence criteria. The current implementation uses the nominal attributes of the FMU variables automatically to normalize both the outputs and the Jacobian (using nominal attributes of both inputs and outputs).

Finally, models that can produce failed simulations or solutions can reduce the robustness of the solver. The current implementation returns a Numpy `NaN` value to the optimization solver in these cases. A possible improvement to this is proposed in the next section.

7 Future Work

Future developments for this work include:

- Native support for parameter tuning (including calibration) with respect to experimental data
- Support for dynamic optimization
- Sensitivity analysis to help the user identify tuner variables
- Improved support for failed solutions
- Testing and support for additional problem types and solvers (in addition to Ipopt that was used for this current work)

In addition to the above list of improvements to the Pyomo Interface Package, there is one item related to FMU creation. The Pyomo Interface Package natively supports FMUs generated with Modelon's steady-state interface for solving NLP problems. A proposed extension, similar to the steady-state interface, would be automatic conversion of mathematical operators with restricted domains into a more optimization friendly format. As described in Wächter 2009, mathematical functions with restricted domains can usually be converted into inequality constraints. It should be possible for Modelica translators to make this conversion automatically. This would make it easy for model developers to write the equations in a familiar format for normal simulation but export the FMU targeted for optimization when needed, similar to exporting an FMU for steady state evaluation.

Disclaimer

The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

Acknowledgements

This material is based upon work supported by the National Alliance for Water Innovation (NAWI), funded by the U.S. Department of Energy, Office of Energy Efficiency and Renewable Energy (EERE), Industrial Efficiency and Decarbonization Office, under Funding Opportunity Announcement DE-FOA-0001905.

A special thanks to Professor George Bollas from the University of Connecticut for also supporting this work.

References

Åkesson, Johan and K-E Årzén, Magnus Gäfvert, Tove Bergdahl, Hubertus Tummescheit (2011). "Modeling and optimization with Optimica and JModelica. org—Languages and tools for solving large-scale dynamic optimization problems", Computers & Chemical Engineering, Volume 34, Issue 11, pp. 1747-1849.

Andersson, Joel and Johan Åkesson, Francesco Casella, Moritz Diehl (2011). "Integration of CasADi and JModelica.org". In: Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical University; Dresden; Germany, pp. 218-231. DOI: 10.3384/ecp11063218.

Baharev, Ali and Hermann Schichl, Arnold Neumaier (2016). "Tearing systems of nonlinear equations I. A survey." URL: https://api.semanticscholar.org/CorpusID:51987111

Bachmann, Bernhard and Lennart Ochel, Vitalij Ruge, Mahder Gebremedhin, Peter Fritzson, Vaheed Nezhadali, Lars Eriksson, and Martin Sivertsson (2012). "Parallel multiple-shooting and collocation Optimization with OpenModelica". In: Proceedings of the 9th International Modelica Conference. Linköping University Electronic Press, September 2012, pp. 659-668. DOI:10.3384/ecp12076659.

Bryson, Arthur E. Jr. (1999), Dynamic Optimization, Addison Wesley Longman, Inc. ISBN: 0-201-59790-X.

Bynum, Michael L. and Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Siirola, Jean-Paul Watson, David L. Woodruff (2021). "Pyomo – Optimization Modeling in Python" 3rd Edition, 2021, ISSN 1931-6828.

Fletcher, R (1987). "Practical Methods of Optimization", 2nd ed. John Wiley and Sons. ISBN: 0-471-49463-1.

Kyprianidis, Konstantinos G. and Andrew M. Rolt, Tomas Grönstedt (2014). "Multi-Disciplinary Analysis of a Geared Fan Intercooled Core Aero-Engine", Journal of Engineering for Gas Turbines and Power, January 2014

Magnusson, Fredrick and Johan Åkesson (2015). "Dynamic Optimization in JModelica.org". In: Processes 2015, 3, pp. 471-496; DOI:10.3390/pr3020471.

Modelica Association (2024), FMI website, https://fmi-standard.org.

Modelon Impact Help Center (2024), website URL: https://help.modelon.com/latest/reference/oct/#dynamic-optimization-of-daes-using-direct-collocation-with-casadi.

OpenModelica (2024) website URL: https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/optimization.html.

Rodriguez, Jose Santiago and Michael Bynum, Carl Laird, Bethany Nicholson, Robby Parker, John Siirola (2024). "PyNumero is a package for developing parallel algorithms for nonlinear programs", https://pyomo.readthedocs.io/en/stable/contributed_packages/pynumero/index.html

Stuber, Matthew D. and Christopher Sullivan, Spencer A. Kirk, Jennifer A. Farrand, Philip V. Schillaci, Brian D. Fojtasek, Aaron H. Mandell (2015). "Pilot demonstration of concentrated solar-powered desalination of subsurface agricultural drainage water and other brackish groundwater sources" Desalination, Volume 355, 2015, Pages 186-196, https://doi.org/10.1016/j.desal.2014.10.037.

Ruge, Vitalij and Willi Braun, Bernhard Bachmann, Andrea Walther, and Kshitij Kulshreshtha (2014). "Efficient implementation of collocation methods for optimization using OpenModelica and adol-c". In: Proceedings of the 10th International Modelica Conference. Modelica Association and Linköping University Electronic Press, March 2014, pp. 1017-1025. DOI:10.3384/ecp140961017.

Wächter, Andreas and L. T. Biegler (2006). "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming", Mathematical Programming, 106(1), pp. 25-57, preprint at http://www.optimization-online.org/DB_HTML/2004/03/836.html

Wächter, Andreas (2009). "Short Tutorial: Getting Started With Ipopt in 90 Minutes", In Combinatorial Scientific Computing. Dagstuhl Seminar Proceedings, Volume 9061, pp. 1-17, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2009), https://doi.org/10.4230/DagSemProc.09061.16

Development and Validation of a Water-to-Air Heat Pump Model Using Modelica

Yuhang Zhang[1] Mingzhe Liu[1] Zhiyao Yang[1] Caleb Calfa[1] Zheng O'Neill[1]

[1]J Mike Walker' 66 Department of Mechanical Engineering, Texas A&M University, USA,
{yuhang.zhang,mingzhe.liu,z.yang,cjcalfa,zoneill}@tamu.edu

Abstract

Water-to-air heat pumps are widely used Heating, Ventilation, and Air Conditioning (HVAC) devices due to their versatility and energy efficiency. However, there is a scarcity of readily available Modelica models that support reversible operation (heating and cooling modes), use compressor speed as the control signal, and accurately predict the system performance. To address this gap, this paper presents a speed-input water-to-air heat pump model developed using Modelica. Performance curves are employed to represent the functionality and predict the system's capacity and power usage. To validate the proposed model's effectiveness, manufacturer-provided data are used to generate the performance curves. The model, based on these curves, is then used to simulate testing conditions, which are implemented in a real heat pump testbed. The comparison between simulated and measured values shows that the errors during normal operation stages are within an acceptable range, demonstrating the effectiveness of the developed water-to-air heat pump model.

Keywords: Modelica, Water-to-air heat pump, Performance curve, Validation

1 Introduction

Water-to-air heat pumps are versatile HVAC devices capable of providing both heating and cooling by transferring heat between water and air. They have gained wide applications due to their energy efficiency, ability to reduce operating costs, and environmentally friendly characteristics. Unlike air-source heat pumps (ASHP) which exchange heat from the outside air, water source heat pumps (WSHP) use water as its heat source or heat sink to provide heating or cooling. The water source could be a lake, river, pond, groundwater, or a closed-loop system where water circulates through buried pipes. Because water has a higher heat capacity and generally maintains more stable and moderate temperatures compared to air, WSHPs tend to be more energy-efficient and weather-independent. In addition, WSHPs are favored for their quiet operation and long lifespan (Chua, Chou, and Yang 2010; Gaur, Fitiwi, and Curtis 2021). These characteristics make WSHPs become a good choice for both residential and commercial buildings, especially where a reliable water source is available or where ground temperatures are stable.

In particular, heat pump-based heating and cooling systems are increasingly being integrated into building-to-grid systems for electrifications. This integration allows buildings equipped with these heat pumps to participate in demand response programs, where the operation of the HVAC system can be adjusted based on the needs of the grid. During periods of peak demand, the heat pumps can reduce their load or shift their operation to off-peak hours, contributing to grid stability and enabling a better use of renewable energy sources. This capability not only provides economic benefits to building owners through incentives and reduced energy bills, but also supports the broader goal of creating a more resilient and sustainable energy infrastructure.

As modeling is a widely acknowledged tool for better understanding, analyzing, and applying heat pumps, numerous studies have been conducted on the development and improvement of heat pump models for various applications (Montagud, Corberán, and Ruiz-Calvo 2013; Baccoli, Mastino, and Rodriguez 2015; Huang et al. 2019). Among these, Modelica, an object-oriented, equation-based language, has been widely adopted in the modeling of heat pumps due to its ability to directly incorporate physical laws into models, facilitate reusable components, and support multi-domain simulations (Fritzson and Engelson 1998). For example, the Modelica Buildings library's Buildings.Fluid.HeatPumps.EquationFitReversible is built upon the curves-based model from EnergyPlus (Crawley et al. 2001) and mainly used for water to water heat pump; the control input of this model is the set point for the leaving fluid temperature (Wetter et al. 2014). The IDEAS library (Jorissen et al. 2018) developed by KU Leuven and 3E offers models of air-to-air and water-to-water heat pumps, but none of aforementioned heat pump models incorporate the compressor speed as a control variable for the heat pump with variable speed drivers, while it is anticipated that inverter-based heat pump will play more roles for electrifying buildings. Additionally, the AixLib (Maier et al. 2024) developed a generic grey-box heat pump model (`AixLib.Fluid.HeatPumps.HeatPump`) that employs empirical data for modeling the refrigerant cycle. This model enables the heat pump on/off control and inverter based reversible operation. However, this

model lacks support for water-to-air heat pump applications. The DLR ThermoFluid Stream Library (Zimmer, Meißner, and Weber 2022) provides robust modeling of complex thermofluid architectures including heat pumps. However, it focuses more on detailed vapor compression component modeling, which can be computationally inefficient for overall system performance. Moreover, this model has not yet been validated.

To the best of the authors' knowledge, there is no readily available and open-source variable-speed water-to-air heat pump model in Modelica that simultaneously meets the following requirements: 1) utilizes the compressor speed as a control signal, 2) seamlessly switches between heating and cooling modes, 3) predicts power usage and capacity, including both sensible and latent components. 4) has been validated using experiment measurements.

The development of such a model is crucial. Since the performance of variable-speed heat pump systems is directly influenced by compressor speed, this model enables more accurate simulations to predict overall system performance, including power usage and both sensible and latent heat capacity under various speed conditions. This is particularly important for control applications, where utilizing compressor speed as a control variable provides a more direct method of adjusting system performance. Modulating compressor speed directly affects the system's capacity and energy consumption, allowing for more precise management of the heat pump's operation. By incorporating compressor speed into the model, engineers can develop and test advanced control strategies that dynamically respond to changing conditions, thereby improving system efficiency and occupant comfort.

The lack of such a model also limits the ability to fully explore building-to-grid integration. A model with compressor speed control could be used to simulate the heat pump's behavior in response to grid signals, helping to design systems that can participate in demand response programs and contribute to grid stability. This integration is a key component of creating energy-efficient buildings that can interact seamlessly with the broader energy grid. In summary, developing a reversible water-to-air heat pump model that takes the compressor speed as a control signal in Modelica is not only necessary for improving the performance and efficiency of HVAC systems but also for advancing research in building energy management and grid integration. Therefore, in this study, a variable-speed reversible water-to-air heat pump model developed using Modelica is presented. The developed water-to-air heat pump model is based on the performance curve method and is capable of supporting the reversible operation (i.e., heating and cooling modes), taking the compressor speed as the control signal to facilitate the investigation of advanced control methods.

The paper is organized as follows: Section 2 introduces the model development process, beginning with a concise introduction to the major components and working principles of water-to-air heat pumps. It then details the Modelica implementation, including interfaces and performance curves. In Section 3, the validation process is described, where simulation results using the developed Modelica model are compared with measurements from a real heat pump testbed. In Section 4, the applicability of the developed model across various heat pump operating stages is discussed. Finally, Section 5 concludes this study.

2 Modelica Development

2.1 Overview of Water-to-Air Heat Pumps

A typical water-to-air heat pump consists of several key components: a compressor, an expansion valve, two heat exchangers (a refrigerant-to-water heat exchanger and a refrigerant-to-air heat exchanger), and a reversing valve. Among these, the compressor, expansion valve, and the two refrigerant heat exchangers are the primary components of the refrigeration cycle, which provides the essential functions of heating and cooling. The reversing valve, located between the compressor and the refrigerant-to-water heat exchanger, enables the reversal of refrigerant flow, allowing the system to switch between heating and cooling modes. Figure 1 illustrates the schematic of the heat pump's heating and cooling cycles. In heating mode, the heat pump extracts heat from the water side (acting as the evaporator) and releases heat into the air side (acting as the condenser), resulting in an increase in room temperature. Conversely, in cooling mode, the reversing valve reverses the refrigerant flow, causing the water side to function as the condenser and the air side to function as the evaporator. Heat is then transferred from the indoor air to the refrigerant, thereby lowering the indoor air temperature.

2.2 Modelica Implementation

Despite including all the aforementioned components in the water-to-air heat pump, the focus of this study is to predict the overall heat pump performance rather than the performance of individual components. Therefore, a vapor compression cycle with each component was not modeled. Instead, performance curves are used to represent the functionality of water-to-air heat pumps due to their

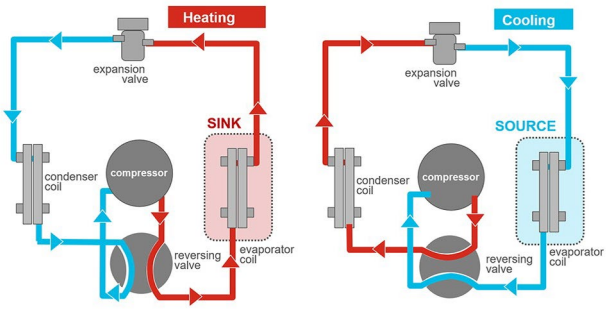


Figure 1. Schematic of the heating and cooling cycles in a reversible water-to-air heat pump (Trane Technologies 2024a)

10.3384/ECP207119 DOI

simplicity in implementation and wide application (Ying Zhang et al. 2020).

The proposed model was developed based on the following assumptions:

1. The model uses heating/cooling/off signals and the compressor speed as control inputs.

2. The steady-state performance of the heat pump, including total capacity ($\dot{Q}$) and Energy Input Ratio (EIR), is computed using polynomial equations. These equations account for the mass flow fractions on both the water and air sides, the inlet temperatures on both sides, and the compressor speed ratio.

Figure 2 shows the Modelica diagram of the developed heat pump model. To simplify and expedite the modeling process, DX coil models (`Buildings.Fluid.DXSystems`) from the Modelica Buildings Library (version 10.0.0) are reused and modified to develop the heat pump model in this study. (Wetter et al. 2014). Two data records (`datCoiHea` and `datCoiCoo`) are included in the model to record the nominal values and performance curves for heating mode and cooling mode, respectively.

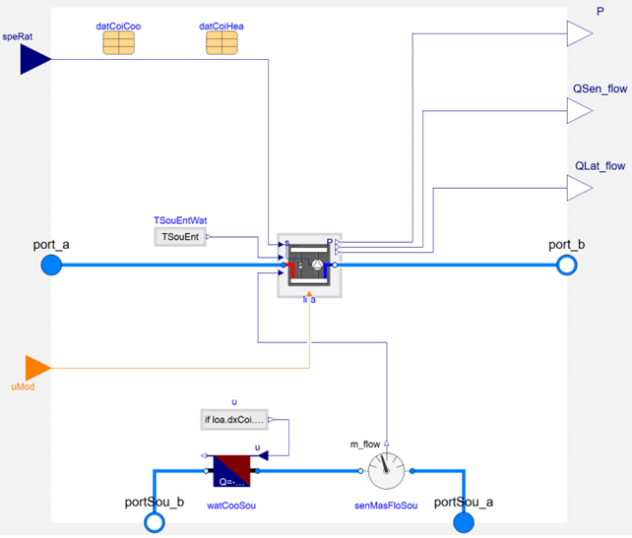


Figure 2. Diagram of the heat pump model in Dymola.

Table 1 lists all the connectors of the developed heat pump model. The model inputs include system operating mode (`uMod`), compressor speed (`speRat`). Both water-side and airside inlet information, such as flow rates, temperatures, and humidity ratios, are read from fluid ports. The model then calculates the real-time system capacity and power usage based on this information. These outputs can also be read from the interfaces shown in Figure 2.

The proposed model calculates the steady-state total capacity ($\dot{Q}$) and energy input ratio (EIR) at off-designed conditions based on Equation 1 and Equation 2, respectively. Both these equations are the product of functions

Table 1. Connectors of the developed heat pump model.

Type	*Name*	*Description*
FluidPort_a	`port_a`	Fluid connector for the inlet of the load side
FluidPort_b	`port_b`	Fluid connector for the outlet of the load side
FluidPort_a	`portSou_a`	Fluid connector for the inlet of the source side
FluidPort_b	`portSou_b`	Fluid connector for the outlet of the source side
input RealInput	`speRat`	Speed ratio [1]
input IntegerInput	`uMod`	Controlinput signal, cooling mode=-1, off=0, heating mode=+1
output RealOutput	`P`	Electrical power consumed by the unit [W]
output RealOutput	`QSen_flow`	Sensible heat flow rate of the load side [W]
output RealOutput	`QLat_flow`	Latent heat flow rate of the load side [W]

that account for changes in the inlet temperatures of both the source side (waterside) ($\theta_{\text{loa,in}}$) and load side (airside) ($\theta_{\text{sou,in}}$), changes in mass flow rates of both the source side ($\dot{m}_{\text{sou}}$) and load side ($\dot{m}_{\text{loa}}$) and compressor speed ratio ($speRat$).

$$\begin{aligned}\dot{Q}\left(\theta_{loa,in},\theta_{sou,in},ff_{loa},ff_{sou}\right) = \\ cap_{\theta}\left(\theta_{loa,in},\theta_{sou,in}\right)\times cap_{FFLoa}\left(ff_{loa}\right)\times \\ cap_{FFSou}\left(ff_{sou}\right)\times cap_{spe}\left(speRat\right)\times\dot{Q}\end{aligned} \quad (1)$$

$$\begin{aligned}EIR\left(\theta_{loa,in},\theta_{sou,in},ff_{loa},ff_{sou}\right) = \\ EIR_{\theta}\left(\theta_{loa,in},\theta_{sou,in}\right)\times EIR_{FFLoa}\left(ff_{loa}\right)\times \\ EIR_{spe}\left(speRat\right)/COP_{nom}\end{aligned} \quad (2)$$

where $\theta_{loa,in}$ is the inlet temperature of the load side (air-side), which is the dry-bulb air temperature if the coil is dry or the wet-bulb air temperature if the coil is wet. $\theta_{sou,in}$ is the inlet temperature of the source side (water-side). ff_{loa} is the normalized mass flow rate at the load side and is calculated as $\dot{m}_{loa}/\dot{m}_{loa,nom}$. ff_{sou} is the normalized mass flow rate at the source side and is calculated as $\dot{m}_{sou}/\dot{m}_{sou,nom}$. $\dot{m}_{loa,nom}$ is the nominal mass flow rate at the load side, and $\dot{m}_{sou,nom}$ is the nominal mass flow rate at the source side. The capacity modifiers cap_{θ}, cap_{FFLoa}, cap_{FFSou}, cap_{spe} and the EIR modifiers EIR_{θ}, EIR_{FFLoa}, EIR_{FFSou}, EIR_{spe} are functions of inlet temperatures, load side flow rate, source side flow rate, and compressor speed ratio, respectively. These modifiers can be calculated using Equation 3 to Equation 10:

$$cap_\theta(\theta_{loa,in},\theta_{sou,in}) = a_1 + a_2\theta_{loa,in} + a_3\theta_{loa,in}^2 + a_4\theta_{sou,in} + a_5\theta_{sou,in}^2 + a_6\theta_{loa,in}\theta_{sou,in} \quad (3)$$

$$cap_{FFLoa}(ff_{loa}) = b_1 + b_2 ff_{loa} + b_3 ff_{loa}^2 + b_4 ff_{loa}^3 + \dots \quad (4)$$

$$cap_{FFSou}(ff_{sou}) = c_1 + c_2 ff_{sou} + c_3 ff_{sou}^2 + c_4 ff_{sou}^3 + \dots \quad (5)$$

$$cap_{spe}(speRat) = d_1 + d_2 speRat + d_3 speRat^2 + \dots \quad (6)$$

$$EIR_\theta(\theta_{e,in},\theta_{c,in}) = e_1 + e_2\theta_{loa,in} + e_3\theta_{loa,in}^2 + e_4\theta_{sou,in} + e_5\theta_{sou,in}^2 + e_6\theta_{loa,in}\theta_{sou,in} \quad (7)$$

$$EIR_{FFLoa}(ff_{loa}) = f_1 + f_2 ff_{loa} + f_3 ff_{loa}^2 + f_4 ff_{loa}^3 + \dots \quad (8)$$

$$EIR_{FFSou}(ff_{sou}) = g_1 + g_2 ff_{sou} + g_3 ff_{sou}^2 + g_4 ff_{sou}^3 + \dots \quad (9)$$

$$EIR_{spe}(speRat) = h_1 + h_2 speRat + h_3 speRat^2 + \dots \quad (10)$$

The coefficients used in the above performance curves can be obtained by fitting the performance data provided by manufacturers or from on-site measurements. It should be noted that although the form of the performance curves is identical for heating and cooling operation modes, different coefficients should be used and fitted respectively.

3 Validation

3.1 Heat Pump Testbed

To validate the effectiveness of the developed heat pump model, measurement data were gathered from an actual heat pump testbed situated at Texas A&M University and then compared with the simulated values using the proposed heat pump model.

Figure 3 shows a photograph of the heat pump testbed. The heat pump used in the testbed is a 2-ton water-to-air heat pump. Load emulators are equipped within the testbed to emulate various testing conditions. Multiple sensors are installed to collect and monitor real-time inlet and outlet conditions for both the waterside and airside, compressor speed, and unit power consumption. A more detailed introduction to the testbed can be found in (Calfa et al. 2023). Table 2 shows part of the rated information from the product catalog (Trane Technologies 2024b).

3.2 Performance Curves from Manufacturer Datasets

The performance curves of the heat pump model are fitted using the dataset provided by the manufacturer, which includes various inlet conditions for both the waterside and airside. The range of each normalized dependent variable in the performance curves is shown in Table 3.

Table 2. Rated information of the studied heat pump model.

Parameter	*Value*
Water side flow rate [kg/s]	0.39
Air side flow rate [m3/s]	0.44
Cooling capacity [kW]	7.21
Cooling COP [-]	5.4
Heating capacity [kW]	8.88
Heating COP [-]	6.1

Table 3. Rated information of the studied heat pump model.

Variable	*Range*
Normalized water flow rate [-]	[0.65, 1]
Normalized air flow rate [-]	[0.46, 1]
Air inlet dry bulb temperature [°C]	[21.1, 32.2]
Air inlet wet bulb temperature [°C]	[11.3, 23.8]
Water inlet temperature [°C]	[7.2, 35]
Compressor speed ratio [-]	[0.5, 1]

The data sets are divided into two distinct subsets: training and test sets, with an 80:20 split ratio. The curve fitting process utilizes solely the data from the training set. The generalized least squares method is used to estimate the coefficients for the performance curves, as introduced in Section 2. Table 4 lists part of the estimated coefficients of performance curves as examples.

Following the training process, the fitted performance curves are tested on the test set to evaluate the fitting accuracy. Figure 4 presents the model performance for cooling and heating operations using the test set. The results indicate that the overall fitting of the heat pump's performance is satisfactory for both heating and cooling conditions, with most prediction errors falling within 15%. To further quantitatively evaluate the model performance, Normalized Mean Bias Error (NMBE) and Coefficient of Variation of the Root Mean Squared Error (CVRMSE) are used as performance metrics to reflect the error between the simulated and measured values. The equations to calculate NMBE and CVRMSE are shown in Equation 11 and Equation 12, respectively.

$$\text{NMBE} = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n \times \bar{y}} \times 100\% \quad (11)$$

$$\text{CVRMSE} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}}{\bar{y}} \times 100\% \quad (12)$$

where y_i are the observed values, $\hat{y}_i$ are the predicted values, $\bar{y}$ is the mean of observed values, and n is the number of observations.

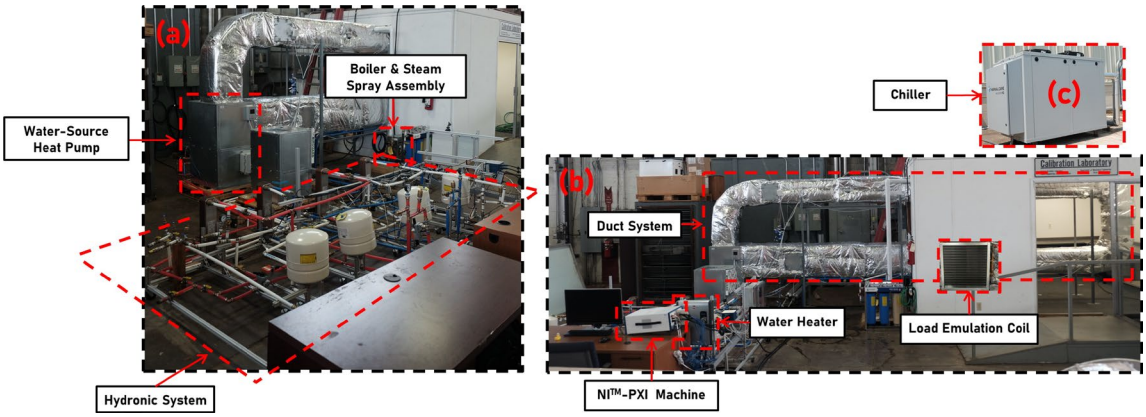


Figure 3. A photograph of the heat pump testbed (Calfa et al. 2023)

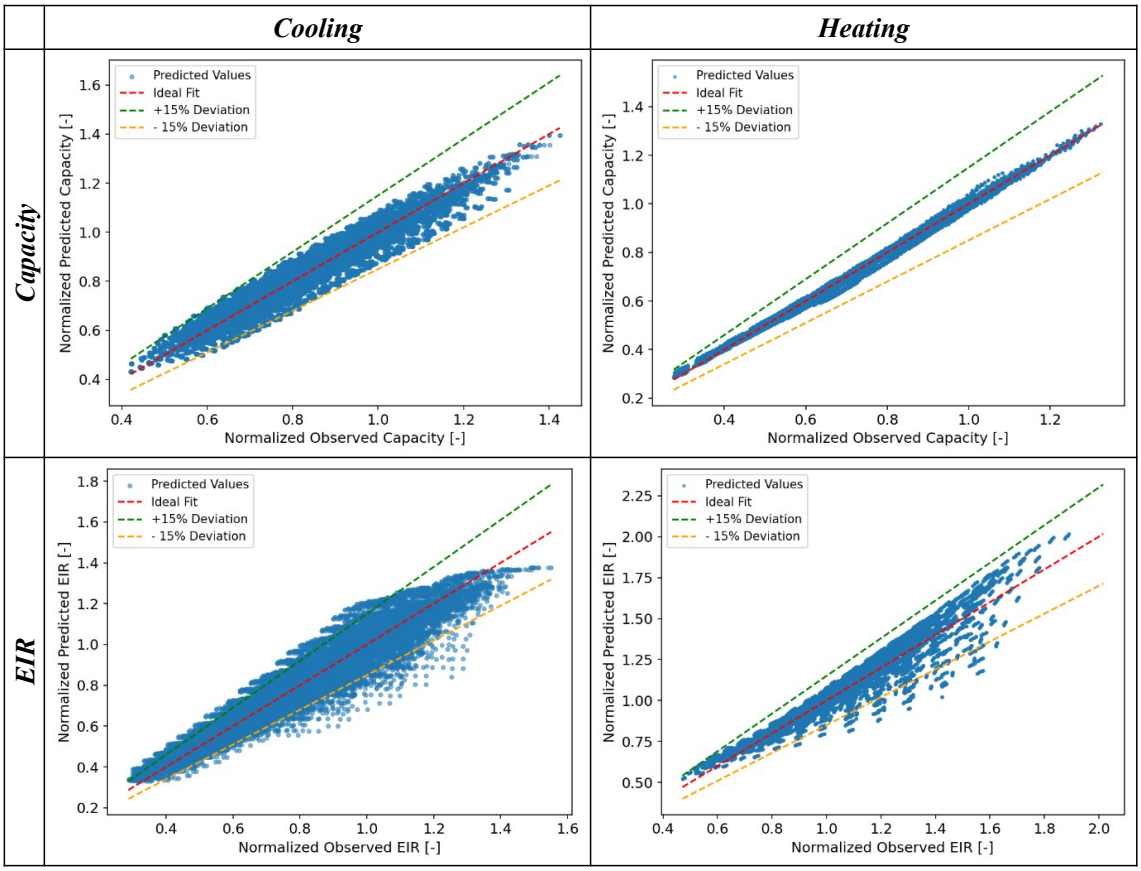


Figure 4. Prediction performance of WSHP performance curve on the testing dataset.

Table 4. Estimated coefficients of performance curves.

Performance curve	*Cooling*	*Heating*
$cap_\theta(\theta_{loa,in}, \theta_{sou,in})$	[0.4861, 0.03002, -2.128×10^{-4},-1.376×10^{-4}, -2.067×10^{-5}, -1.994×10^{-4}]	[0.4593, -0.0013051, -1.902×10^{-5}, 0.01901, 4.276×10^{-5}, -1.448×10^{-4}]
$cap_{FFLoa}(ff_{loa})$	[0.7686, 0.315]	[0.8935, 0.1463]
$cap_{FFSou}(ff_{sou})$	[0.9856, 0.01642]	[0.9213, 0.07482]
$cap_{spe}(speRat)$	[0, 1.628, -0.4200]	[0, 1.807, -0.1575]

Table 5 presents the performance metrics on the test set, demonstrating the accuracy of the predictive model. This model will then be applied to the actual measurement dataset for additional validation.

Table 5. Performance metrics on the test set.

	NMBE Cooling	*CVRMSE Cooling*	*NMBE Heating*	*CVRMSE Heating*
Capacity	0.039%	5.70%	0.159%	2.23%
EIR	0.121%	8.19%	0.039%	7.47%

3.3 Validation with Experimental Datasets

Various testing conditions are conducted in the testbed, and data are collected accordingly. After data preprocessing, such as excluding data points recorded during startup/shutdown cycles, a total of 311 data points are obtained for cooling mode and 181 data points are obtained for heating mode. Figure 5 uses compressor speed as an example to illustrate the data distribution for both manufacturer and measurement data. For each data point, the monitored conditions as listed in Table 1 are used as inputs for the developed Modelica heat pump model. After the simulations, the model outputs are compared with the measurements. The compared values include heating/cooling capacity, EIR and electricity power. Figure 6 shows comparisons between simulated and measured values for capacity, EIR, and power in cooling and heating modes, respectively. Table 6 presents the calculated performance metrics.

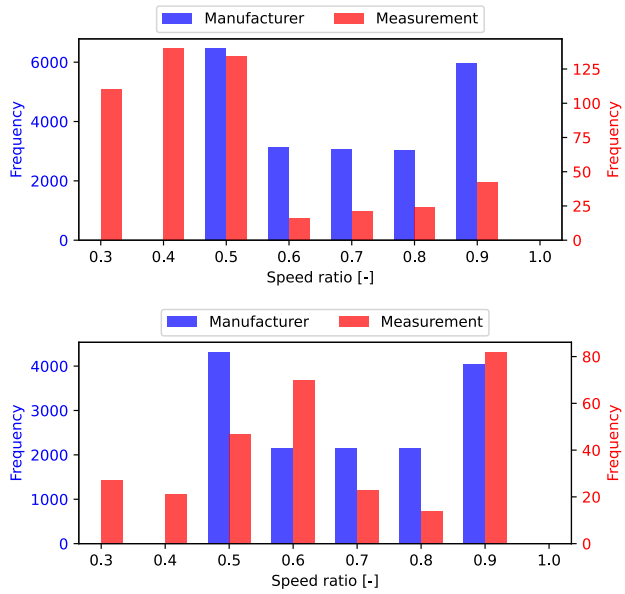


Figure 5. Data distribution of compressor speed ratio for both manufacturer and measurement data in cooling (top) and heating (bottom) modes.

Table 6. Performance metrics on the measurement set.

	NMBE Cooling	*CVRMSE Cooling*	*NMBE Heating*	*CVRMSE Heating*
Capacity	1.09%	3.77%	4.89%	8.22%
EIR	1.24%	-5.33%	3.96%	9.77%
Power	2.96%	-1.32%	5.67%	5.85%

From the comparison results, it can be observed that the simulated values using the developed model generally align with the observed values, and the error metrics are within an acceptable range, validating the effectiveness of the developed reversible water-to-air heat pump model. The discrepancy between the simulated and measured values might arise from the following sources:

1. Model Simplifications and Assumptions: The proposed heat pump model uses simplified performance curves to represent its performance. Although the results prove its effectiveness, some inevitable model errors arise from this simplification.

2. Measurement Errors: Due to sensor uncertainties, measurement values can have errors, leading to discrepancies. This error can be mitigated by using highly accurate sensors.

4 Discussions

This section evaluates the effectiveness of the developed WSHP model by comparing continuous testing conditions gathered from two full days of operation—one focused on cooling and the other on heating. Unlike the validation section which only considers normal steady-state running stages, the testing conditions here encompass all heat pump operation stages, including startup, normal steady-state running, and shutdown phases. Data were sampled at 5-second intervals, with a rolling average applied—5 minutes for the heating day and 3 minutes for the cooling day—resulting in 288 testing points for cooling and 480 for heating. The model required inputs detailed in Table 1 are obtained from onsite measurements and then fed into the developed model. The simulation outputs are subsequently compared with the measured values, specifically focusing on system capacity as a comparison example. Figure 7 presents the comparison results for two days with cooling and heating operations, respectively. The figure uses a yellow background to denote continuous steady-state operation stages and a pink background to highlight other stages, including startup, shutdown cycles, and off periods.

From Figure 7, it can be observed that during normal steady-state operation periods, the predicted system capacity closely matches the measured values. However, during startup and shutdown cycles, the model fails to capture the transient variations for the system capacity, leading to discrepancies between the predicted and measured

10.3384/ECP207119 DOI

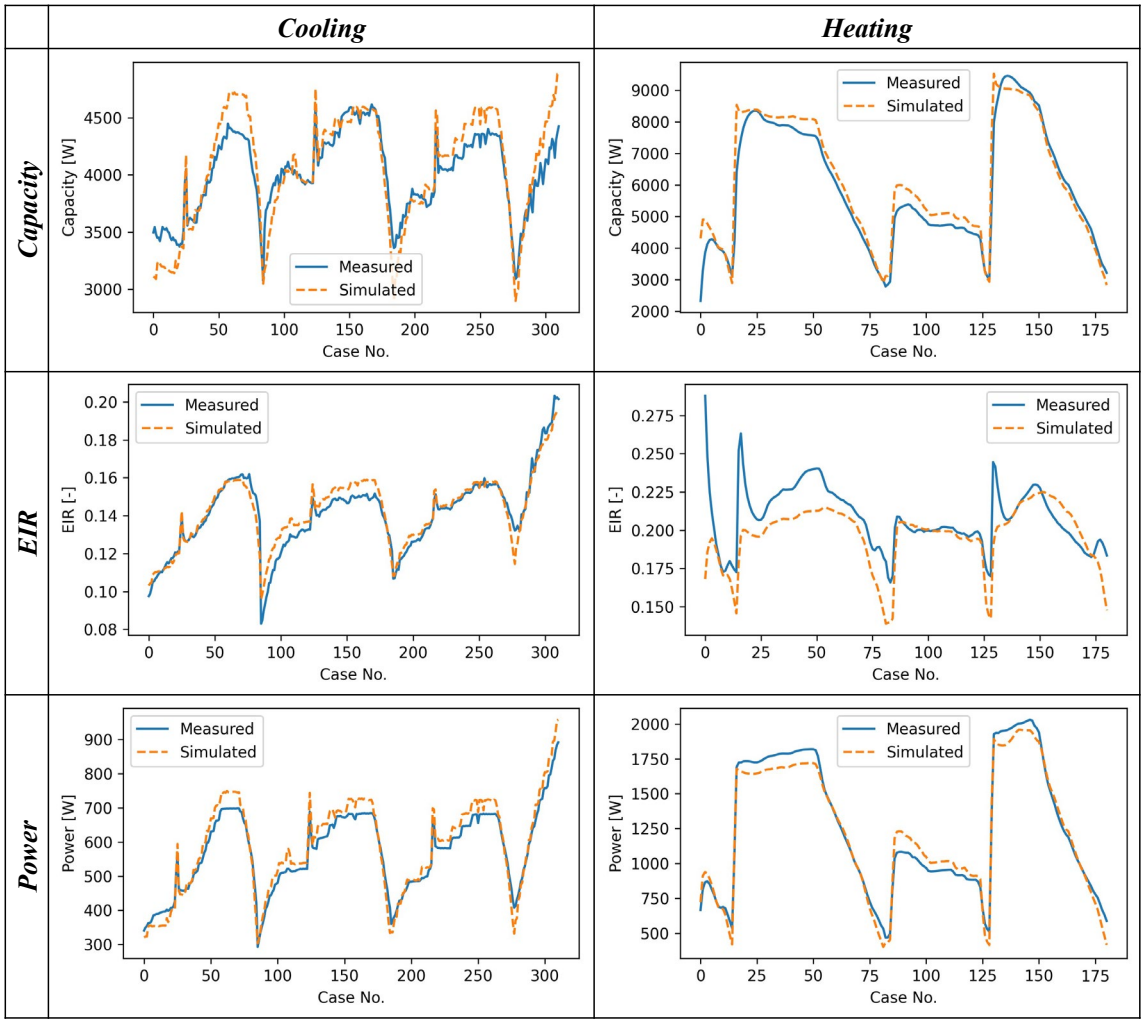


Figure 6. Comparison between simulated and measured values for system capacity, EIR, and power.

values. The comparison results demonstrate the application restrictions of the proposed heat pump model. As the performance curves adopted in the proposed model are only valid during stable operation conditions, this model is not suitable for simulating transient heat pump performance, such as startup and shutdown cycles. Therefore, other modeling approaches should be considered if transient behaviors are expected. In particular, behaviors from Pulse Width Modulation (PWM) to adjust the system capacity need to be incorporated.

5 Conclusions and Future Work

In this paper, a new inverter-based variable-speed water-to-air heat pump model developed in Modelica is presented. This model is expanded and modified based on the DX coil model provided in the Modelica Buildings Library. Using multiple performance curves to represent the overall functionality of the heat pump, it is able to simulate the heat pump's total capacity and power usage under different operational modes (heating/cooling) and variable speed scenarios. The effectiveness of the proposed model is validated through a comparison between simulated values and measurements from a real heat pump testbed. The simulated capacity, EIR, and power correspond well with measurements for both heating and cooling conditions, demonstrating the model's capability in predicting variable-speed heat pump performance.

Future work in the following aspects can be considered:

- Improving Sensible Heat Ratio (SHR) Prediction: Although the current model performs well in predicting the total cooling capacity, the simulated sensible heat ratio does not align well with measurement values. SHR, which describes the ratio of sensible heat load to total heat load, needs further refinements to improve the model performance.

- Extending Application Scenarios: The proposed heat pump model can be applied to additional scenarios, such as building-to-grid systems or district heat pump systems (Yuhang Zhang et al. 2024). Further verification of its effectiveness in these contexts is needed.

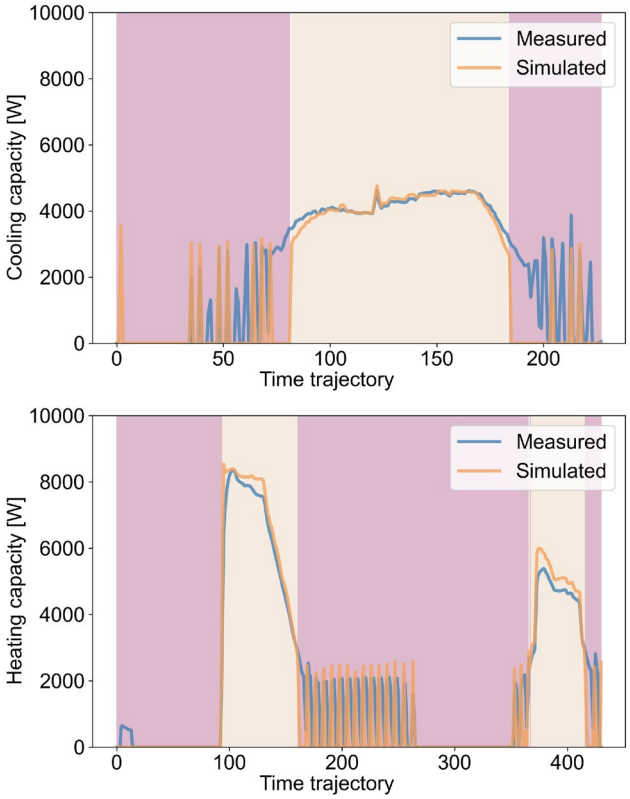


Figure 7. Comparisons between simulated and measured system capacity over two whole days (top: cooling, bottom: heating).

Acknowledgements

This work is partly supported by the National Science Foundation (2309030). The authors would also like to acknowledge the support and assistance provided by the manufacturer Trane.

References

Baccoli, Roberto, Costantino Mastino, and Giuseppe Rodriguez (2015). "Energy and Exergy Analysis of a Geothermal Heat Pump Air Conditioning System". In: *Applied Thermal Engineering* 86, pp. 333–347. DOI: 10.1016/j.applthermaleng.2015.03.046.

Calfa, Caleb et al. (2023). "Performance Assessment of a Real Water Source Heat Pump within a Hardware-in-the-Loop (HIL) Testing Environment". In: *Science and Technology for the Built Environment* 29.10, pp. 1011–1026. DOI: 10.1080/23744731.2023.2261810.

Chua, K. J., S. K. Chou, and W. M. Yang (2010). "Advances in Heat Pump Systems: A Review". In: *Applied Energy* 87.12, pp. 3611–3624. DOI: 10.1016/j.apenergy.2010.06.014.

Crawley, Drury B. et al. (2001). "EnergyPlus: Creating a New-Generation Building Energy Simulation Program". In: *Energy and Buildings*. Special Issue: BUILDING SIMULATION'99 33.4, pp. 319–331. DOI: 10.1016/S0378-7788(00)00114-6.

Fritzson, Peter and Vadim Engelson (1998). "Modelica — A Unified Object-Oriented Language for System Modeling and Simulation". In: *ECOOP'98 — Object-Oriented Programming*. Ed. by Eric Jul. Berlin, Heidelberg: Springer, pp. 67–90. ISBN: 978-3-540-69064-1. DOI: 10.1007/BFb0054087.

Gaur, Ankita Singh, Desta Z. Fitiwi, and John Curtis (2021). "Heat Pumps and Our Low-Carbon Future: A Comprehensive Review". In: *Energy Research & Social Science* 71, p. 101764. DOI: 10.1016/j.erss.2020.101764.

Huang, Shifang et al. (2019). "Performance Comparison of a Heating Tower Heat Pump and an Air-Source Heat Pump: A Comprehensive Modeling and Simulation Study". In: *Energy Conversion and Management* 180, pp. 1039–1054. DOI: 10.1016/j.enconman.2018.11.050.

Jorissen, Filip et al. (2018). "Implementation and verification of the IDEAS building energy simulation library". In: *Journal of Building Performance Simulation* 11.6, pp. 669–688.

Maier, Laura et al. (2024). "AixLib: an open-source Modelica library for compound building energy systems from component to district level with automated quality management". In: *Journal of Building Performance Simulation* 17.2, pp. 196–219.

Montagud, Carla, José Miguel Corberán, and Félix Ruiz-Calvo (2013). "Experimental and Modeling Analysis of a Ground Source Heat Pump System". In: *Applied Energy* 109, pp. 328–336. DOI: 10.1016/j.apenergy.2012.11.025.

Trane Technologies (2024a). *Ascend® Air-to-Water Heat Pump Model ACX*. https://www.trane.com/commercial/north-america/us/en/products-systems/chillers/air-cooled-chillers/ascend-air-to-water-heat-pump.html.

Trane Technologies (2024b). *Axiom™ Horizontal and Vertical Water Source Heat Pumps*. https://www.trane.com/commercial/north-america/us/en/products-systems/packaged-units-and-split-systems/water-source-heat-pumps/high-ef-horizontal-vertical-wshp.html.

Wetter, Michael et al. (2014). "Modelica Buildings Library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Zhang, Ying et al. (2020). "Study on Model Uncertainty of Water Source Heat Pump and Impact on Decision Making". In: *Energy and Buildings* 216, p. 109950. DOI: 10.1016/j.enbuild.2020.109950.

Zhang, Yuhang et al. (2024). "Temperature Control Strategies for Fifth Generation District Heating and Cooling Systems: A Review and Case Study". In: *Applied Energy* 376, p. 124156. DOI: 10.1016/j.apenergy.2024.124156.

Zimmer, Dirk, Michael Meißner, and Niels Weber (2022). "The DLR ThermoFluid Stream Library". In: *Electronics* 11.22, p. 3790. DOI: 10.3390/electronics11223790.

A Modelica Implementation of an Organic Rankine Cycle

Hongxiang Fu[1] Ettore Zanetti[1] Jianjun Hu[1] David Blum[1] Michael Wetter[1]

[1]Building Technology and Urban Systems Division, Lawrence Berkeley National Laboratory, USA,
{hcasperfu,ezanetti,jianjunhu,dhblum,mwetter}@lbl.gov

Abstract

Organic Rankine cycle (ORC) systems generate power from low-grade heat sources, such as geothermal sources and industrial waste heat. A key feature is that a working fluid is selected to match the temperature of the source. With the vast pool of candidate working fluids comes the challenge of developing a large number of robust thermodynamic media models. We implemented a subcritical ORC model in Modelica that uses working fluid data records and interpolation schemes in lieu of thermodynamic medium evaluation for energy recovery estimation. This is a component model that can be integrated into a larger energy system model. It does not require detailed thermodynamic, heat transfer, or machine analysis. Our ORC model fills a gap where working fluids are ready to choose or easy to add, and at the same time can be integrated into an energy system.

Keywords: organic Rankine cycle, media model, component model

1 Introduction

Organic Rankine cycle (ORC) systems have been an important waste heat recovery technology used to generate power from low-grade heat sources, such as geothermal sources and industrial process waste heat. It is particularly valuable to building and district energy applications, because in these areas both the electric power generated by the expander and heat rejected from the condenser can be used. Under the current background of decarbonisation (U.S. DOE 2024), it is valuable to model such systems for utilisation of renewable energy in district energy systems design.

ORC systems typically use a working fluid whose boiling point is matched to a specific waste heat source (U.S. DOE 2021). This versatility also poses a significant challenge: Developing robust and computationally efficient medium models for a wide range of candidate working fluids is a time-consuming and complex task. To accommodate a wide range of heat sources, including geothermal sources at 80°C to biomass sources at 500°C, there can be hundreds of potentially suitable substances (Bao and Zhao 2013). Studies on working fluid selection routinely examined tens of candidates. For example Saleh et al. (2007) investigated 31 pure fluids, and the number goes up substantially if mixtures were considered due to the possibility of combinations and mixing ratios (Abadi and Kim 2017).

There are existing open-source Modelica libraries that support modelling of thermodynamic cycles. The DLR ThermofluidStream library (Zimmer 2020; Zimmer, Meißner, and Weber 2022) has five refrigerant models that can be used for ORC (as of Version 1.1.0). It provides a helpful medium model template, but defers the implementation of additional robust models to the user, which remains challenging for non-experts. The ThermoPower library (Casella and Leva 2005) provides component models as well as control blocks suitable for thermodynamic cycle and system modelling. However, it is not specifically geared towards ORC modelling and additional component and medium models are needed. The ThermoCycle library (Quoilin, Desideri, et al. 2014; Oliveira, Iten, and Matos 2022) has Rankine and Brayton cycle models with detailed components such as heat exchangers and turbines, as well as control blocks. However, it only supports the steam cycle and not the ORC. It also requires external dependency for the medium models through the Modelica ExternalMedia library (Modelica 3rd-party libraries 2023) to the open-source software CoolProp (Bell et al. 2014) and the commercial software FluidProp (Asimptote 2023).

Multi-phase fluid property computations needed for ORC models are numerically challenging because of sharp derivative changes at phase transition. Naïve integration with detailed medium models can lead to long computation times and convergence problems. Literature has reported that, using professional software such as the Modelica ExternalMedia library for medium property computations was not enough. Interpolation methods at the phase transitions led to one order of magnitude shorter simulation time (Quoilin, Van Den Broek, et al. 2013; Twomey 2016).

Because of these challenges, it is understandable that we have not found an open-source ORC model with a collection of medium models that do not require use of an external code for media calculations. This is a drawback because selecting a working fluid with properties matching the waste heat source is a key step in the ORC system design. The following gap exists in the literature: Studies with a large pool of working fluids are usually limited at thermodynamic analysis; in the meantime, studies that performed detailed machine analysis or control designs often either only used one specific working fluid or resorted to external code for medium models. We there-

fore report a Modelica implementation of the ORC model with energy system integration that comes with ready-to-use and easy-to-add working fluid models to fill this gap. Our method is based on data records of working fluids converted from CoolProp and are used together with interpolation schemes in lieu of detailed thermodynamic fluid property computation. It improves upon existing open-source models we found in the literature as follows: First, all code is contained in one standalone Modelica library. Because there is no need to link to external code, usability and compatibility are improved. Second, unlike developing full-fledged media models, adding more media records to the package is easy for users, which suits the nature of ORC system design, allowing consideration of a broad pool of candidate working fluids. We have currently implemented ten fluids as listed in Table 2. We selected CoolProp as the source of fluid properties because it is open-source and offers wrappers for various languages and environments. It is important to note that any software that provides thermodynamic fluid properties can be used to generate the data records. Furthermore, because the property data are stored in Modelica records once generated, the choice of fluid property sources becomes irrelevant.

The vast pool of ORC working fluids manifests the versatility and also challenges of ORC modelling. Our method circumvents this challenge by using specialised and interpolation-based medium models that forego computationally challenging thermodynamic property evaluations. The result is a fast and robust model to obtain energy recovery estimation for an ORC component that can be integrated into an energy system for system-level design and analysis.

2 System Description

We consider a subcritical organic Rankine cycle as a bottoming cycle to recover energy from a hot fluid stream. Figure 1 is its concept schematic.

The system is not controlled to track any load, electric or thermal, and all generated power is assumed to be consumed and heat dissipated. The working fluid evaporating temperature $T_{w,eva}$ is a user-specified parameter. This is in line with the optimisation results reported by Quoilin, Aumann, et al. (2011) and Imran et al. (2020) that keeping a constant $T_{w,eva}$ was a common and proper control strategy for small-scale ORCs.

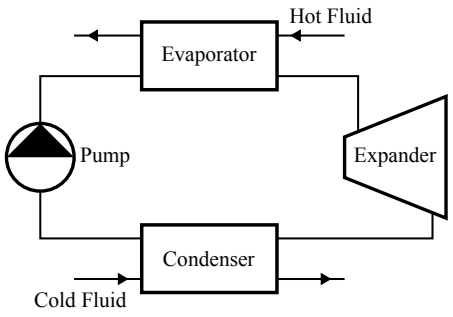


Figure 1. Schematic of the modelled ORC system.

The heat source is variable in terms of temperature and flow rate and an ORC system needs to accommodate to that. To achieve this, the working fluid flow rate $\dot{m}_w$ is controlled to maintain the evaporator pinch point temperature difference $\Delta T_{pin,eva}$. The following constraints are in place:

- The mass flow rate $\dot{m}_w$ will not go higher than a set upper limit. Rather, $\dot{m}_w$ stays at the user-specified upper limit and $\Delta T_{pin,eva}$ increases beyond its set point. This may happen when the incoming hot fluid has a high flow rate or a high incoming temperature, i.e., it carries more energy than the cycle is sized to process.
- When $\dot{m}_w$ needs to go lower than a set lower limit, $\dot{m}_w$ is set to zero and the cycle is switched off. This may happen when the incoming waste heat fluid has a low flow rate or a low incoming temperature, i.e., it carries too little energy.

On the condenser side, an upper limit is needed for the working fluid condensing temperature $T_{w,con}$ to maintain sufficient pressure difference between evaporator and condenser. In some applications, a lower limit is also needed so that the condensing pressure p_{con} remains above the atmospheric pressure to prevent a vacuum. However, unlike the evaporator control which actuates on the working fluid pump, the condenser is controlled via the cold fluid (Manente et al. 2013; Nami et al. 2018). Implementation of these constraints are therefore the responsibility of the encompassing system rather than the component.

3 Model Description

This section describes the model implemented in Modelica.

3.1 Assumptions

The model uses the idealised thermodynamic cycle shown in Figure 2. Depending on the type of the fluid, the cycle has two variants. Figure 2(a) shows a *dry fluid* whose saturated vapour line has a section with positive slope. This implies that no superheating is needed. Figure 2(b) shows a *wet fluid* where superheating is required to avoid liquid formation in the turbine which would cause damage. In this case we assume that the superheating is controlled to be minimised, i.e. the expander outlet state is exactly on the saturation line. How this affects the calculation will be explained in section 3.3.2. There is no subcooling out of the condenser outlet. For any given working fluid, wet or dry, the cycle is fully determined by the working fluid evaporating temperature $T_{w,eva}$, working fluid condensing temperature $T_{w,con}$, the expander efficiency η_{exp}, and the pump efficiency η_{pum}. We will further explain how the working fluid property influences the computation in section 3.3.2.

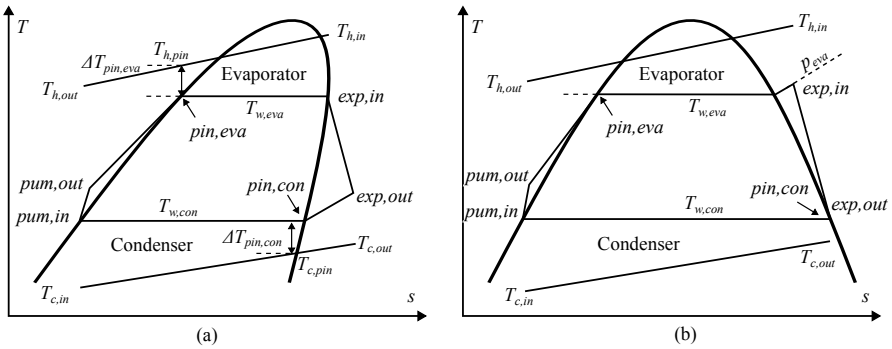


Figure 2. Idealised thermodynamic cycle used to implement the ORC. (a) For a dry fluid, the cycle has no superheating and the expansion starts on the saturation line; (b) For a wet fluid, the cycle is superheated and the expansion ends on the saturation line.

The evaporator pinch point (PP) is at the *bubble point* (where evaporation starts) and the condenser PP is at the *dew point* (where condensation starts). Pan and Shi (2016) discussed where the PP can occur at other places in a phase-change heat exchanger, but our model assumes the PP's are only at these two places.

The evaporation and condensation processes are assumed isobaric. This model therefore excludes any working fluid with a temperature glide, such as zeotropic mixtures. It also means there is no pressure loss along the pipes.

The thermodynamic cycle of the working fluid is steady-state, but the hot and cold fluid streams can be configured to be either steady-state or dynamic. The model does not perform detailed machine analysis. The mass flow rate $\dot{m}_w$ in the model is solved analytically to meet the set point, subject to the described constraints, instead of being controlled using feedback control. For the condenser, at the component level, we use the `assert()` function with `AssertionLevel.error` to stop the simulation when $T_{w,eva} - T_{w,con} < 1$ K and we use `assert()` with `AssertionLevel.warning` when $p_{con} < 101325$ Pa.

3.2 Governing Equations

The evaporator heat exchange is

$$\dot{Q}_{eva} = \dot{m}_h \, c_{p,h} \, (T_{h,out} - T_{h,in}), \tag{1}$$

$$\dot{Q}_{eva} = \dot{m}_w \, (h_{pum,out} - h_{exp,in}), \tag{2}$$

with evaporation taking place at a constant, user-specified temperature $T_{w,eva}$. The evaporator PP difference $\Delta T_{pin,eva}$ is also specified by the user, which is used in

$$\frac{T_{h,pin} - T_{h,out}}{T_{h,in} - T_{h,out}} = \frac{h_{eva,pin} - h_{pum,out}}{h_{exp,in} - h_{pum,out}}, \tag{3}$$

$$\Delta T_{pin,eva} = T_{h,pin} - T_{w,eva}. \tag{4}$$

The condenser side uses the same equations with the variables replaced by their condenser counterparts where appropriate:

$$\dot{Q}_{con} = \dot{m}_c \, c_{p,c} \, (T_{c,out} - T_{c,in}), \tag{5}$$

$$\dot{Q}_{con} = \dot{m}_w \, (h_{exp,out} - h_{pum,in}), \tag{6}$$

$$\frac{T_{c,pin} - T_{c,in}}{T_{c,out} - T_{c,in}} = \frac{h_{con,pin} - h_{pum,in}}{h_{exp,out} - h_{pum,in}}, \tag{7}$$

$$\Delta T_{pin,con} = T_{w,con} - T_{c,pin}. \tag{8}$$

Equations 1 through 8 are eight equations and eight unknowns: $\dot{Q}_{eva}$, $T_{h,out}$, $\dot{m}_w$, $T_{pin,eva}$, $\dot{Q}_{con}$, $T_{c,out}$, $T_{pin,con}$, and $T_{w,con}$. Note that all enthalpy values are known through $T_{w,eva}$, $T_{w,con}$, η_{exp}, and η_{pum}. This will be explained in detailed in section 3.3.

The expander power P_{exp}, pump power P_{pum}, electrical power generated P_{ele}, and cycle thermal efficiency η_{the} are

$$P_{exp} = \dot{m}_w \, (h_{exp,out} - h_{exp,in}), \tag{9}$$

$$P_{pum} = \dot{m}_w \, (h_{pum,out} - h_{pum,in}), \tag{10}$$

$$P_{ele} = P_{exp} + P_{pum} \tag{11}$$

$$\eta_{the} = \frac{-P_{ele}}{\dot{Q}_{eva}}. \tag{12}$$

Note that all energy transfer and power terms follow the sign convention where energy into the cycle is positive. Therefore, $\dot{Q}_{eva}$ and P_{pum} are positive; $\dot{Q}_{con}$ and P_{ele} are negative.

The information flow of the model is summarised in Table 1.

3.3 Thermodynamic Properties

The thermodynamic properties of the working fluid are not computed by a medium model as in `Modelica.Media`, but rather by interpolation schemes. Support points for interpolation are given on the saturated liquid line, saturated vapor line, and a superheated vapor line (called the *reference line*), as shown in Figure 3. Each support curve consists of an array of specific enthalpy, specific entropy, temperature, and pressure. The temperature and pressure arrays are paired as the corresponding saturation values of each other. By default we set the reference line to be 30 K higher than the saturated vapor line. We determined the

User-specified parameters		Inputs		Outputs	
$T_{w,eva}$	Working fluid evaporating temperature,	$T_{h,in}$	Evaporator hot fluid incoming temperature,	$\dot{m}_w$	Working fluid flow rate,
$\Delta T_{pin,eva}$	Evaporator PP temperature difference,	$\dot{m}_h$	Evaporator hot fluid flow rate,	$T_{w,con}$	Working fluid condensing temperature,
$\Delta T_{pin,con}$	Condenser PP temperature difference.	$T_{c,in}$	Condenser cold fluid incoming temperature,	$T_{h,out}$	Evaporator hot fluid outgoing temperature,
η_{exp}	Expander efficiency	$\dot{m}_c$	Condenser cold fluid flow rate	$T_{c,out}$	Condenser cold fluid outgoing temperature,
η_{pum}	Pump efficiency			$\dot{Q}_{eva}$	Evaporator heat flow rate,
				$\dot{Q}_{con}$	Condenser heat flow rate,
				P_{exp}	Expander power output,
				P_{pum}	Pump power consumption.

Table 1. Information flow of the model

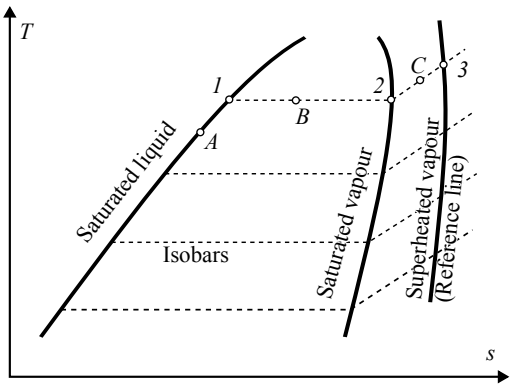


Figure 3. Support curves for interpolation. A, B, and C are example points on a saturation line, between the two saturation lines, and in the superheated region, respective. 1, 2, and 3 are example reference points on the saturation lines and the reference line used to find the example points.

values of these support points using CoolProp (Bell et al. 2014) with its Python wrapper. It should be noted that any software that provides thermodynamic fluid properties can be used to find these points.

3.3.1 Interpolation Schemes

We will demonstrate the interpolation schemes using Figure 3.

- On the saturation line, the specific enthalpy, specific entropy or density, here labeled as y_A, are obtained using cubic Hermite spline interpolation as

$$y_A = s(u_A, d) \tag{13}$$

  where $s(\cdot,\cdot)$ is a cubic Hermite spline, u_A is the input property, and d are the support points. For the saturation curves, the user can configure the model to use either the saturation pressure or the saturation temperature for u_A; for the reference line in Figure 3, u_A is the pressure.

- If the fluid is wet, the isentropic expander outlet point would be in between the saturation lines, shown in Figure 4(b). In this case, its enthalpy h_B is obtained from

$$\frac{h_B - h_1}{s_B - s_1} = \frac{h_2 - h_1}{s_2 - s_1} \tag{14}$$

  where s_B is known because it equals the expander inlet entropy, and all other points are on the saturation line and therefore can be found using (13).

- C is a point in the superheated vapor region. This is the case for the expander outlet and the isentropic expander outlet in Figure 3(a), the expander outlet in Figure 3(b), the expander inlet and the isentropic expander inlet in Figure 3(c). The isobaric lines are not straight in this section, but they are assumed linear so that (13) can be applied using the saturated vapor line and the reference line, albeit with less accuracy.

3.3.2 Expander and Pump

Some expander and pump state points cannot be directly found via interpolation and are estimated using the methods described in this section.

Expander Inlet and Outlet

The calculations of expander inlet and expander outlet depend on the characteristics of the fluid and on the expander efficiency η_{exp}.

Working fluids can be classified as *dry fluids* and *wet fluids* according to the shape of their saturation lines and this has significant implications on ORC system design and efficiency (Hung 2001; Mago, Chamra, and Somayaji

 10.3384/ECP207127 DOI

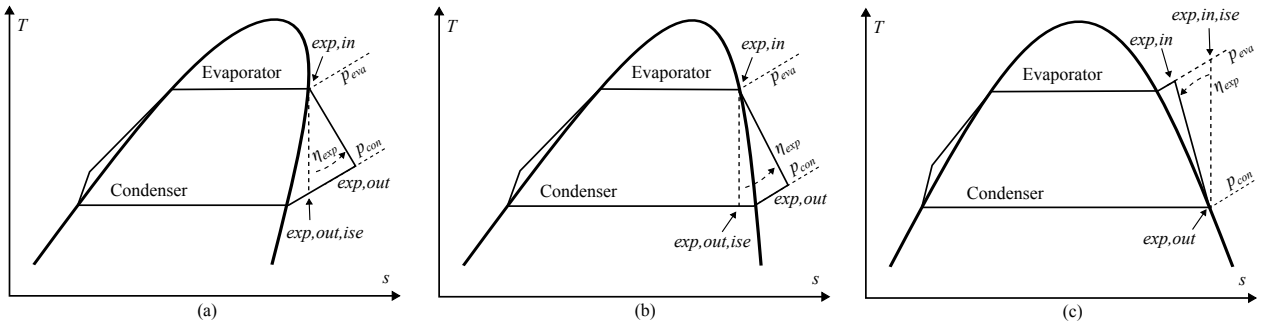


Figure 4. Characteristics of the fluid and the cycle. (a) Dry fluid, dry cycle; (b) wet fluid, dry cycle; (c) wet fluid, wet cycle.

2007; B.-T. Liu, Chien, and C.-C. Wang 2004; Yu, Feng, and Y. Wang 2016). On T-s charts, a dry fluid has a section of positive slope on its saturated vapor line, as shown in 4(a); whereas a wet fluid does not, as shown in 4(b) and (c). Dry fluids are preferable for ORC because as the expansion starts from the saturated vapor line, there is no risk of condensation in the expander. Therefore, superheating before expansion is not needed. For a wet fluid, whether exp, out will be under the dome depends on η_{exp} as well as the location of exp, in.

With the objective to minimise the superheating temperature difference ΔT_{sup} and with η_{exp} known, we distinguish the following two computational paths:

- We call a *dry cycle* a cycle in which the expansion starts from the saturated vapor line (i.e. $\Delta T_{sup} = 0$) and ends in the superheated vapor region. For either a dry fluid or a wet fluid undergoing such a cycle, shown in Figure 4(a) and (b), the expander outlet specific enthalpy $h_{exp,out}$ is obtained from

$$\frac{h_{exp,in} - h_{exp,out}}{h_{exp,in} - h_{exp,out,ise}} = \eta_{exp}. \tag{15}$$

- We call a *wet cycle* a cycle in which the expansion starts from the superheated vapor region and ends on the saturated vapor line. This way ΔT_{sup} assumes the smallest value without causing condensation at expander outlet. In this scenario, the expander inlet specific enthalpy $h_{exp,in}$ is obtained from

$$\frac{h_{exp,in} - h_{exp,out}}{h_{exp,in,ise} - h_{exp,out}} = \eta_{exp}. \tag{16}$$

Pump Outlet

The pump outlet state is obtained from

$$h_{pum,out} = h_{pum,in} + w_{pum}. \tag{17}$$

In our Modelica implementation, the pump power consumption is estimated from fluid work and efficiency as

$$P_{pum} = \frac{\dot{V}\,\Delta p}{\eta_{pum}}. \tag{18}$$

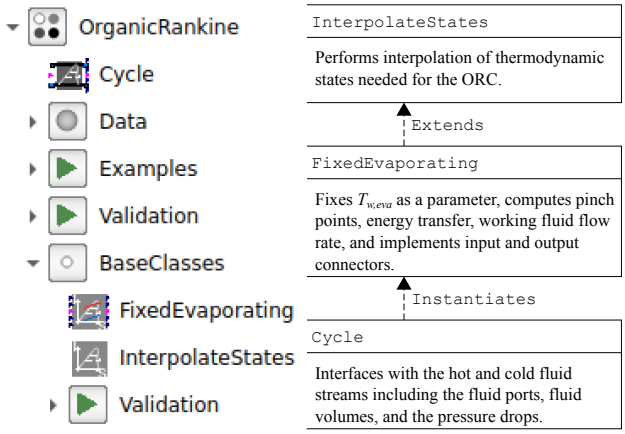


Figure 5. A screenshot from the the package browser and a structure diagram of the implementation

Dividing both sides of (18) by $\dot{m}_w$ yields the specific pump work

$$w_{pum} = \frac{\Delta p}{\rho_w\,\eta_{pum}}. \tag{19}$$

Using the pump inlet state for ρ_w and expanding Δp yields

$$w_{pum} = \frac{p_{eva} - p_{con}}{\rho_{pum,in}\,\eta_{pum}}. \tag{20}$$

This approximation takes advantage of the negligible density change of liquid to avoid property search in the subcooled liquid region and an additional reference line.

In section 5.1, we will validate the above pump work approximation assuming constant density against the pump work estimated from the isentropic process using CoolProp, similar to the expander work in Equation 15, i.e.

$$\frac{h_{pum,out,ise} - h_{pum,in}}{h_{pum,out} - h_{pum,in}} = \eta_{pum}. \tag{21}$$

4 Modelica Package Structure

Inside the Modelica package, the ORC model was implemented in three levels, as shown in Figure 5. At the

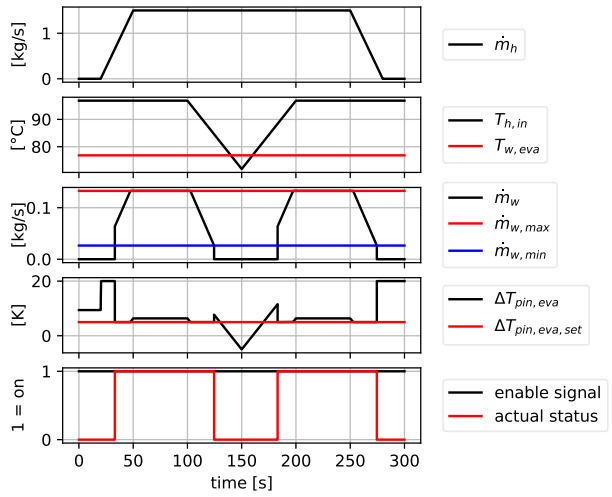


Figure 6. Model validation with variable hot fluid incoming temperature and flow rate

lowest level, thermodynamic property interpolation for the ORC was implemented in `InterpolateStates`. This functionality is in its standalone model for two reasons. First, thermodynamic property estimations from the interpolation schemes are easy to be validated against property tables. Second, this model has no constraints that are imposed by the cycle and its control. When extended by `FixedEvaporating`, various constraints are imposed on the cycle computation to satisfy control objectives described in sections 2 and 3.1. Having the unconstrained model `InterpolateStates` available by itself makes it easy to add different ORC models in the future. At the intermediate level, `FixedEvaporating` adds calculation of $\dot{m}_w$, $T_{pin,eva}$, and $T_{pin,con}$, which are central to the control objectives of the modelled system. At the top level, `Cycle` finally involves the hot and cold fluid streams and is ready to be integrated into an energy system.

5 Model Validation

5.1 Medium Property

The fluid property interpolation implemented in Modelica was validated by comparing its results against direct property readings from CoolProp. Tests were performed with five dry and five wet working fluids to compare results for the specific energy terms

$$w_{exp} = h_{exp,out} - h_{exp,in}, \tag{22}$$

$$w_{pum} = h_{pum,out} - h_{pum,in}, \tag{23}$$

$$q_{eva} = h_{exp,in} - h_{pum,out}, \tag{24}$$

and

$$q_{con} = h_{pum,in} - h_{exp,out}, \tag{25}$$

using the error term

$$err_y = \frac{y_M - y_C}{y_C}, \tag{26}$$

Fluid			ORC Setup			Results		Errors			
	M [g/mol]	T_{cri} [°C]	T_{eva} [°C]	T_{con} [°C]	ΔT_{sup} [K]	η_{the} (Modelica)	η_{the} (CoolProp)	(w_{exp})	(w_{pum})	(q_{eva})	(q_{con})	(η_{the})
Dry												
n-Heptane	100	267	147	37	0	20.1%	19.2%	3.6%	1.3%	-0.5%	-1.5%	4.2%
n-Pentane (R601)	72	197	112	37	0	16.8%	16.6%	0.2%	0.9%	-0.5%	-0.6%	0.7%
Toluene	92	319	173	37	0	23.3%	24.7%	-6.1%	1.4%	-0.6%	1.2%	-5.7%
R123	153	184	105	37	0	16.3%	16.4%	-1.4%	0.9%	-0.5%	-0.3%	-1.1%
R245fa	134	154	91	37	0	13.4%	13.6%	-1.3%	0.8%	-0.3%	-0.2%	-1.1%
Wet												
Acetone	58	235	131	37	5.9	23.1%	23.1%	0.3%	1.2%	0.1%	0.0%	0.2%
Ethanol	46	242	134	37	41.1	26.1%	26.5%	-2.2%	1.6%	-0.6%	0.0%	-1.7%
Propane (R290)	44	97	62	37	3.9	7.4%	7.3%	1.6%	0.5%	0.1%	0.0%	1.7%
R134a	102	101	64	37	4.2	8.2%	8.0%	1.5%	0.4%	0.1%	0.0%	1.5%
R32	52	78	53	37	14.3	4.8%	4.6%	3.7%	0.3%	0.2%	0.0%	4.2%

Table 2. Working fluids properties, experiment setup, results, and errors.

where the subscript M represents Modelica and C represents CoolProp. All tests use $T_{eva} = T_{cri} - 20\,\text{K}$, $T_{con} = 310\,\text{K}$, $\eta_{exp} = 0.8$ and $\eta_{pum} = 0.7$. Their properties, experiment setup, and errors are listed in Table 2.

Table 2 shows the largest errors in w_{exp}. This is thought to be caused by the linear approximation in the superheated region, namely for $h_{exp,in}$ in the case of wet cycles and for $h_{exp,out}$ in the case of dry cycles, and affecting η_{the}. The maximum error is for w_{exp} and η_{the} for Toluene, with

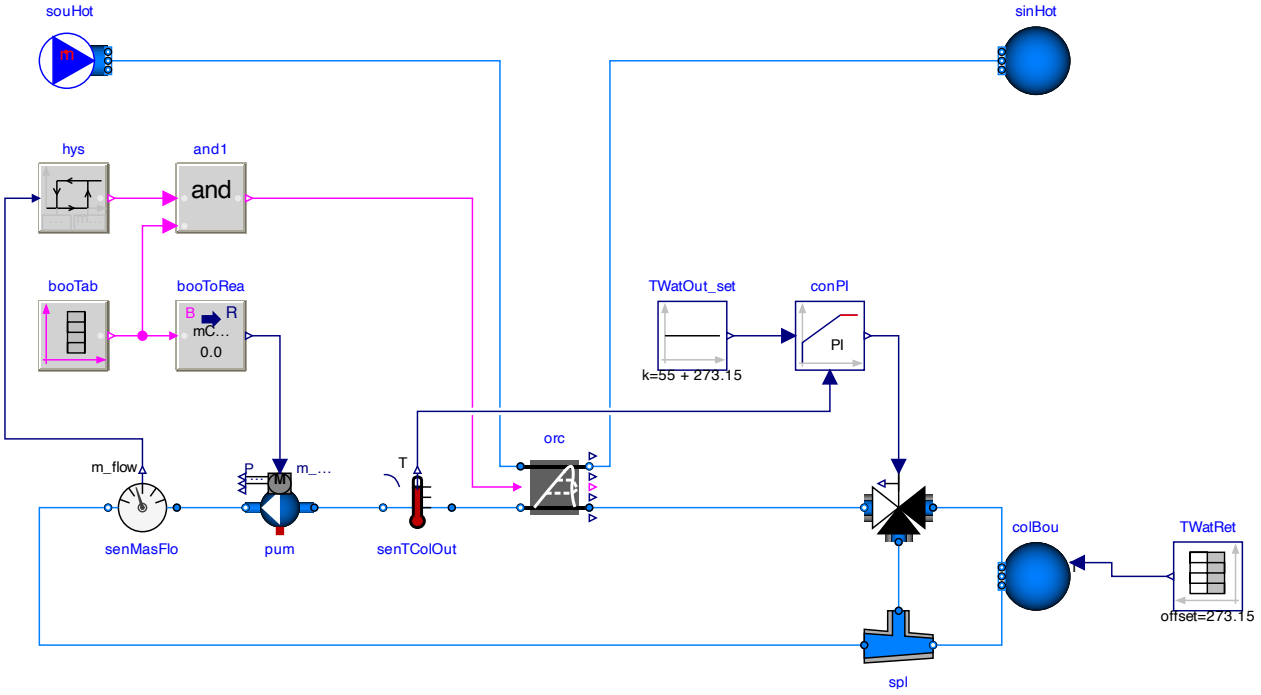


Figure 7. Modelica graphics of the example model where the ORC component is integrated in a district heating system.

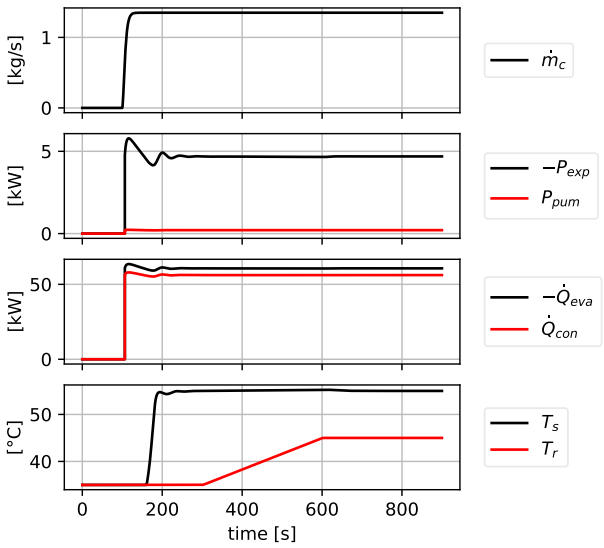


Figure 8. Results output of the example model.

an error of 6.1% and 5.7%, respective. All other errors are below 5%.

Our property computation results agree with the thermodynamic analysis performed by Borsukiewicz-Gozdur (2013), H. Liu, Shao, and Li (2011), and Chen et al. (2006).

5.2 Pinch Point

We conducted a validation to test the constraints on $\dot{m}_w$. We used R245fa as working fluid with $T_{w,eva} = 350$ K, $\Delta T_{pin,eva} = 5$ K, $\Delta T_{pin,con} = 10$ K.

Figure 6 shows how the model deals with a waste heat source whose temperature and flow rate are both variable. It goes through the following stages:

- At $t = 0$, $T_{h,in}$ is sufficiently high but $\dot{m}_h$ is too low. The cycle does does not start ($\dot{m}_w = 0$ and "actual status" is `off`). The set point for $\Delta T_{pin,eva}$ is ignored.
- As $\dot{m}_h$ goes higher, the cycle starts ($\dot{m}_w > 0$ and "actual status" is `on`) when $\dot{m}_h > \dot{m}_{w,min} + \Delta m_{hys}$, where Δm_{hys} is a parameter for the mass flow rate hysteresis. At this stage, $T_{pin,eva}$ is maintained at its set point.
- With $\dot{m}_h$ increasing further, $\dot{m}_w$ reaches its upper limit and no longer increases along with $\dot{m}_h$. $\Delta T_{pin,eva}$ is allowed to go higher than its set point.
- Then at $t = 100$, $T_{h,in}$ starts to decrease. $\dot{m}_w$ is again able to maintain $\Delta T_{pin,eva}$ at its set point shortly after, before the cycle shuts down again when $T_{h,in}$ becomes too low.
- From $t = 150$ the stages above run again in reverse order.

6 Example Model

We integrated our ORC model in a hypothetical district energy system as an example. The model graphics is shown in Figure 7. The hot water return from the system is connected through the ORC condenser, acting as the ORC cold fluid. The mixing valve is modulated with

a PI loop controlling the cold fluid outgoing temperature (i.e. district hot water supply temperature).

The working fluid is R123. It is a dry fluid, i.e. no superheating in the cycle.

Carrying waste heat, the evaporator hot fluid is air, with a constant flow rate and a constant incoming temperature.

The condenser cold fluid represents water from a district heating system. A dedicated condenser pump is used to maintain a constant water flow rate through the condenser. The district hot water return temperature T_r (i.e. the condenser cold fluid incoming temperature $T_{c,in}$ of the ORC) fluctuates between 35 to 45°C. The ORC is controlled to lift its temperature to a supply temperature T_s of 55°C. Additionally, a safety control is implemented to prevent the cycle from starting until the water flow in the condenser is established, i.e. $\dot{m}_c > \dot{m}_{c,threshold}$.

Nominal conditions of this model are shown in Table 3. Simulation results of key variables of this example model are shown in Figure 8.

Quantity	Value	Unit	Quantity	Value	Unit
T_{eva}	100	°C	$\dot{m}_h$	1	kg/s
T_{con}	59.5	°C	$T_{h,in}$	150	°C
p_{eva}	786	kPa	$T_{h,out}$	90.2	°C
p_{con}	282	kPa	$\dot{m}_c$	1.4	kg/s
η_{exp}	0.8	-	$T_{c,in}$	45	°C
η_{pum}	0.6	-	$T_{c,out}$	55	°C
$\dot{m}_w$	0.34	kg/s	η_{the}	7.7%	-

Table 3. Example model nominal conditions.

7 Discussion

We envision that that the model can be further developed in the future to address the following.

Table 2 shows that the interpolation schemes are highly accurate in energy transfer calculation with errors up to 1.6% for w_{pum}, q_{eva}, and q_{con}. For w_{exp} and η_{the}, the highest error was 6.1% and 5.7% with toluene. These higher errors appear to be caused by the linear approximation of isobars in the superheated vapor region and related to the specific characteristics of the fluids. Deeper understanding in this will be valuable in deciding how the medium simplification can be modified to achieve higher accuracy. Note that the validation was intentionally designed to have a very high evaporating temperature ($T_{eva} = T_{cri} - 20$ K) to test extreme cases. Real-world subcritical applications may not use such a high T_{eva} and the model estimation would be more accurate.

Although machine analysis is beyond the scope of this study and we leave the expander efficiency to the user to specify, it is nonetheless important to note that the computation of expander efficiency is important and can improve the estimation accuracy of the model.

In our current model, we used a simple ORC architecture without considering subcooling, recuperating, multiple-stage expansion, or throttling. Supporting more sophisticated architectures such as reviewed by (Lecompte et al. 2015) can expand the general utility of this model.

8 Conclusion

In this work we reported a Modelica implementation of an ORC model. Our model fills the gap of a general-use, open-source ORC model in Modelica with both the following features: a working fluid model that is ready to use or easy to add, and the ability to be integrated into a larger energy system. Because the working fluid selection is an important design decision in ORC system design, our method opens up the ability to choose working fluids from a pool of candidates for an early-stage analysis of ORC system integration. This relieves modellers from the challenges of developing a large number of detailed and computationally efficient medium models. This also results in a standalone model that does not depend on external software for fluid property queries, improving usability and compatibility.

Acknowledgements

This research was supported by the Assistant Secretary for Efficiency and Renewable Energy, Office of Building Technologies and Industrial Efficiency and Decarbonization Office of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231.

Data Availability

This is an open-source development. At the time of writing, the data and code used in this study are available at Modelica Buildings Library (Wetter et al. 2014) through commit `4d9b7fd` and will be released in future versions of the Modelica Buildings Library.

Nomenclature

Quantities:

c_p	specific heat capacity at constant pressure
h	specific enthalpy
M	molar mass
$\dot{m}$	mass flow rate
P	power
p	pressure
$\dot{Q}$	heat flow rate
q	specific heat flow
s	specific entropy
T	temperature
$\dot{V}$	volumetric flow rate
w	specific work
x	vapor quality
η	efficiency
ρ	density

Subscripts:

c	cold fluid

ele	electrical
h	hot fluid
hys	hysteresis
in	incoming, inlet
ise	isentropic
out	outgoing, outlet
pin	pinch point
w	working fluid
con	condenser
cri	critical
eva	evaporator
exp	expander
pum	pump
sup	superheating
the	thermal

References

Abadi, Gholamreza Bamorovat and Kyung Chun Kim (2017). "Investigation of organic Rankine cycles with zeotropic mixtures as a working fluid: Advantages and issues". In: *Renewable and Sustainable Energy Reviews* 73, pp. 1000–1013.

Asimptote (2023). *FluidProp*. https://asimptote.com/fluidprop/. Date accessed: 22-Oct-2023.

Bao, Junjiang and Li Zhao (2013). "A review of working fluid and expander selections for organic Rankine cycle". In: *Renewable and sustainable energy reviews* 24, pp. 325–342.

Bell, Ian H. et al. (2014). "Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp". In: *Industrial & Engineering Chemistry Research* 53.6, pp. 2498–2508. DOI: 10.1021/ie4033999. eprint: http://pubs.acs.org/doi/pdf/10.1021/ie4033999. URL: http://pubs.acs.org/doi/abs/10.1021/ie4033999.

Borsukiewicz-Gozdur, Aleksandra (2013). "Pumping work in the organic Rankine cycle". In: *Applied Thermal Engineering* 51.1-2, pp. 781–786.

Casella, Francesco and Alberto Leva (2005). "Object-oriented modelling & simulation of power plants with modelica". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, pp. 7597–7602.

Chen, Yang et al. (2006). "A comparative study of the carbon dioxide transcritical power cycle compared with an organic Rankine cycle with R123 as working fluid in waste heat recovery". In: *Applied thermal engineering* 26.17-18, pp. 2142–2147.

Hung, Tzu-Chen (2001). "Waste heat recovery of organic Rankine cycle using dry fluids". In: *Energy Conversion and management* 42.5, pp. 539–553.

Imran, Muhammad et al. (2020). "Dynamic modeling and control strategies of organic Rankine cycle systems: Methods and challenges". In: *Applied Energy* 276, p. 115537.

Lecompte, Steven et al. (2015). "Review of organic Rankine cycle (ORC) architectures for waste heat recovery". In: *Renewable and sustainable energy reviews* 47, pp. 448–461.

Liu, Hao, Yingjuan Shao, and Jinxing Li (2011). "A biomass-fired micro-scale CHP system with organic Rankine cycle (ORC)–Thermodynamic modelling studies". In: *Biomass and Bioenergy* 35.9, pp. 3985–3994.

Liu, Bo-Tau, Kuo-Hsiang Chien, and Chi-Chuan Wang (2004). "Effect of working fluids on organic Rankine cycle for waste heat recovery". In: *Energy* 29.8, pp. 1207–1217.

Mago, Pedro J, Louay M Chamra, and Chandra Somayaji (2007). "Performance analysis of different working fluids for use in organic Rankine cycles". In: *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy* 221.3, pp. 255–263.

Manente, Giovanni et al. (2013). "An Organic Rankine Cycle off-design model for the search of the optimal control strategy". In: *Energy* 58, pp. 97–106.

Modelica 3rd-party libraries (2023). *ExternalMedia*. https://github.com/modelica-3rdparty/ExternalMedia. Date accessed: 23-Oct-2023.

Nami, Hossein et al. (2018). "Gas turbine exhaust gas heat recovery by organic Rankine cycles (ORC) for offshore combined heat and power applications-Energy and exergy analysis". In: *Energy* 165, pp. 1060–1071.

Oliveira, Miguel Castro, Muriel Iten, and Henrique A Matos (2022). "Simulation and assessment of an integrated thermal processes and Organic Rankine Cycle (ORC) system with Modelica". In: *Energy Reports* 8, pp. 764–770.

Pan, Lisheng and Weixiu Shi (2016). "Investigation on the pinch point position in heat exchangers". In: *Journal of Thermal Science* 25, pp. 258–265.

Quoilin, Sylvain, Richard Aumann, et al. (2011). "Dynamic modeling and optimal control strategy of waste heat recovery Organic Rankine Cycles". In: *Applied energy* 88.6, pp. 2183–2190.

Quoilin, Sylvain, Adriano Desideri, et al. (2014). "ThermoCycle: A Modelica library for the simulation of thermodynamic systems". In: *10th international Modelica conference*.

Quoilin, Sylvain, Martijn Van Den Broek, et al. (2013). "Techno-economic survey of Organic Rankine Cycle (ORC) systems". In: *Renewable and sustainable energy reviews* 22, pp. 168–186.

Saleh, Bahaa et al. (2007). "Working fluids for low-temperature organic Rankine cycles". In: *Energy* 32.7, pp. 1210–1221.

Twomey, Braden Lee (2016). "Dynamic simulation and experimental validation of an Organic Rankine Cycle model". In.

U.S. DOE (2021-04). *Waste Heat to Power*. https://betterbuildingssolutioncenter.energy.gov/sites/default/files/attachments/Waste_Heat_to_Power_Fact_Sheet.pdf. Date accessed: 15-Aug-2023.

U.S. DOE (2024). *Decarbonizing the U.S. Economy by 2050: A National Blueprint for the Buildings Sector*. URL: https://www.energy.gov/eere/articles/decarbonizing-us-economy-2050.

Wetter, Michael et al. (2014). "Modelica Buildings Library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Yu, Haoshui, Xiao Feng, and Yufei Wang (2016). "Working fluid selection for organic Rankine cycle (ORC) considering the characteristics of waste heat sources". In: *Industrial & Engineering Chemistry Research* 55.5, pp. 1309–1321.

Zimmer, Dirk (2020). "Robust object-oriented formulation of directed thermofluid stream networks". In: *Mathematical and Computer Modelling of Dynamical Systems* 26.3, pp. 204–233. DOI: 10.1080/13873954.2020.1757726. URL: https://doi.org/10.1080/13873954.2020.1757726.

Zimmer, Dirk, Michael Meißner, and Niels Weber (2022). "The DLR ThermoFluid Stream Library". In: *Electronics* 11.22. ISSN: 2079-9292. DOI: 10.3390/electronics11223790. URL: https://www.mdpi.com/2079-9292/11/22/3790.

Advancements in Building-to-Grid Interactions: Thermo-Electric Coupling Models of Motor-driven Devices

Viswanathan Ganesh[1] Zhanwei He[1] Wangda Zuo[1, 2]

[1]Department of Architectural Engineering, Pennsylvania State University, University Park, PA, USA
{viswanathan.ganesh,zuh45,wangda.zuo}@psu.edu
[2]National Renewable Energy Laboratory, Golden, Colorado, CO

Abstract

Building-to-grid (B2G) integration transforms buildings into active components of the electricity grid, enhancing dynamic energy management and optimizing usage to reduce operational costs and carbon emissions. However, existing modeling tools for building and power systems often overlook or oversimplify the interactions between power system dynamics and building dynamics. This paper introduces Modelica-based thermo-electric coupling models for motor-driven devices in buildings, such as pumps and heat pumps. The developed models assess transient oscillations and negative active power in these devices within B2G systems. We compare the proposed models with a base model from the Modelica Building Library that uses a radiator and heat pump to maintain room temperature. The simulation results demonstrate that the motor-driven models effectively capture transient oscillations in current and power when the systems are activated and deactivated. Additionally, the occurrence of negative power when systems turn off is a critical factor in enhancing B2G system stability and energy efficiency. These findings underscore the model's ability to improve grid support, advancing energy management practices in B2G applications.

Keywords: Thermo-Electric Coupling, Building-to-Grid (B2G), Heat Pumps, Pumps

1 Introduction

In modern engineering applications, the quest for energy efficiency and system reliability is of paramount importance. Systems that integrate pumps, heat pumps, and chillers are critical components in various industrial processes, HVAC (Heating, Ventilation, and Air Conditioning) systems, and renewable energy applications. These systems rely heavily on the interplay between thermal and electrical domains, where thermo-electric coupling plays a vital role in their overall performance and energy management.

Thermo-electric coupling involves the interaction between thermal and electrical energy, where changes in thermal conditions can significantly impact the electrical performance of a system and vice versa (Fachini, De Castro, et al. 2022). Understanding these interactions is crucial for optimizing the design, operation, and control of integrated systems. However, the transient effects caused by thermal changes in such coupled systems are not fully understood, posing challenges for engineers and researchers aiming to enhance system reliability and efficiency.

The primary objective of this study is to investigate the transient effects on the electrical side due to thermal variations in systems involving pumps, heat pumps, and chillers. This research aims to develop a theoretical model that simulates these effects and to validate this model through a comparative analysis with findings from existing available established simulation models. By doing so, the study seeks to bridge the gap between theoretical predictions and practical observations, providing a more comprehensive understanding of thermo-electric coupling in transient conditions.

The rest of the paper is organized as follows: Section 2 provides a literature review and related work, Section 3 describes motor-driven devices such as fans, pumps, and heat pumps. Section 4 introduces the governing equations for modeling induction motors, and Section 5 presents the motor-driven models. The case study and the simulation results are discussed in Sections 6 and 7, respectively. Finally, Section 8 concludes the paper.

2 Literature Review

Thermo-electric coupling, in the context of this study, refers to the process of integrating mechanical devices such as pumps, heat pumps, and chillers, which are responsible for thermal performance in buildings with electrical devices like motors. This coupling involves the interaction between the electrical and mechanical parts, where changes in thermal conditions within the mechanical components can significantly impact the electrical performance of the system, and vice versa (Fu, Huang, Vrabie, et al. 2019; Fu, Huang, Liu, et al. 2019). Understanding this interaction is crucial for optimizing the performance and energy efficiency of integrated systems used in HVAC and other industrial applications (Li et al. 2022).

Pumps are devices used to move fluids (liquids or gases) by mechanical action. They play a crucial role in various applications, including water supply, air conditioning, refrigeration, and industrial processes. Pumps can be classified into different types, such as centrifugal pumps, which

use a rotating impeller to add velocity to the fluid, and positive displacement pumps, which move fluid by trapping a fixed amount and forcing (displacing) it into the discharge pipe (Karassik 2001). Heat pumps transfer thermal energy from a cooler space to a warmer space using mechanical energy, effectively functioning as both a heating and cooling device. They are commonly used in HVAC systems to provide space heating and cooling. The efficiency of heat pumps is significantly influenced by the thermodynamic properties of the working fluid and the design of the system components.

Chillers are used to remove heat from a liquid via a vapor-compression or absorption refrigeration cycle. This cooled liquid can then be circulated through a heat exchanger to cool air or equipment. Chillers are essential in industrial cooling processes and large-scale air conditioning systems. The performance of chillers depends on factors such as refrigerant type, system design, and operational conditions. The study of transients in electrical systems, particularly those induced by thermal changes, is critical for ensuring the stability and reliability of integrated systems (Stoecker and Stoecker 1998). Transient phenomena occur due to sudden changes in system conditions, such as load variations, switching operations, or environmental factors. These transients can lead to voltage fluctuations, current surges, and potential system instability (Kundur, Balu, and Lauby 1994).

One significant challenge in current research is the use of simplified models that do not fully capture the complexity of thermo-electric interactions. These models often assume steady state conditions and neglect dynamic behaviors, leading to discrepancies between theoretical predictions and real world observations. There is a lack of empirical validation for many theoretical and simulation based studies. Without experimental data to corroborate simulation results, it is difficult to assess the accuracy and reliability of the models used. The application of advanced simulation techniques, such as multi-physics modeling and co-simulation, is limited. These techniques are essential for accurately representing the interactions between thermal and electrical domains, yet their use remains under explored in existing literature.

3 Motor-driven Devices in Buildings

Motor-driven devices in buildings, such as fans, pumps, heat pumps, and chillers, are significant electricity consumers. The physical system diagram of a motor-driven device, such as a pump, is illustrated in Figure 1. In this setup, the induction motor acts as the primary power source, providing rotational force to the motor shaft. The motor is directly coupled with the pump, allowing the motor to drive the pump's shaft. This system efficiently converts electrical energy into mechanical energy, which is then transformed into fluid movement. For modeling purposes, motor-driven devices can be considered as coupling models, consisting of two main components: the induction motor and the mechanical device.

1. Induction Motor: Induction motors are commonly used in building applications. These motors include key sub-components such as coils, magnets, stators, and rotors. The coils and stators, connected to the VFD's electrical circuit, generate an induced magnetic field. This magnetic field interacts with the rotor to produce electromagnetic torque, causing the rotor to spin at a constant or variable speed (Fachini, Castro, et al. 2023; Fachini, Castro, et al. 2024).

2. Mechanical Devices: These devices convert the transferred torque into mechanical work. For instance, pumps work by converting the input mechanical energy in the fluid being pumped. In the heat pump or chiller, compressor, which is driven by motor, is transferring heat in desired directions—either for heating or cooling. as it enables the refrigerant to absorb or release heat as needed, allowing the heat pump or chiller to function efficiently.

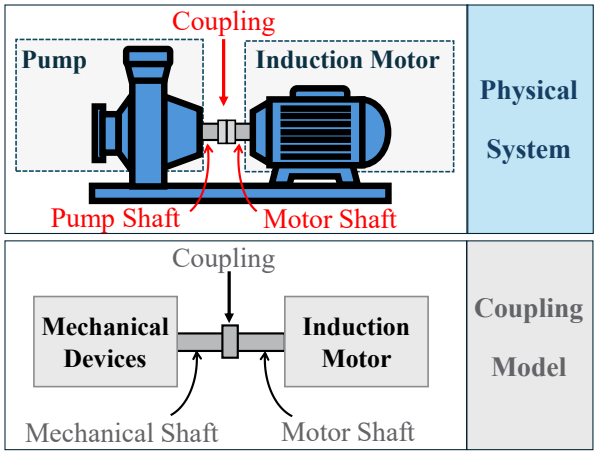


Figure 1. Motor-driven Devices and Coupling Models

Since the mechanical devices, such as pumps, heat pumps, and chillers are available in the Modelica Building Library (Wetter et al. 2014), the motor model for coupling the mechanical devices should be included. The next section illustrates the governing equations for modeling the induction motor.

4 Induction Motor Modeling

The physical components of an induction motor are illustrated in Figure 2. The two main components are the stator and the rotor. The stator, the stationary outer part, receives power and generates a rotating magnetic field. The stator, the stationary outer part, gets the supplied power and generates a rotating magnetic field. The rotor is the rotating part, which is located inside the stator. The magnetic field from the stator induces a current in the rotor, which in turn creates a secondary magnetic field. This secondary field

interacts with the magnetic field of the stator, resulting in the production of electromagnetic torque.

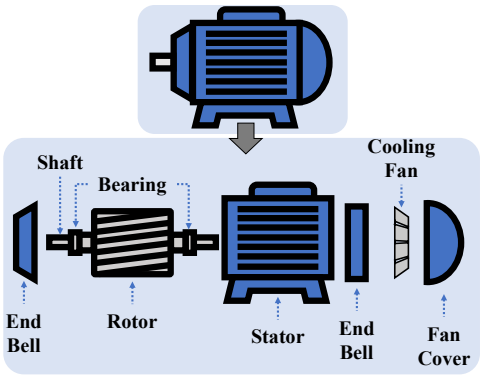


Figure 2. Physical Components of an Induction Motor

For modeling the dynamic behaviors of the induction motor, the DQ-axis method (Gol 1993) is employed to formulate the governing equations. This approach simplifies the analysis by transforming the three-phase system into a two-axis (direct and quadrature) coordinate system, decoupling the complex interactions in the induction motor.

The D and Q axis equivalent circuits of an induction motor are shown in Figures 3 and 4, respectively. The D-axis and Q-axis circuits employ similar components, arranged differently to represent their respective axes within the synchronous reference frame. These circuits share identical resistances and inductances, presenting consistent properties.

The D-axis circuit includes the D-axis stator voltage (u_{ds}) and current (i_{ds}), stator resistance (R_s) and leakage inductance (L_{ls}), induced voltage $(\omega\psi_{ds})$, D-axis rotor voltage (u_{dr}), current (i_{dr}), and resistance (R_r), rotor leakage inductance (L_{lr}) and induced voltage $((\omega_e-\omega_r)\psi_{dr})$, and mutual inductance between stator and rotor (L_m).

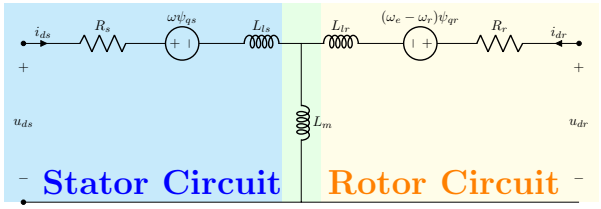


Figure 3. D-axis of the Induction Motor

The Q-axis circuit similarly includes the Q-axis stator voltage (u_{qs}) and current (i_{qs}), with induced voltage $(\omega\psi_{qs})$, stator resistance (R_s) and leakage inductance (L_{ls}), Q-axis rotor voltage (u_{qr}), current (i_{qr}), resistance (R_r), and leakage inductance (L_{lr}).

The primary governing equations include voltage, flux linkage, rotor speed, electromagnetic torque, and current. The following subsections provide detailed explanations of these equations.

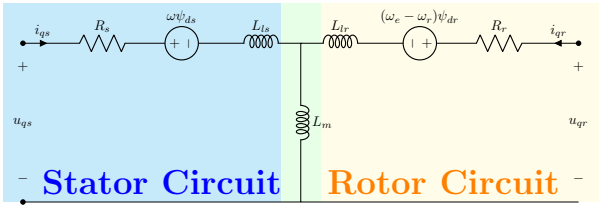


Figure 4. Q-axis of the Induction Motor

4.1 Voltage Equations in DQ-axis

The equivalent voltage equations for the stator are expressed as follows:

$$u_{ds} = R_s i_{ds} + \frac{d\Psi_{ds}}{dt} + \omega\Psi_{qs}, \tag{1}$$

$$u_{qs} = R_s i_{qs} + \frac{d\Psi_{qs}}{dt} + \omega\Psi_{ds}, \tag{2}$$

where ω is the base electrical frequency [rad/s], R_s is the stator resistance [Ω], u_{ds} and u_{qs} are the D and Q axis stator voltages [V], i_{ds} and i_{qs} are the D and Q axis stator currents [A], Ψ_{ds} and Ψ_{qs} are the D and Q axis stator flux linkages [Wb]. Similarly, the rotor equivalent voltage equations can be written as:

$$u_{dr} = R_r i_{dr} + \frac{d\Psi_{dr}}{dt} - (\omega_e-\omega_r)\Psi_{qr}, \tag{3}$$

$$u_{qr} = R_r i_{qr} + \frac{d\Psi_{qr}}{dt} + (\omega_e-\omega_r)\Psi_{dr}, \tag{4}$$

where R_r is the rotor resistance [Ω], ω_e and ω_r are the electrical frequency and rotor angular frequency [rad/s], u_{dr}, u_{qr} are the D and Q axis rotor voltages [V], i_{dr}, i_{qr} are the D and Q axis rotor currents [A], Ψ_{dr}, and Ψ_{qr} are the D and Q axis rotor flux linkages [Wb].

4.2 Flux Equations in DQ-axis

The next step is to calculate the magnetic flux linkages of the stator and rotor, using the underlying equations:

$$\Psi_{ds} = i_{ds}L_s + i_{dr}L_m, \tag{5}$$

$$\Psi_{qs} = i_{qs}L_s + i_{qr}L_m, \tag{6}$$

$$\Psi_{dr} = i_{dr}L_r + i_{ds}L_m, \tag{7}$$

$$\Psi_{qr} = i_{qr}L_r + i_{qs}L_m, \tag{8}$$

where L_s, L_r, and L_m are the stator, rotor, and mutual inductance [H]. The L_s and L_r can be written as:

$$L_s = L_{ls} + L_m, \tag{9}$$

$$L_r = L_{lr} + L_m, \tag{10}$$

where L_{ls} and L_{lr} are the stator and rotor leakage inductance of the machine [H].

10.3384/ECP207136 DOI

4.3 Rotor Speed Equation and Electromagnetic Torque

Since induction motor is an electro-mechanical device, we can formulate the rotor speed based on the torque as

$$\omega_r = \frac{P}{2J}\int (T_e - T_l)\,dt, \tag{11}$$

where P is the number of poles of induction motor, J is the moment of inertia [kg/m^2], T_e and T_l are the electromagnetic and load torque [Nm]. The details for calculating the electromagnetic torques is shown as follows:

$$T_e = \frac{3}{2}\frac{P}{2}L_m(i_{qs}i_{dr} - i_{ds}i_{qr})\ . \tag{12}$$

4.4 Current Equations for Stator and Rotor in DQ-axis

By substituting Equation (5) and Equation (6) into Equation (1) and Equation (2), the stator currents in DQ frames, namely i_{ds} and i_{qs}, can be expressed:

$$\frac{d}{dt}i_{ds} = \frac{1}{L_s}[u_{ds} - i_{ds}R_s - L_m\frac{d}{dt}i_{dr} + \omega_e L_s i_{qs} + \omega_e L_m i_{qr}] \quad \text{and} \tag{13}$$

$$\frac{d}{dt}i_{qs} = \frac{1}{L_s}[u_{qs} - i_{qs}R_s - L_m\frac{d}{dt}i_{qr} - \omega_e L_s i_{ds} - \omega_e L_m i_{dr}]\ . \tag{14}$$

By integrating the Equation (13) and Equation (14), the i_{ds} and i_{qs} are expressed as:

$$i_{ds} = \int \frac{1}{L_s}[u_{ds} - i_{ds}R_s - L_m\frac{d}{dt}i_{dr} + \omega_e L_s i_{qs} + \omega_e L_m i_{qr}]\,dt \quad \text{and} \tag{15}$$

$$i_{qs} = \int \frac{1}{L_s}[u_{qs} - i_{qs}R_s - L_m\frac{d}{dt}i_{qr} - \omega_e L_s i_{ds} - \omega_e L_m i_{dr}]\,dt\ . \tag{16}$$

Similarly, when the flux expressions are replaced in the voltage equations, the rotor currents i_{dr} and i_{qr} can be written as:

$$\frac{d}{dt}i_{dr} = \frac{1}{L_r}[u_{dr} - i_{dr}R_r - L_m\frac{d}{dt}i_{ds} + \omega_e L_r i_{qr} + \omega_e L_m i_{qs}] \quad \text{and} \tag{17}$$

$$\frac{d}{dt}i_{qr} = \frac{1}{L_r}[u_{qr} - i_{qr}R_r - L_m\frac{d}{dt}i_{qs} - \omega_e L_r i_{dr} - \omega_e L_m i_{ds}]\ . \tag{18}$$

After integration, they are described by Equations (19) and (20):

$$i_{dr} = \int \frac{1}{L_r}[u_{dr} - i_{dr}R_r - L_m\frac{d}{dt}i_{ds} + \omega_e L_r i_{qr} + \omega_e L_m i_{qs}]\,dt \quad \text{and} \tag{19}$$

$$i_{qr} = \int \frac{1}{L_r}[u_{qr} - i_{qr}R_r - L_m\frac{d}{dt}i_{qs} - \omega_e L_r i_{dr} - \omega_e L_m i_{ds}]\,dt\ . \tag{20}$$

5 Motor-driven Models

This paper studies two motor-driven models: the motor-driven heat pump and the motor-driven pump. The induction motor model is coupled with a heat pump or pump model available in the Modelica Building Library. Specifically, the path for the heat pump model in the library is `Buildings.Fluid.HeatPumps.Carnot_y`, and the path for the pump model is `Buildings.Fluid.Movers.SpeedControlled_y`. Figure 5 presents that the motor-driven heat pump is equipped with an electrical interface to which the induction motor model connects at the electrical terminal. Additionally, the motor is mechanically coupled with the heat pump, referred to as the mechanical interface. Similarly, Figure 6 shows

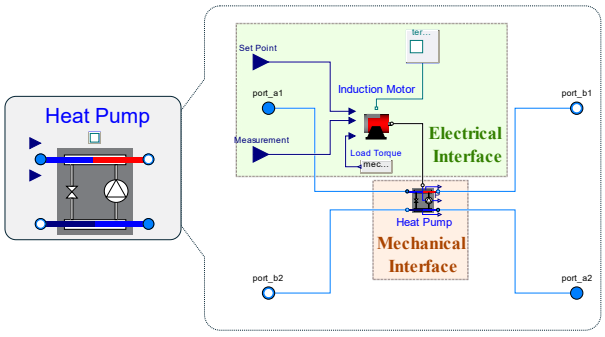


Figure 5. Motor-driven Heat Pump

the coupling of the induction motor model with the pump model in the motor-driven pump. The motor, representing the electrical interface, connects to the electrical terminal. This connection provides detailed insights into the real-time power consumption, current, and other electrical domain information. Also, the mechanical coupling of the motor to the pump allows for a more realistic representation of the pump operation. Based on the devel-

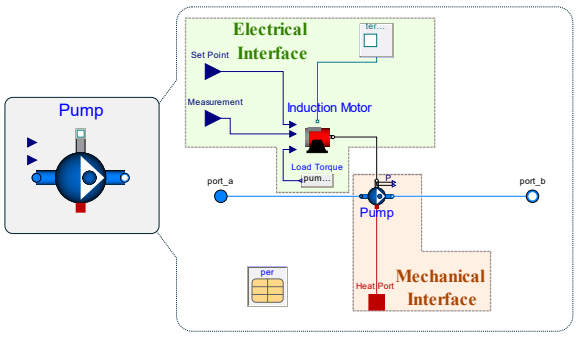


Figure 6. Motor-driven Pump

oped motor-driven heat pump and pump, a case study has been conducted to demonstrate how motor-driven models can reveal more detailed insight into the electrical domain information and more realistic interactions between the power system.

6 Case Study

In this case study, the heat pump and pump supply water to a radiator for heating, aiming to maintain a room temperature of 20 °C. A model is found in the Modelica Building Library, and the path is `Buildings.Fluid.HeatPumps.Examples.ScrollWaterToWater_OneRoomRadiator`.

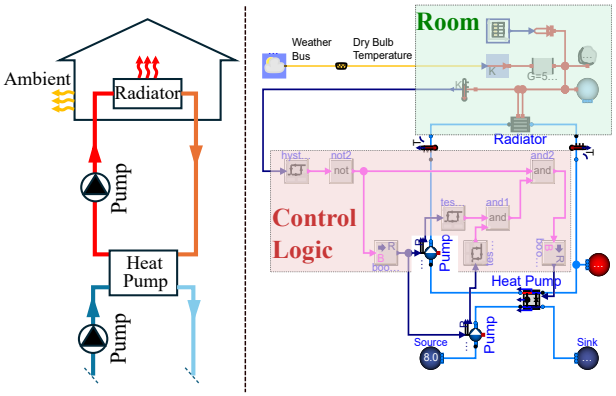


Figure 7. System Schematics and the Modelica Model

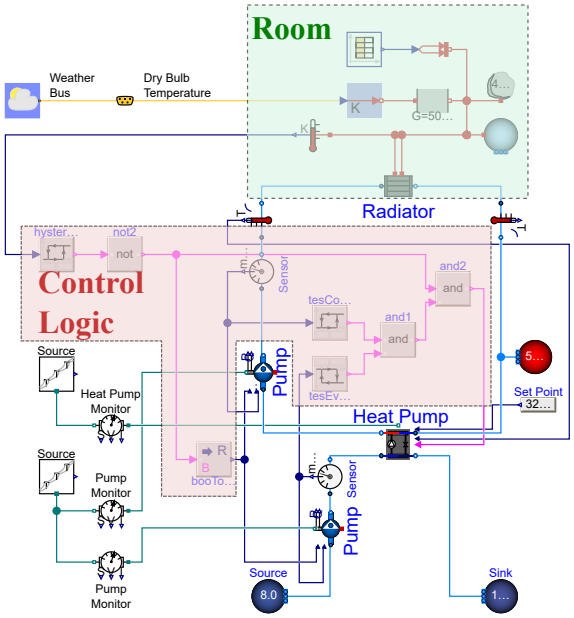


Figure 8. Motor-driven Model Implementation

As shown in Figure 7, This system model simulates a single room equipped with a radiator, which is heated by a 24 kW nominal capacity heat pump. The heat pump operates as follows: the source side water, entering the evaporator at a constant temperature of 10°C, is heated to a nominal condenser output temperature of 50 °C for the radiator. The return temperature from the radiator is set at 45°C. The heat pump is set to activate when the room temperature drops below 19°C and deactivate when the temperature exceeds 21°C. The on/off control for both the heat pump and pumps is achieved by the control logic, highlighted by the pink shadow on the right-hand side of Figure 7.

This system model can serve as a baseline because it calculates the power consumption of the heat pump and pump based on the heat flow through the condenser and evaporator and the empirical efficiency of the heat pump. The motor-driven heat pump and pump models, depicted in Figures 5 and 6, replace the base models in the system model. Figure 8 details the implementation of these motor-driven models, highlighting the addition of electrical terminals.

The following section presents simulation results that compare the performance of the base system model with the version that includes motor-driven models. This comparison specifically focuses on highlighting differences in electrical performance.

7 Results and Discussions

7.1 Fluid System

The primary objective of the system model is to maintain the room temperature and Figure 9 displays simulation results showing that the room temperature fluctuates between approximately 291K (17.85 °C) and 295K (21.85 °C). Both models have effectively demonstrated their ability to maintain the room temperature.

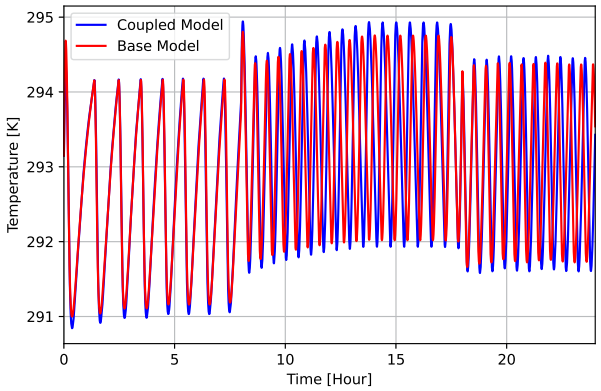


Figure 9. Room Temperature

The room temperature is maintained by the radiator. The coupled model provides an accurate simulation of room temperature changes, closely aligning with actual observed temperatures. The results for the supply temperature and the return temperature for the radiator are shown in Figure 10 and Figure 11. The radiator supply temperature ranges from about 300K (26.85°C)to 325K (51.85 °C), with the coupled model's simulations more consistent with observed data, highlighting its capability to track supply temperature variations over time.

The return temperature (Figure 11) shows variations between approximately 300K (26.85 °C) and 320K (46.85

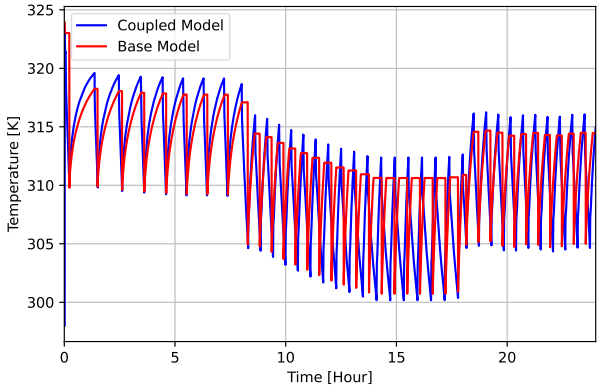


Figure 10. Radiator Supply Temperature

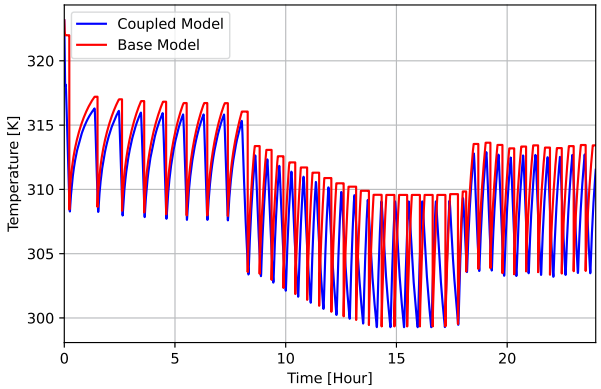


Figure 11. Radiator Return Temperature

°C), with the coupled model demonstrating closer alignment with base model, capturing the dynamics of return temperature fluctuations throughout the day. Overall, the comparative analysis across these three temperature metrics demonstrates that the coupled model consistently provides accurate and reliable simulations, proving its ability to simulate temperature dynamics and heat flow effectively.

7.2 Electrical System

7.2.1 Heat Pump Electrical Monitoring

The heat pump power (Figure 12) shows variations between -5 kW and 20 kW in active power, while reactive power (Figure 13) varies from -1 kVAr to 17 kVar. On the other hand, the heat pump power factor (Figure 14) shows variations between 0 - 1 and the nominal power factor during operation is 0.93, while the heat pump current (Figure 15) ranges from -10A to 70A. In the coupled model, significant transient oscillations in power and current are observed, particularly during the initial and transition phases, which are not as pronounced in the base model.

These oscillations are crucial to consider as they impact the stability and performance of B2G systems. Additionally, instances of negative active power are observed in both power and current data, indicating periods where the heat pump contributes power back to the grid, an important factor for energy efficiency and grid support.

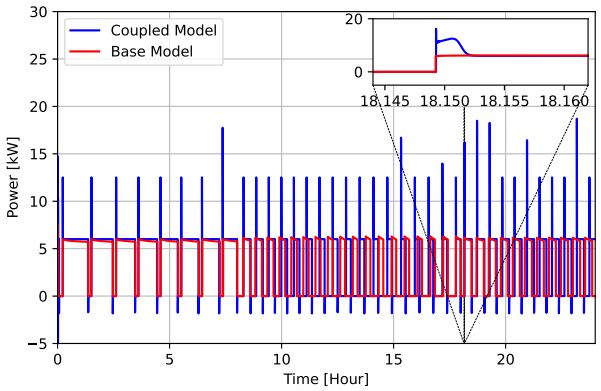


Figure 12. Motor-drive Heat Pump Active Power Consumption

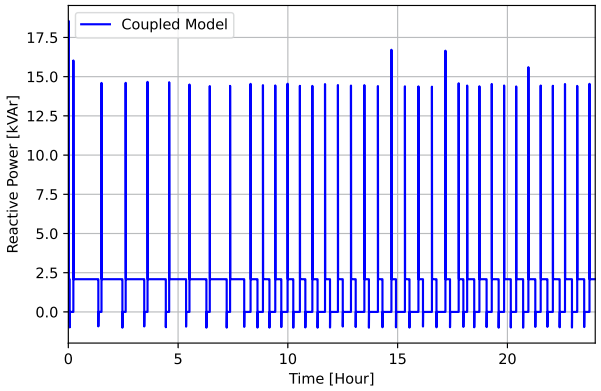


Figure 13. Motor-drive Heat Pump Reactive Power Consumption

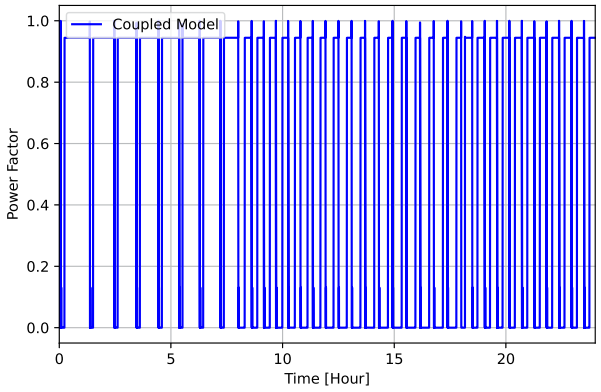


Figure 14. Motor-drive Heat Pump Power Factor

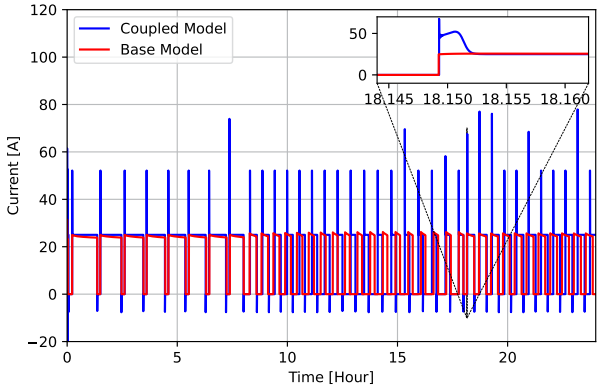


Figure 15. Motor-drive Heat Pump Current Consumption

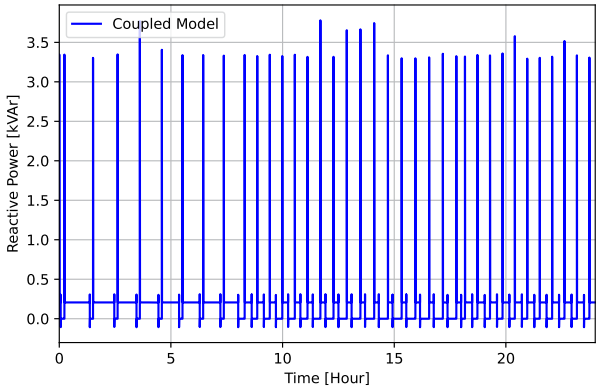


Figure 17. Motor-drive Pump Reactive Power Consumption

7.2.2 Pumps Electrical Monitoring

The pump power (Figure 16) ranges -1 kW and 4 kW in active power, while reactive power (Figure 17) varies from -0.5 kVAr to 3.5 kVar. On the other hand, the pump power factor (Figure 18) shows variations between 0 - 1 and the nominal power factor during operation is 0.6, while the pump current (Figure 19) ranges from -2A to 18A.

The coupled model exhibits significant transient oscillations in power and current, particularly during the initial and transition phases, which are not as pronounced in the base model. Instances of negative active power are also observed in both power and current data, indicating periods where the pump contributes power back to the grid.

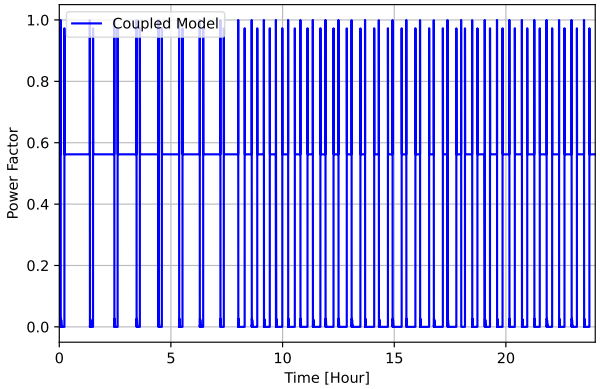


Figure 18. Motor-drive Pump Power Factor

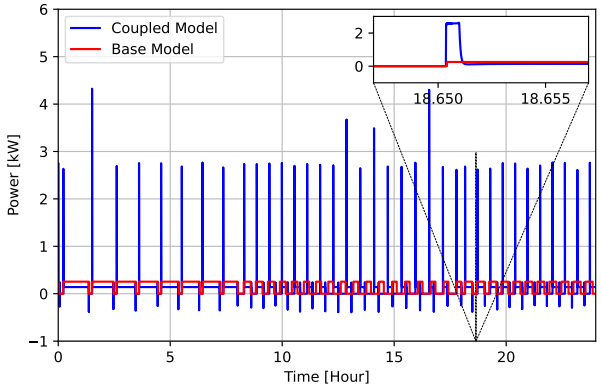


Figure 16. Motor-drive Pump Active Power Consumption

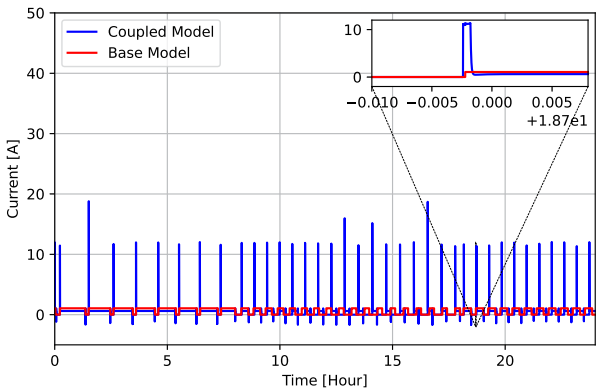


Figure 19. Motor-drive Pump Current Consumption

The coupled model is built using an adaptation of physics based equations (Le Fosse 2021) that is capable of simulating sudden variations in signal during turn on and turn off conditions. The adaptation reduces the complexity of drive system, thereby increasing the computation speed.

The aspect of B2G interaction of HVAC equipment's, presents both opportunities and challenges. On the positive side, B2G can enhance grid stability by distributing energy generation, particularly during peak demand. This reduces the load on centralized power plants and enhances overall grid resilience.

However, significant challenges arise with this approach. Managing the influx of energy from numerous residential sources adds complexity to grid operations, potentially leading to instability if not properly managed. The grid must be equipped with advanced technology to

Table 1. Comparison of Translated Model Statistics

Metrics	Base Model	Coupled Model
Constants	1149	1455
Free parameters	172 scalars	383 scalars
Parameter depending	589 scalars	998 scalars
Outputs	24	
Continuous time states	12 scalars	39 scalars
Time-varying variables	338 scalars	512 scalars
Alias variables	746 scalars	1246 scalars
Sizes of linear systems of equations	[2, 2, 2, 2]	[2, 2, 2, 2, 2, 4, 4, 2, 4]
Sizes of nonlinear systems of equations	[3, 1, 19]	[3, 5, 3, 2]
Sizes after manipulation of nonlinear systems	[1, 1, 2]	[1, 1, 1]
Number of numerical Jacobians	0	

Table 2. Comparison of Computational Metrics

Metrics	Base Model	Coupled Model
CPU-time for integration (s)	7.92	4.69
CPU-time for one grid interval (ms)	0.0916	0.0543
CPU-time for initialization (s)	0.086	0.084
Number of result points	86554	86746
Number of grid points	86401	
Number of accepted steps	18509	73064
Number of f-evaluations (dynamics)	307805	146002
Number of crossing function evaluations	102815	160670
Number of Jacobian-evaluations	1963	3839
Number of model time events	2	
Number of input time events	0	
Number of state events	75	171
Number of step events	0	
Minimum integration stepsize	8.0e-08	1.09e-08
Maximum integration stepsize	129	440
Maximum integration order	5	

balance supply and demand in real time, increasing operational complexity.

Frequent energy cycling can cause accelerated wear and tear on heat pumps, reducing their lifespan and increasing maintenance needs. Local infrastructure strain is another potential consequence. Most distribution networks are not designed for significant residential energy inputs, leading to increased stress on components like transformers. This could result in more frequent outages and necessitate costly upgrades to the local grid.

In summary, while B2G integration offers promising benefits for energy management, it also introduces challenges that must be addressed through advanced technology, careful planning, and infrastructure investment to ensure long-term sustainability and efficiency.

7.3 Computational Performance

The computational analysis highlights the comparative performance and complexity between base model and coupled model. The translated model statistics (Table 1) reveal that the coupled model demonstrates increased complexity with a higher number of constants, free parameters, and time-varying variables, leading to a more intricate system of equations. However, despite this increased complexity, the coupled model exhibits superior computational efficiency (Table 2), as indicated by a reduction in CPU-time for integration and grid interval processing. Specifically, the coupled model's integration time decreased by approximately 40% compared to the base model, showcasing its enhanced performance in dynamic simulations. The analysis of computational metrics further underscores the efficiency of the coupled model, with a significantly higher number of accepted steps and crossing function evaluations, alongside a more refined control over integration step size and maximum integration limits. This detailed comparison underscores the coupled model's capability to handle more complex dynamics while maintaining or improving computational performance, making it a valuable advancement over the base model. In the future, we plan to pursue additional enhancement of model by integration of soft starters and compare the impact of

coupled model in terms of transients and computational performance.

8 Conclusion

This study explores the interactions between buildings and the power system. A case study is considered that use heat pump and radiator with on/off controller to maintain the room temperature. The comparison between the motor-drive models with the baseline models has demonstrated that our motor-driven models offers a more realistic estimation of the electrical responses when the pumps and heat pump turn on and off. The coupled model is also 40% faster than the base model. This implementation of the motor-drive mechanical models will enhance our understanding of the dynamic interactions between buildings and power grids without compromise in terms of computational time. Furthermore, we intend to use our model in Building-to-Grid (B2G) activities to aid in designing and evaluating control strategies relevant to this field.

Acknowledgement

This research was supported by the DOE's Office of Energy Efficiency and Renewable Energy under the Advanced Manufacturing Office, award number DE-EE0009139. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DOE.

References

Fachini, Fernando, Marcelo de Castro, et al. (2023). "OpenIMDML: Open Instance Multi-Domain Motor Library utilizing the Modelica modeling language". In: *SoftwareX* 24, p. 101591.

Fachini, Fernando, Marcelo de Castro, et al. (2024). "Modeling of Induction Motors and Variable Speed Drives for Multi-Domain System Simulations Using Modelica and the OpenIPSL Library". In: *Electronics* 13.9, p. 1614.

Fachini, Fernando, Marcelo De Castro, et al. (2022). "Multi-domain power and thermo-fluid system stability modeling using modelica and openipsl". In: *2022 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, pp. 1–5.

Fu, Yangyang, Sen Huang, Yuan Liu, et al. (2019). "A multidisciplinary model to couple power system dynamics and building dynamics to enable building-to-grid integration". In: *16th International Conference of the International Building Performance Simulation Association, Building Simulation 2019*. International Building Performance Simulation Association, pp. 940–947.

Fu, Yangyang, Sen Huang, Draguna Vrabie, et al. (2019). "Coupling power system dynamics and building dynamics to enable building-to-grid integration". In: *Proceedings of 13th International Modelica Conference, OTH Regensburg, Germany*.

Gol, Ozdemir (1993). "Dynamic modelling of induction machines." PhD thesis.

Karassik, Igor J (2001). *Pump handbook*. McGraw-Hill.

Kundur, P., N.J. Balu, and M.G. Lauby (1994). *Power System Stability and Control*. EPRI power system engineering series. McGraw-Hill Education. ISBN: 9780070359581.

Le Fosse, Roberta (2021). "Dynamic modeling of induction motors in developing tool for automotive applications." PhD thesis. Politecnico di Torino.

Li, Guangdi et al. (2022). "Optimal Scheduling of Thermoelectric Coupling Energy System Considering Thermal Characteristics of DHN". In: *Sustainability* 14.15, p. 9764.

Stoecker, Wilbert F and Wilbert F Stoecker (1998). *Industrial refrigeration handbook*. Vol. 10. McGraw-Hill New York.

Wetter, Michael et al. (2014). "Modelica buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Modelica as Model Aggregator for holistic Architecture Validation of Electric Vehicles

Marcel Gottschall[1] Torsten Blochwitz[1] Andreas Abel[1] Alex Magdanz[2]

[1]ESI Germany GmbH, ESI Group, Germany, name.surname@esi-group.com
[2]Xcelerated Prototyping Inc., Canada, alex@xpincorporated.com

Abstract

Automotive OEMs and suppliers are facing recent challenges in the development process, induced by ever shortened product cycles, further distributed development as well as increasing demands for virtual testing and certification using virtual proving grounds or digital twins.

This paper presents a real-life demonstration of a federated, seamlessly integrated design process for a complex cyberphysical system (electric truck), where simulation is used for early-stage performance validation and decision making. Since holistic, but abstract architecture models created in systems engineering discipline contain relevant information with respect to logical system structure and allocated requirements, the simulation domain will benefit from a cross domain linking of model artefacts. By aligning system interfaces across model abstractions and augmenting logical models with physical information, behavioural model templates for design can be generated in a smart, traceable and automated fashion. With the additional information of requirements allocated to certain architectural components in those abstract architecture models, it is demonstrated how scenario-based component and system simulation will contribute to analysis tasks like architecture exploration or specific design optimization in efficient, continuous engineering environments.

Keywords: Digital Thread, MBSE, Virtual Testing, Electric Vehicles Architecture

1 Introduction and Engineering Ecosystem

Ongoing digitalization of todays product lifecycle, from development to operation, creates new opportunities, but also new challenges need to be handled introduced by the ever increasing complexity of products and their underlying processes. Traditionally, product development is divided into stages with clear separation, based on each discipline's view of the specific system of interest (see Figure 1), to enable systematic processes at each design step. Systems Engineering (SE) is such major process, taking place at the architectural level of mechatronic or cyberphysical systems[1], physical performance design (1D) or geometrical specification (3D) are successors in a natural top-down workflow. As shown in Figure 1, the very first phases of a digital, holistic product definition, from stakeholder needs to system functions and logical implementation to the physical realisation, are commonly included in the term *model-based systems engineering* (MBSE).

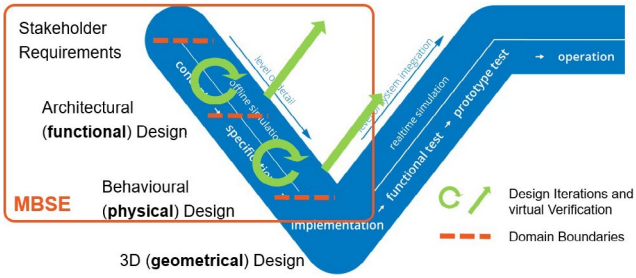


Figure 1. Generic representation of product lifecycle with focus on early-stage phases of design and the involved engineering disciplines, commonly allocated to model-based systems engineering terminology (MBSE)

Systems engineers describe the principal system architecture and its requirements to be fulfilled in early development stages with abstract models, most commonly in *systems modeling language* SysML or forks thereof, (Object Management Group 2022). Despite rare cases, these logical models are lacking real representations of the physical behavior of the system. On the other hand, those are often created in later design stages by design engineers for detailed analysis of the system use cases. The Modelica modeling language is a well established method, in particular for such analysis of the dynamic, non-linear behavior of multiphysics systems. Due to limited simulation capabilities in architecture tools (SysML), there is still no possibility to effectively and conveniently link more sophisticated physical models created in different modeling platforms by distributed engineering teams involved in the design. However, providing physical system models to systems engineers would enable complex tasks, like architectural exploration and early-stage decision making as well as virtual testing of allocated requirements. In addition to that, it is important to remark, that bottom-up processes like change management and impact analysis (e.g. in case of findings on a lower, more detailed level) will benefit, or in the first place will become possible, by sim-

[1]products or systems containing a physical part, often called *plant* and a software or logical part, often called *controller*

ulation capabilities on (early stage) holistic views of complicated and complex products like cyberphysical systems with tight interdependencies between the different components.

Hence, this paper presents a new approach by integrating physical (1D) models into an architecture representation of an electric truck, which is transferred from modeling (SysML) to simulation domain in early design stages. This way, physical models and corresponding data like parameters are linked to SysML model representations of requirements, functional and logical views, as a key aspect of traceability and certification by simulation. Based on the linked data approach, relevant information from the SysML model is automatically transferred into a corresponding Modelica model components library with all system- and subcomponents (e.g. battery, drivetrain, battery cooling system) and their interaction represented by Modelica connections. Incorporating existing 1D models available in such federated, multitool engineering environments and augmenting these architecture component models with certain stimulation and evaluation creates dedicated testmodels ready to use in virtual validation campaigns. It is emphasized, that such closed loop, fully automated testmodel execution and analysis, enables early-stage architectural and design decisions as shown in Figure 1, their monitoring and evaluation considering full variability in large product lines with respect to a complete set of requirements, hence enabling agile design changes in case of failed tests.

2 Motivation and Implementation

Looking at the very common visualisation of product lifecycle by the V-Model shown in Figure 1, explains that systems architecture design and system performance simulation are direct neighbours, where the latter is consuming major output from the previous stage and vice versa. However, these disciplines are separated by their specific workflows, tools and artefacts they deal with. Digitalisation allows to break these silos of knowledge and establish a consistent and continous information flow across the original boundaries.

From this perspective, system simulation is the perfect tool to execute verification and validation steps on architectural level, providing proof and confidence on performance to enable correct decisions in typical large architecture design spaces before going down in detail and spending effort on the next level of system description like CAD or FEM. Rework costs and time-to-market are significantly decreased by such approach, thus implementing an agile methodology, known from software design and mapped to physical systems in Figure 2.

Due to its design and because of the multiphysics system character, system simulations and in particular those implemented by Modelica language, e.g. (Modelica Association 2021), are well suitable for several validation stages along the design cycle. Starting with simplified or

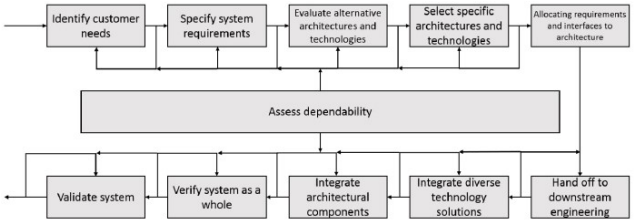


Figure 2. Detailed pocesses in early phases of product development for integration between architecting and simulation in agile systems engineering methodology, (Douglass 2016)

even surrogate model descriptions of components in very early phases of architecture drafts with lots of unknowns, down to more detailed and sophisticated models at higher maturity levels of the design throughout the system decomposition phase, system simulation perfectly serves the intended purpose. Hence, it is the key integrator between architecture and geometry of a system.

The benefits of reusing information between these domains have been already discussed and demonstrated in several ways and projects, e.g (The INTO-CPS Association 2018). Applying a predefined SysML profile in architecture models allowed the automatic preparation of co-simulation and proper parametrization between different simulation objects, thus representing a system model. However, the complexity of the models and logical structure have been quite limited, and the creation of dedicated diagrams providing the required information imposed additional modeling effort aside of the systems engineering process.

Nevertheless, to emphasize the natural combination of architecture and simulation, *Modelica Association* defined a dedicated standard called SSP[2], to apply simulation models in an architectural (= *structure*) context, (Modelica Association 2022b). Different to that approach and the available implementations, in this paper the relevant information is reused from the systems model (SysML). Such data, at first the logical structure or decomposition, meaning the different subsystems and components as well as their interrelations, enables an automatic generation of a model template and corresponding library as shown in Figure 3. This template replicates the system structure and hierarchy without any modeling effort. Hence, the design process time is significantly decreased, while it becomes more reliable regarding missmatching subsystems and components as the degree of freedom of the design engineer is limited because of predefined ports and connections. However, it should be noted, that some information have been traditionally missing in systems engineering domain (SysML) and need special treatment to bridge the gap from abstract functional system perspective to behavioural or physical view, (Cederbladh et al. 2024). Similar to SSP approach, the actual performance simulation models will be then integrated by means of

[2]System, Structure and Parametrization

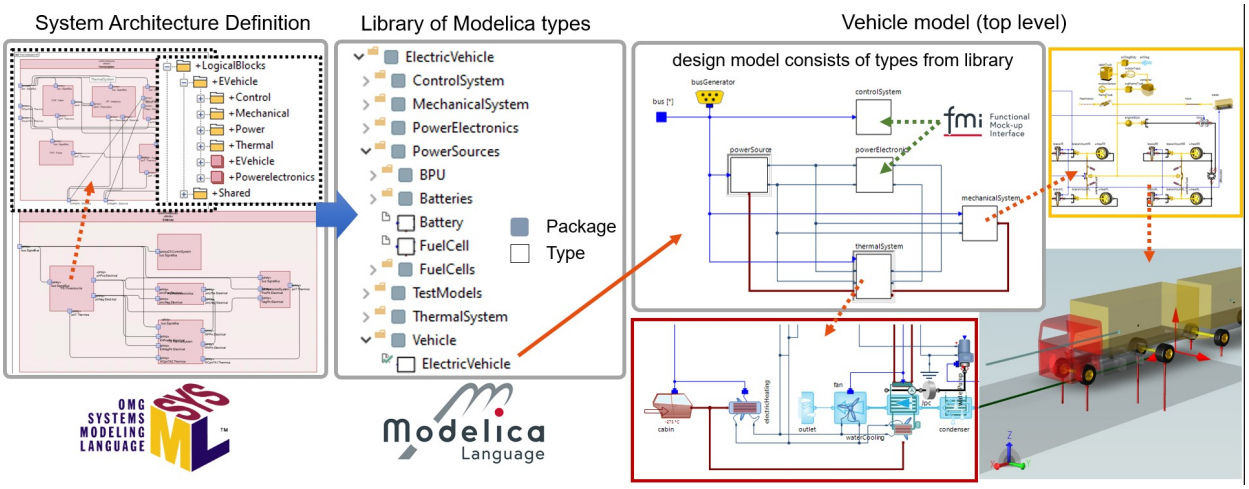


Figure 3. Cross-domain process to replicate system architecture information (logical structure) in automatically generated simulation templates to aggregate native Modelica models or existing models from other sources by means of FMI

functional mockup interface FMI or native Modelica components into these architectural templates. In particular, the latest version 3 (Modelica Association 2022a) with the support of physical connectors is a major improvement towards automation and user convenience for the aspect of continous integration between architecture and simulation, see next section for details from practical application perspective.

In a second step, the requirements or stakeholder needs, which are connected to architectural components by trace links in the SysML model, are used to extend the previous architecture simulation models for the purpose of mission- or scenario-based virtual architecture validation, as introduced above. A more detailed, walktrough visualisation of MBSE artefacts integration from requirements to virtual testing is given in (Gottschall, Binder, and Castel 2022).

At this stage in the design cycle, the application of the SSP technology mentioned above becomes obvious by exporting such full architecture simulation models in a tool agnostic model exchange container for collaborative use cases.

In order to achieve such digital cross domain integration (not limited to architecture and performance), we developed and applied a *linked data* approach, based on microservices which are compliant to open standard specification OSLC[3], (Open Services Project 2021). As shown in Figure 4, these webservices are acting as a middleware between frontend and backend tools, exposing all relevant information, collected in a multidomain datamodel of overlapping entities, and providing consistency throughout the various managed artefacts that are created by the stakeholders of the process. Engineering tools are connected by clients which implement discipline specific workflows on that data. This way, the digital thread approach becomes tool agnostic, since the frontend and backend tools can be replaced while the datalayer stays intact. Such non intrusive implementation supports the *best tool for the job* paradigm, allowing the user to keep his established ecosystem and dedicated workflows.

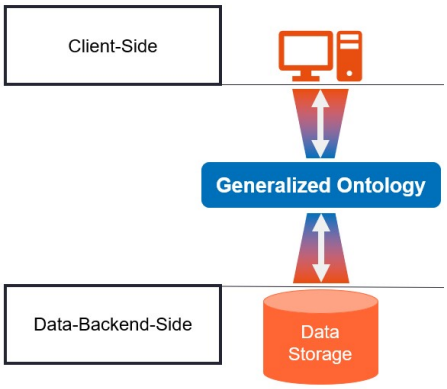


Figure 4. Middleware integration approach between engineering frontends (e.g. simulation tools) and data backends establishing coherent engineering artefacts to achieve a tool agnostic digital thread along product lifecycle with contributions of various disciplines

The benefits of continuous integration between systems engineering and simulation for early-stage development tasks will be demonstrated in the next section by top-down virtual design of a rather complex electrical vehicle (EV) system, applying incremental performance validation and corresponding decisions to highlight the value of scenario-based system simulation applications.

3 Complex System Development Process Demonstration

As shown in Figure 1, a representative development process starts with high level, mostly abstract requirements or stakeholder needs. The subsequent system description and

[3]Open Services for Lifecycle Collaboration

decomposition in the design phase for products or systems with sufficient level of complexity (like an EV) follows MBSE methodology in a top-down fashion, including the major abstraction layers and their relations:

- Requirements (R) are *satisfied* by functions (to be provided by a system)
- Functions (F) are *fulfilled* by logical components (contained in a system)
- Logical components (L) are *implemented* by physical models
- Physical models (P) are used to validate systems (subsystems, components, et.) against requirements

Such formal, hierarchical process allows breaking down a complex design task, where requirements are progressively derived and propagated along the different levels of detail or abstraction from system (e.g. electric vehicle) to subsystem (e.g. electrical system) to components (e.g. battery), always based on the simulation results and decisions made the step before, indicated by the green arrows in Figure 1.

Use Case and Sample Tooling

With the demonstration, a model-based, incremental top-down design and operation optimization use case is executed, applying either an electric truck (long distances) or electric bus (short distances, not shown) ecosystem, which results in different system architectures to be selected based on the simulation results. Here, the exemplary engineering tooling listed below is used for visualisation purposes at the different stages of the process (as mentioned above, tools can be replaced by the user):

- *PTC Windchill Modeler* (requirements and architecture modeling, SysML)
- *ESI SimulationX* (1D modeling and simulation environment based on Modelica)
- 3rd party 1D model sources like *Simulink, GTSuite, Dymola*, etc. providing FMU
- *ESI VCM*[4] (webbased virtual test management and executation environment)

Depending on the type of utility vehicle, e.g. longrange truck or city bus, the corresponding target and missions are specified and applied for the product level stakeholder needs, exemplarely listed below. Such performance requirements will be the entry point into and drive the design and verification process considering certification standards like ISO 8714, or safety regulations on components like fail safe battery design compliant to ISO 6469 or SAE J2929:

[4]Validation Campaign Manager

- **Range:** *The vehicle must be capable to serve a distance of 350 km (+50 km safety margin), or a cycle time of 4,5 hours, respectively before recharge or refill is required.*
- **EnergyConsumption:** *The specific energy consumption must be below the threshold of 1,20 kWh/km on customized, specific missions at maximum payload.*
- **ClimbingPower:** *The vehicle shall maintain a velocity of 80 km/h at a road gradient of 7 percent with maximum mass.*
- **CabinComfort:** *The thermal system must maintain a cabin temperature between 18 - 25 deg Celsius, in outside operating environmental conditions between -40 and +50 deg Celsius with respect to heating and cooling performance.*
- **BatterySafety:** *The maximum battery temperature under peak load conditions must stay below 70 deg Celsius.*

Apart from technical requirements, also economical aspects may be considered and evaluated based on the simulated loadcases (still on high, abstract level) like

- **Costs:** *The estimated, selected vehicle architecture operating costs must stay below 0.40 €/km, also considering personnel costs spent on downtime like recharge cycles.*

The payload of the EV or capacity of the power source are design parameters among others listed in Table 1, reflecting variability between different configurations of the same system architecture. It should be remarked, that the given high level requirements are cascading along the system decomposition and maturizing, where additional requirements on the sublevels, e.g. recuperation or charging power, are derived and validated accordingly.

Table 1. Examples of architecture design parameters for the EV

Subsystem	*Parameter*	*Name*
Powersource	Capacity	cap
Mechanical System	No. of Motors	nMot
Mechanical System	Wheel Dimension	rWheel
Mechanical System	Total Mass	mTot
Thermal System	Climatisation Power	hP
Thermal System	Trailer Cooling Power	cgP

Process and simulation-based Decision Making

Figure 5 summarizes the seamless integrated, collaborative workflow using MBSE principles and virtual testing to verify architecture performance. SysML models are created by formal processes following a strict R-F-L methodology to decompose the complex system of interest, (Aleksandraviciene and Morkevicius 2018; Weilkiens

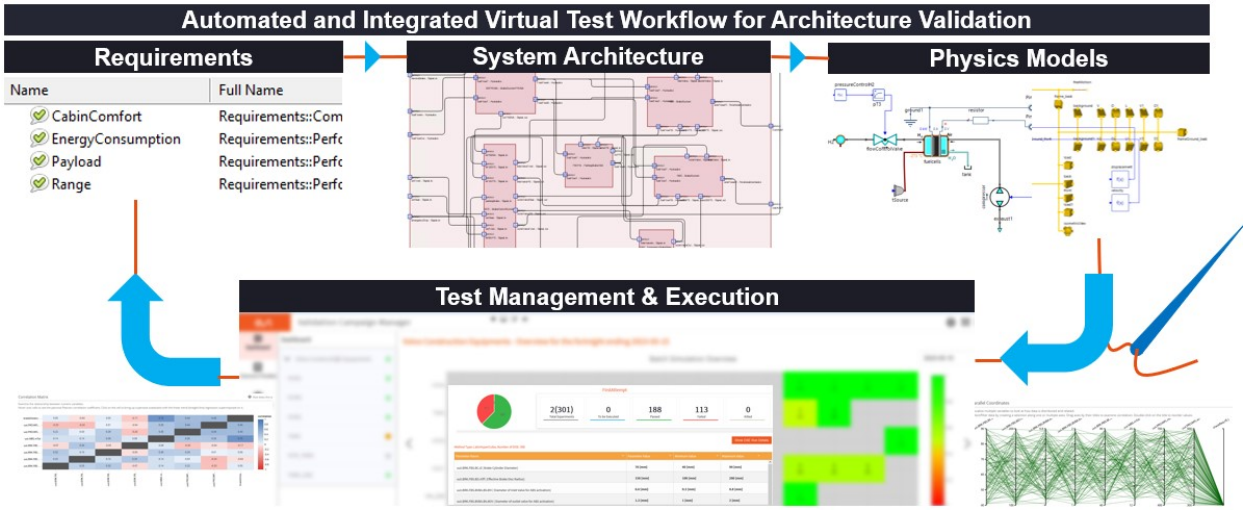


Figure 5. Integrated digital thread workflow spanning MBSE disciplines from requirements to architecture to physics for simulation and application of the physical models in virtual testing scenarios for early stage validation and design decision feedback

2014). In such hierarchical approach, lower level requirements and specifications are derived from higher level engineering results (e.g. *system* L defines F on *subsystem*, *subsystem* F defines R on *components*, etc). Applying "shift-left" paradigm by virtual design (with increasing model capabilities and fidelity along the process) enables decision making based on simulation results and the automatic propagation of top level stakeholder needs throughout the development task. In particular, use cases in this publication like

1. Architecture exploration for electrical system (battery or fuel cell) and thermal system (cabin heating with resistor or heatpump) with available or derived simplified models e.g. transfer functions, validating against high level performance indicators defined by the requirements above

2. Scenario-based design optimization regarding sizing and performance of subsystems on previously selected architecture level, e.g. number of drive motors in mechanical system, or deriving geometrical parameters and requirements for subsequent 3D design (CAD)

3. Operation and mission optimization for a given/ frozen design of the EV for in-service phases of the system

4. Prediction of performance degradation after 5 years or 2.000 cycles in operation, e.g. using aging models of power source system, and verification against lifecycle requirements

will benefit from continous artefact integration (data) and workflow automation, towards future AI[5] supported engineering business. Further downstream, to enable a traceable application of the generated physical simulation models with respect to scenario-based architecture and system performance validation, relevant information for this purpose is reused from the abstract systems model in SysML. More specifically, standard systems engineering entities like structural diagrams (L) shown in Figure 5, are augmented by stereotyping to specify the nature of physical connections between the different elements (*blocks*) and describe the architecture of the design system. Their allocation to certain functions (F) allows a filtering of components throughout the model generation in the simulation tool, hence supporting the focus of dedicated engineering teams. With the *verification* link given in requirement diagrams (R-L), specific parts of the system architecture can be applied in dedicated virtual testing models representing a specific scenario described by the linked requirement. As shown in Figure 6, this way, the physical design model (P), realizing the logical structure of the architecture, and the requirement (R) become ingredients of the actual testmodel. This testmodel generation is semi-automatic by instantiating the corresponding design component (as "Unit under Test" *uut*) that is linked to the requirement, and ensures traceability along the artefacts generation. Since requirements are often not formalized but given in natural language, the design model augmentation by stimulation and evaluation corresponding to the requirement is manual modeling effort. However, in previous studies, these aspects have been already automated by applying standardized interfaces for test automation[6].

As already mentioned above, the aspect of *configurations* of a product becomes more and more important in engineering workflows incorporating modeling and simulation shown in Figure 5, particularly for complex cyber-physical systems OEMs or suppliers have to handle full

[5] artificial intelligence

[6] ASAM XiL

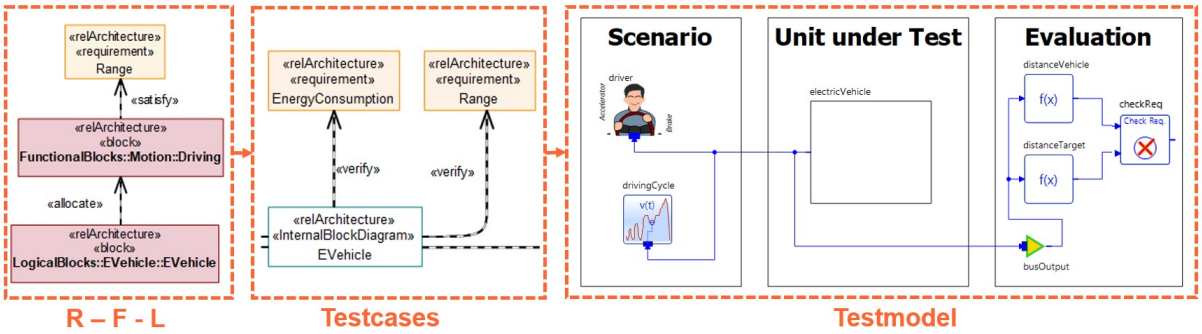


Figure 6. Concept to reuse tracelinks between SysML artefacts to generate testmodels in Modelica. Allocation to system functions allows filtering (left), verification links between logical layer and requirements (center) allow augmentation of design models with corresponding stimulation and test verdict (right, *failed* test) for automatic virtual testing processes

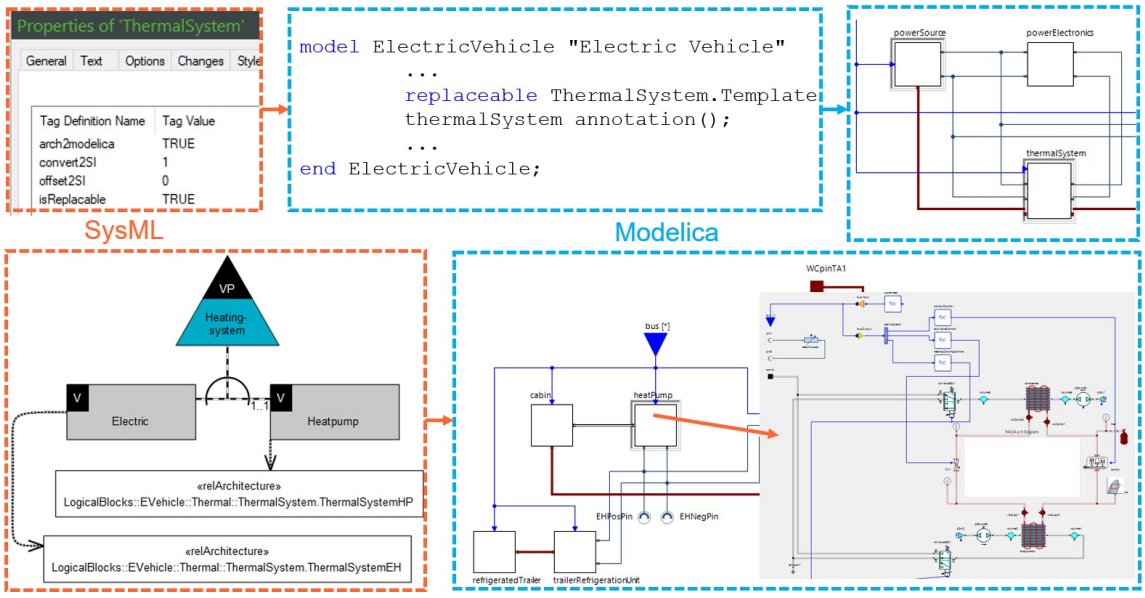


Figure 7. Automatic cross-domain transfer of product configuration specification by utilizing SysML entities like *stereotyping* (top left) and *variant diagrams, decision sets* (bottom left), and the Modelica *replacable* mechanism (right), to describe different architecture or component implementations, demonstrated on the *Thermalsystem* and the inner, selected *Heatpump* structure

product families. The need for cost reduction drives the modularization and identification of commonalities in design, and continous integration is a measure to achieve that. Both, architecture and simulation provide techniques to enable cross-domain transfer of such information, see Figure 7 for an overview on the implementation. Two main aspects need to be handled:

1. Express the variable component in the SysML model (L) by stereotyping and tagging as shown top left and transfer that information to the corresponding, automatically generated Modelica template (P) using the builtin replacable/redeclare mechanism in a certain instance of a model, e.g. for testing, as shown for the *ThermalSystem* block

2. Express the various, potential configurations of the system of interest as shown bottom left for the *Heatpump* or *Resistor Heating* implementation of the *ThermalSystem*, SysML tools offer dedicated artefacts like *variant diagrams* and *decision sets* to represent *variability* in a certain system, such expressions are standardized by ISO 26550 and variability or variants modeling is becoming a crucial part of SysML v2 description language (Object Management Group 2023), potentially leading to further enhanced capabilities and automation in this regard

In addition to the systems structure, the design parameters, that are defined on architectural or systems engineering level and attached to the different blocks/components (L), need to be handed over to simulation domain. Such parameters, like in Table 1, have to be used for testcampaign definition to reflect the design space definition for each physical implementation (P). Moreover, considering the corresponding parameter ranges in an automated fashion throughout the test description serves the use case of estimating the impact of real product deviations or toler-

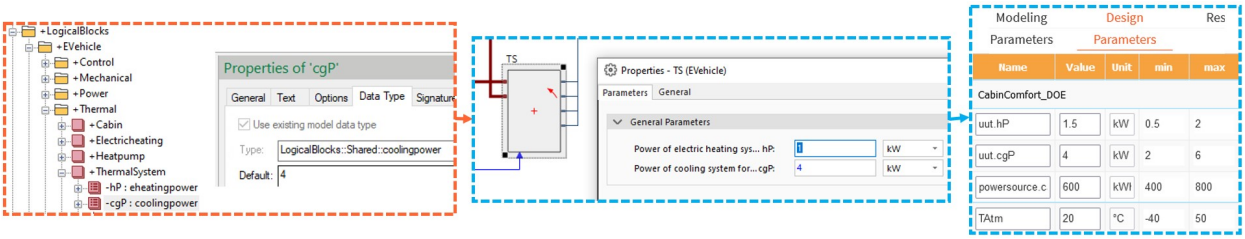


Figure 8. Automatic cross-domain transfer of global design parameters (default value, ranges and units) defined on architectural level by utilizing SysML entities like *Block Property* (left), and corresponding Modelica implementation (center) to reuse in VCM for experiment definition (right), visualised for the *Thermalsystem* verification

ances ("as manufactured"), to achieve a robust design right from the beginning. Apart from such conventional use of parameters for system or component sizing, the Modelica language offers another beneficial option for the application of architecture and structural decision making. Integer values on logical blocks are used to describe the number of instances of a same component inside that block, without changing the architecture with respect to physical domains of interfaces. A visual example is the *No. of Motors* parameter in Table 1, where the external structure and connectivity of the mechanical subsystem does not change, but the desired number of electrical machines connected to the outside interface (*multiplier*) is transfered from systems engineering level to simulation and vice versa. Optimization tasks like "Is one big motor better or worse regarding overall energy consumption compared to two smaller ones?" are significantly improved with better traceability and user convenience across the different development domains.

Again, the standard mechanisms of architectural and physical modeling are used to express and transfer the desig nparameters, their description, default values and ranges, Figure 8. In a continous integration implementation, the test management and execution system in Figure 5 lists these parameters automatically, to be used for the definition of the different experiments/ campaigns, either single runs or multiple variants simulations in applications like design space exploration or optimization (design of experiments).

Since the different architectural components discussed above, are linked to corresponding requirements that they should fulfil, Figure 6, the information of test cases which require a simulation model is automatically provided to the simulation engineer in the physical modeling tool, see Figure 9. Similar to the template generation of design models shown in Figure 3, the testmodels will be automatically generated by instantiating the correct modeling component when the user selects a test case in the list.

These features highlight the contribution of cross-domain integration to the scope of more efficient and reliable, collaborative engineering workflows. Aside of that, it should be emphasized again, that the underlying linked data ensures strict traceability, and continous integration allows

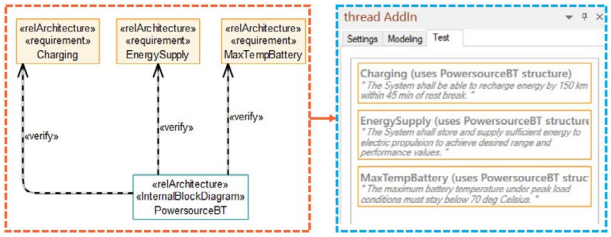


Figure 9. Representation of requirements that need to be tested by performance models in SimulationX Addin (right), defined by *verifies* links on architectural models in SysML (left), shown for the *Powersource* subsystem, see also Figure 6

for automatic top-down change transfer along the process. Changes in parameters or architectural structure are propagated and trigger model regenerations. On the other hand, fully bi-directional automatic bottom-up updates, like from 1D simulation results to architectural changes, are ususally not allowed in real development processes of complex systems, as they require impact assessment on higher, holistic system level.

Commercial Electric Vehicle Example

With respect to page limit, the technical engineering solutions described above, will be visualised by one specific example of a long-range truck design and sizing. Based on the explained high level requirements, system simulation is used to verify the performance of different, potential system architectures. This is a two-stage process, where both, the architecture evaluation and the following system and component sizing are verified on a detailed mission simulation. Such inherent incremental maturity rise along the development means, that the structure (L) of the system determines the functions (F) to be provided by the subsystem (and so on), hence lower level requirements are derived based on higher level simulation results. Close to state of the art parameter settings of heavy electric trucks with cooled trailers have been applied for the simulations below, as they became available and published recently.

Saying that, the scenario setup for the "architecture exploration" use case consists of two conditions for summer (30 °C) and winter (-15 °C). As mentioned above, it should be investigated which configuration of the

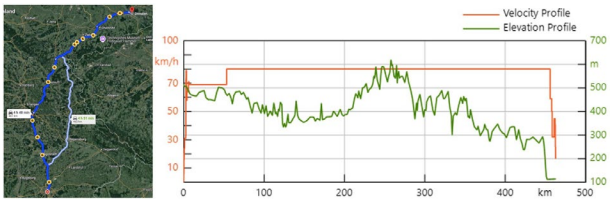


Figure 10. Examplary routing from Munich to Dresden with average velocity and elevation profiles exported by a route planner

- *PowerSource*: Battery vs. Fuelcell
- *ThermalSystem*: Heatpump vs. electrical Heating

would be the best choice with respect to the decision criteria of specific energy consumption for driving as well as climatisation of cabin and trailer. The actual models in the different domains have been created as explained and visualised above and considering the resulting test matrix, 8 simulations have been executed. Please note, that the physical implementations of the systems and subsystems of interest (*PowerSorce* and *ThermalSystem*) are simplified at this stage. Usually, at these very early phases of design more detailed models are simply not available, or an enormous number of simulation runs is required within multi-dimensional design space exploration of complex systems and the computational performance has to be maximised. Moreover, it should remarked again, that the physical models that are plugged into the architecture template by means of FMI originate from different sources, or are represented by native Modelica components.

However, all "full system" evaluations are done using the exemplary, envisioned route for the truck shown in Figure 10. This is crucial, as the different system implementations have different efficiencies and capabilities, e.g. regarding recuperation. Hence, the route profile of the specific mission has significant impact on the selection of a certain architecture configuration. Relevant information from the route like elevation (inclination) and average velocity are imported to a "drivingCycle" block, compare Figure 6, in the testmodels to apply proper conditions for stimulation. Please note, the maximum speed for heavy trucks is limited to 80 km/h on highways. It should be also remarked, that the tested route has a direction, means that the results will differ when going in reverse direction. Such considerations are subject to common trade-off studies.

A qualitative analysis is given in Figure 11 for first step decision making. With respect to the overall specific energy consumption shown on top, it becomes obvious that the battery configuration is the better choice, independent of the ambient conditions. This is because of several reasons. The major driver is the efficiency of the fuelcell to convert the hydrogen into electrical energy, in comparison to the battery that stores the required energy

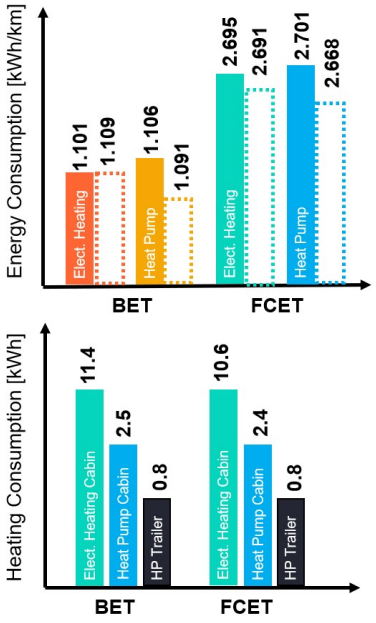


Figure 11. Qualitative comparison of performance numbers for the battery electric (BET) and fuel cell electric configuration (FCET), top: overall energy consumption for electric heating and heatpump configuration, each at summer (colored) and winter (dotted) ambient conditions, bottom: heating energy for the cold conditions, distributed over main consumers

directly. However, depending on the overall scenario, it might be necessary to extend the system boundaries and consider the costs and effort for external electrical energy (for charging) as well, which might in turn change the outcome as well. Aside of that, the energy comsumption in could conditions is lower for both configuration because of the lower demand by the trailer cooling system. Also, it should be remarked, that the electric heating cabin climatisation variant, seems to be slightly more efficient for the summer scenario, compared to the more complex heatpump system. However, this is simply to the fact that the eletric heating cannot cool the cabin in hot conditions, thus there is no energy consumption at all in this case, but it violates the functional requirement of "cooling" which is indicated by a *failed* test for *CabinComfort* requirement.

Looking at the *ThermalSystem* performance at the bottom of Figure 11, shows a clear benefit of the heatpump configuration in cold conditions, because of the much higher efficiency - as expected in this demonstration.

So, the overall architecture evaluation on this specific customer mission results in a decision for the battery-heatpump configuration. Based on this outcome, the detailed design of the specific subsystem implementations is taking place. The same architecture representation in the system simulation tool is used, but the simplified physical models of *Heatpump* and *Battery* are replaced by enhanced, more sophisticated model components, (Pukrushpan 2003; Hariharan, Tagade, and Ramachandran 2018), as shown in Figure 7. In this step, subsystem

Experiment Name	Experiment Status	Check Criteria	Required Value	Final Value
EnergyConsumption	Passed	checkReq.rf[-]	1	1
Range	Passed	checkReq.rf[-]	1	1
ClimbingPower	Failed	checkReq.rf[-]	1	0
Acceleration	Passed	checkReq.rf[-]	1	1

Experiment Name	Experiment Status	Check Criteria	Required Value	Final Value
MaxTempBattery	Passed	checkReq.rf[-]	1	1
CabinComfort	Failed	checkReq.rf[-]	1	0

Experiment Name	Experiment Status	Check Criteria	Required Value	Final Value
Recuperation	Passed	checkReq.rf[-]	1	1
Charging	Passed	checkReq.rf[-]	1	1

Figure 12. Test execution and requirements fulfilment overview in VCM triggering design decisions, models in the *driving performance* domain (top), models in the *thermal performance* domain (center), and models in the *electric performance* domain (bottom) for a given, exemplary parameter setting

and component requirements are derived incrementally throughout test execution with the detailed models, requirements on charging features that depend on the type of architecture.

Figure 12 provides an overview of test results (with respect to *passed/failed* verdict) using the configuration identified above, with a certain, exemplary setting of design parameters selected from the given ranges as shown in Figure 6 (right). It can be seen, that most of the requirements are met by the design in the described mission. However, the *ClimbingPower* and *CabinComfort* tests are failed. The analyis requires a more detailed view into the transient behaviour of a certain simulation, to identify the root cause. As an example, the latter testcase is executed in Figure 13 in a "design of experiments" run, to figure out the impact of input conditions and design parameter values. It can be seen, that some, mainly the coldest ambient conditions, do not fulfil the requirement of a cabin climatisation between 18 and 25 deg Celsius (left) for the particular parameter setting. The VCM provides various analysis and data analytics tools and capabilities. With the parallel coordinates on the right, the ranges of relevant parameters (here the *climatisation power hP* and the *trailer cooling power cgP*) can be limited to valid combinations, that satisfy the *CabinComfort* requirement. Such evaluation will be feed back into the systems engineering design phase and can be further automated, for applications like functional optimization.

Once the ideal component design and parameter setting regarding the different requirements are identified, the configuration can be frozen and used for detailed transient analysis and load case generation for design steps in the 3D geometry domain downstream the V-cycle, see section below. In example, Figure 14 visualises the evolution of battery charge, recuperation power and mechanical braking action along the route (excerpt), testing the *Range* and *Recuperation* requirement. Aside of that, further engineering tasks like mission optimization, or operational predictions like aging of battery power source are supported by the digital twin character of the architecture simulation models.

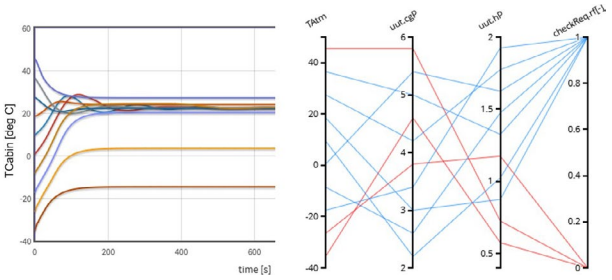


Figure 13. Example of visualisation in VCM from the *CabinComfort* testcase executed at 8 different ambient conditions for demonstration of engineering tasks like design optimization and parameter identification, transient results left, parallel coordinates to narrow down valid parameter ranges within design space satisfying the requirement (blue)

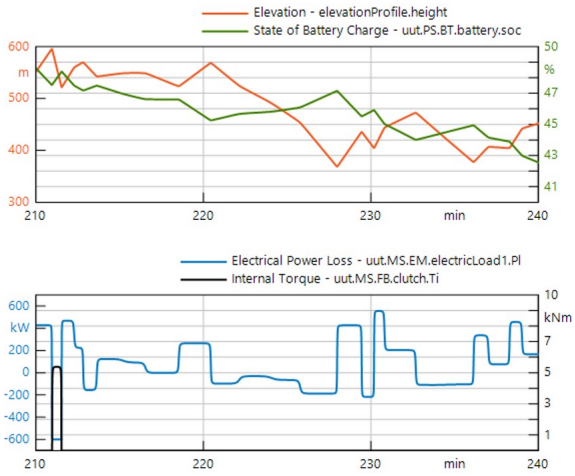


Figure 14. Example of a zoom into transient results from the *Range* testcase running the mission in Figure 10 for design optimization, bottom diagram indicates mechanical breaking action when maximum recuperation is achieved on steep descents

Finally, it should be mentioned that the results and derived architecture and components look different for the system of a city bus because of the different scenario. However, the major benefit of such continous integrated design process, is the agility to quickly identify new architectures on changed requirements.

Reuse of Architecture Performance Models

As mentioned above, the simulation results gained in the demonstrated process will serve as inputs further downstream the product development cycle, for geometrical definitions (system sizing) or providing load cases for

Figure 15. Example of 1D physical model (ABS brake) integrated into 3D FEM model by means of FMI for realistic pre-crash simulation

detailed 3D CAE analysis tasks. Aside of these natural cross-domain interactions, particularely the Modelica models generated for performance validation of system architectures or subsystems can be reused as FMI integrated components, for sophisticated, more realistic, scenario-based 3D FEM simulations, as shown in Figure 15. With such enhanced coupling, certification credits for crash simulations on safety critical battery and fuel cell architectures (e.g. ISO 23273) in the challenging EV domain are enabled and already demonstrated.

4 Conclusion

Establishing continuous and agile workflows in design and validation of complex cyberphysical systems, by enabling collaboration on heterogeneous tool and stakeholder ecosystems in early phases of development, leverages the potential of ongoing digitalisation as shown in the present demonstration. It addresses currently existing process and traceability gaps between the engineering disciplines of requirements management, architectural design, physical development and virtual performance and puts system simulation in a broader, holistic system context. This way, 1D simulation evolves from an isolated activity, acting in a silo with well known issues and friction when it comes to integration, towards an integral part of virtual development applying and following model-based systems engineering methodologies to master present and future process and product complexity. With its ability of serving as model aggregator, Modelica plays a crucial role in collaborative multipartner processes, examplary for complex systems, early stage validation.

With the presented digital thread implementation, not only design and verification becomes more efficient, reliable and collaborative, but also sales engineering tasks like RFP phases (request for proposal) benefits from much reduced task cycle times. The cross-domain variability support allows the fast and reliable selection of the best configuration in a complex product family for a customer specific mission.

In a next step, the architecture identification can be automated by enabling experiment definitions on component implementations. With the reuse of dedicated SysML artefacts (variant diagram, decision set, etc) in the test management system (e.g. VCM) design space explorations can be easily executed and evaluated. This would enable further AI support by simulation-based decision making towards more autonomous processes.

Finally, with the upcoming layered standard on *SSP traceability*, further automated test model generation with respect to stimulation and evaluation of performance models, based on reusable meta data for test specification, is expected.

References

Aleksandraviciene, Aiste and Aurelijus Morkevicius (2018). *NoMagic Magic Grid - Book of Knowledge*. 1st ed. Vitae Litera.

Cederbladh, Johan et al. (2024). "Correlating Logical and Physical Models for Early Performance Validation - An Experience Report". In: *IEEE Systems Conference SYSCON2024*. IEEE.

Douglass, Bruce Powel (2016). *Agile Systems Engineering*. 1st ed. Morgan Kaufmann Publishers. ISBN: 978-0-12-802120-0.

Gottschall, Marcel, Bastian Binder, and Alexis Castel (2022). "Towards Certification by Simulation with model-based continuous Engineering Processes showcased on eVTOL Application". In: *78th VFS Forum and Technolgy Display*. Vertical Flight Society.

Hariharan, Krishnan, Piyush Tagade, and Sanoop Ramachandran (2018). *Mathematical Modeling of Lithium Batteries*. 1st ed. Springer. ISBN: 978-3-319-03526-0.

Modelica Association (2021). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.5*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.5/MLS.html.

Modelica Association (2022a). *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 3*. Tech. rep. Linköping: Modelica Association. URL: https://fmi-standard.org.

Modelica Association (2022b). *System Structure and Parameterization Version 1*. Tech. rep. Linköping: Modelica Association. URL: https://fmi-standard.org.

Object Management Group (2022). *OMG Systems Modeling Language Version 1*. Tech. rep. Massachusetts: Object Management Group. URL: https://www.omg.org/spec/category/systems-engineering/.

Object Management Group (2023). *OMG Systems Modeling Language Version 2 - Language Specification*. Tech. rep. Massachusetts: Object Management Group. URL: https://www.omg.org/spec/category/systems-engineering/.

Open Services Project (2021). *Open Services for Lifecycle Collaboration Version 3*. Tech. rep. Open Services Project. URL: https://open-services.net/.

Pukrushpan, Jay Tawee (2003). "Modeling and Control of Fuel Cell Systems and Fuel Processors". Doctoral dissertation. The University of Michigan, Department of Mechanical Engineering.

The INTO-CPS Association (2018). *The INtegrated TOolchain for Cyber-Physical Systems: a Guide*. Tech. rep. INTO-CPS. URL: https://into-cps.org/fileadmin/into-cps.org/Filer/INTO-CPS-Manifesto.pdf.

Weilkiens, Tim (2014). *Systems Engineering mit SysML/UML*. 3rd ed. dpunkt. ISBN: 978-3-86490-091-4.

Multiphysics Acausal Modeling and Simulation of Satellites Using Modelica Library

Salvatore Borgia[1] Francesco Topputo[1]

[1]Department of Aerospace Science and Technology, Politecnico di Milano, Italy, salvatore.borgia@polimi.it

Abstract

The multiphysics modeling has a great importance when a complex space system (as a satellite) is considered. Indeed, it is necessary to analyse how the system's behavior is affected by the space environment or by on board failures. In this paper, the *Modelica Library* is used to hierarchically build and connect the main subsystems that can be found in a traditional satellite. Specifically, the modeling and simulation of the entire system is carried out in the *Dymola*[1] environment. Finally, the FMI is applied to simulate in Dymola some specific satellite models/logics created with higher fidelity in the Matlab/Simulink[2] domain.

Keywords: Multiphysics modeling, Space system, Modelica library, Dymola, FMI tool

1 Introduction

A space system is generally composed of several subsystems belonging to different physical domains. A malicious entity could compromise even one of them to produce escalation effects involving the whole system. In this situation, the modeling task requires a holistic approach in order to simulate multiphysics interactions that occur inside the system. For this reason, the *Dymola* environment has been used to build a hybrid-complex system from the basic physical elements of the *Modelica library* exploiting the acausal modeling technique (Tiller 2001). Moreover, thanks to Modelica text coding and FMI (Functional Mock-up Interface) tool, ad hoc functions have been created to connect physical variables (simulating their mathematical relation), and to import from other tools (as *Matlab/Simulink*) the acausal model of system's subparts.

1.1 Satellite system

In this paper, a small satellite is considered as an example of space system. It is worth noting that each satellite has its own architecture and on board equipment depending on the specific mission to be accomplished. However, for the purpose of modeling the multyphysics interaction and creating a general simulation platform, a breakdown architecture (Figure 1) has been considered in order to catch the main physical domains characterizing almost any satellite: mechanical, thermal, fluid and electrical. Specifically, the considered subsystems are:

- ADCS (Attitude and Determination Control Subsystem): it allows to change/keep the satellite orientation in space through actuators (as reaction wheels (RWs)), or estimate it using sensors (as gyroscope (Gyro)).
- THR (Thermal subsystem): it allows to monitor the satellite instrumentation temperature or surface temperature keeping it within certain nominal bounds.
- PROP (Propulsion subsystem): it is responsible of delivering thrust to perform/support ADCS attitude maneuver or to counteract disturbance torques (as solar radiation pressure (SRP)).
- EPS (Electrical Power Subsystem): it allows generating power on board, using for example photovoltaic (PV) solar panel, and consequently the Power Management Unit (PMU) manages and distributes it to all subsystems (exploiting Maximum Power Point Tracking (MPPT), battery element, and DC-DC converter (Czarkowski 2011)).

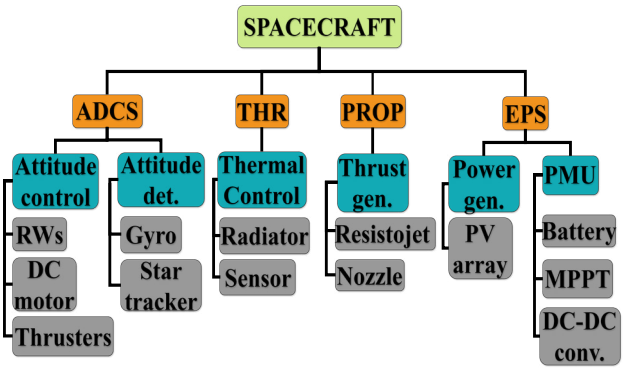


Figure 1. Satellite breakdown structure considered for Dymola modeling.

As it can be noticed in Figure 1: the subsystems of which the satellite is composed are at the first level (orange boxes), the subsystems tasks are reported at the second level (blue boxes), and the physical components (such

[1]https://www.3ds.com/products/catia/dymola

[2]https://it.mathworks.com/products/matlab.html

as actuators or sensors) to perform the tasks are depicted at the third level (grey boxes). The proposed architecture is more likely to be found in CubeSats (Song and al. 2018) due to the choice of resistojet as propulsion solution (Tummala and Dutta 2017). Apart from that, the structure can be then easily adaptable for other cases like large spacecraft. In the following sections, the modeling design of each subsystem and its principle of operation will be presented.

2 Modeling design

The main modeling steps of a general real system can be summarized as (Umez-Eronini-Eronini 1999):

1. Extract a physical model from reality: this process requires engineering judgment to isolate only the physical (state) variables which play a dominant role for the systems behaviour.

2. Extract the mathematical model from the physical one: at this stage, the identified physics phenomena shall be translated into mathematical expressions through the constitutive law equations of the specific physical component.

3. Simulate the mathematical model: the mathematical model of the system then has to be resolved trough the use of integrator scheme that returns the evolution of the state variables in time.

4. Perform sensitivity analysis: from the previous stage, it is possible then to compare the simulation results with the real response matching the behaviour of the digital world with the real one (state identification problem). This procedure allows closing the modeling loop and eventually obtaining a "digital twin" (Singh, Fuenmayor, and al. 2021).

In Figure 2, an example of the modeling process (above described) for the case of a typical electrical direct current (DC) motor which drives a shaft. It can be noticed how the initial complex real system is reduced to a physical representation governed by a set of simple first order differential equations (Cannon 1967).

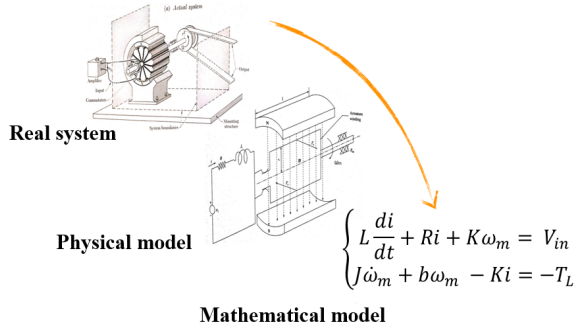


Figure 2. Modeling phases for a DC motor (Cannon 1967).

2.1 Satellite general architecture

After the definition of the high-level structure, the subsystems correlation flow scheme has to be derived to understand how the different physical variables interacts each other within the satellite system. Figure 3 reports the general satellite configuration considered for the modeling part and Figure 4 the relative actuators configuration. Without loss of generality, to simplify the simulation part, the PV arrays are rigidly connected with the main body and the satellite center of mass (CM) is assumed to be located at the origin of the Dymola world frame {**xyz**} where the attitude dynamics equations are expressed. Regarding the thrusters, for modeling simplification, they are assumed to be 6 and, according to which nozzle is activated, three-axis control torques are generated. Specifically, a water resistojet propulsion system is considered for this work. Regarding the star trackers, there are 2 looking at inertial fixed stars respectively along **x** and **y** direction. The satellite characteristic dimensions considered in terms of main body and actuators size, together with the main physical parameters of the system, are similar to the one of a 16U CubeSat (as it will be ramarked in section 5).

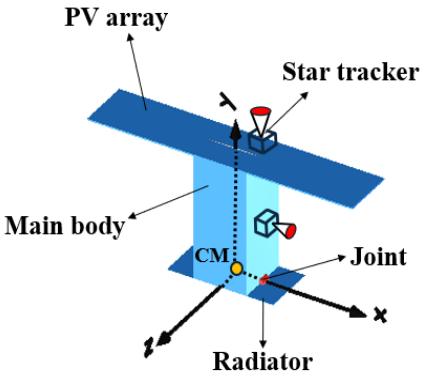


Figure 3. The satellite external configuration.

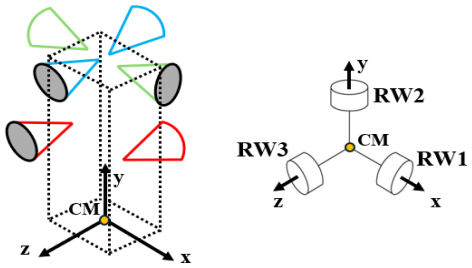


Figure 4. The satellite actuators configuration.

2.1.1 Satellite subsystems interaction

According to the satellite architecture and relative configuration previously presented, the subsystems mutual interaction is derived in Figure 5. From the diagram we can notice that:

 10.3384/ECP207155 DOI

- the EPS bus voltage $\mathbf{V}_{Bus}$, given by the DC-DC converter, or the battery voltage $\mathbf{V}_{Batt}$ feed RWs actuators of the ADCS and the resistojet of the PROP subsystem.

- The attitude matrix $\hat{\mathbf{R}}_{123}$, expressed as euler sequence "123", determines the relative orientation between Sun/Earth and Dymola body-frame. This affects which surfaces are receiving radiation heat or not as boolean vector ν_S. The latter vector is computed by the shadow model knowing the inertial Earth and Sun position ($\hat{\mathbf{r}}_{Earth}$ and $\hat{\mathbf{r}}_{Sun}$, respectively).

- The amount of power generated by the PV array is affected by the PV-Sun incidence angle $\hat{\alpha}_{SA}$.

- The steam obtained after a water mass flow rate $\dot{\mathbf{m}}_{H20}$ is heated up from the electro-thermal circuit of the resistojet, expands through the nozzle generating a level of thrust $\hat{\mathbf{F}}_p$ function of the steam pressure $\mathbf{P}_{st}$ and temperature $\mathbf{T}_{st}$.

- The fuzzy logic regulates the firing time and the thrust direction of each nozzle. In this case, the final output is the relative torque induced on the satellite $\hat{\mathbf{T}}_p$.

- The battery temperature $\mathbf{T}_{Batt}$ and PV array temperature $\mathbf{T}_{SA}$ are inputs for the EPS affecting the level of power generated on board and the battery charge-discharge profile.

- The SRP torque block is able to determine the disturbance torques $\hat{\mathbf{T}}_{srp}$ acting on the satellite knowing both the Sun inertial position and the satellite attitude.

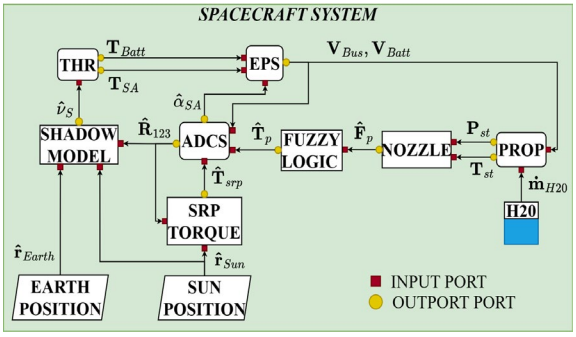


Figure 5. Satellite physical cross-interactions.

The subsystems interaction diagram just described is an example of basic working logic that can be found in many satellite (specifically in CubeSat). Again, each satellite can implement its own architecture slightly changing how the outputs of a subsystem affect the other ones. However, for the purpose of this work, the main multiphysics connections are considered and they shall be taken into account in the modeling part of the satellite physical components. Moreover, during the modeling process, some control logics have been assumed to link variables from a physical world to another. In this way, the space system simulated assumes the characteristics of a typical cyber-physical system.

2.1.2 ADCS subsystem

In this section, the ADCS subsystem is analysed more in details focusing on its working logic flow and on the corresponding modeling translation into Dymola. Figure 6 shows the specific subsystem logic, in particular from left to right we have:

1. The error between the desired satellite attitude angles $\hat{\alpha}^{(123)}_{target}$ (as Euler sequence "123") and the on bard estimation from sensors $\hat{\alpha}^{(123)}_{sat}$ is an input for the PID (Knospe 2006) control block.

2. The PID block returns the continuous signal voltage $\tilde{\mathbf{V}}_m$ to be supplied on the virtual DC motor in order to match the ideal control torque with the one generated by the virtual reaction wheel $\tilde{\mathbf{T}}_{RW}$.

3. According to the virtual RWs angular velocity ω_{RW}, another PID block calculates the duty cycle $\mathbf{D}_c$ of the H-bridge circuit to generate the needed square voltage signal $\mathbf{V}_m$ of amplitude given by the EPS bus voltage $\mathbf{V}_m$.

4. The square signal $\mathbf{V}_m$ supplies the real DC motor block. According to the motor angular rate ω_M, the real reaction wheel follows an angular velocity profile similar to the virtual one. The final effect is the torque released on satellite by the real RWs $\hat{\mathbf{T}}_{RW}$ due to the principle of "action-reaction".

5. Besides the RWs, the SRP torques and the PROP torques drive the attitude dynamics of the satellite assumed to be a simple rigid-body. This latter simplification avoids us to model flexibility which is dominant when large impulsive maneuvers occur on satellite having long solar arrays (Wei, Cao, and al. 2017). Indeed, for a small system, as the one considered in this paper (16U CubeSat), the rigid-body assumption makes more sense because of the short solar panel length and dominant main-body inertia.

As explained in section 2, the mathematical model has to be extracted from the physical world to simulate the behavior along the time. For the examined ADCS subsystem (Figure 6), the main governing equations are:

$$J\dot{\omega}_{sat} = -\omega_{sat} \times (J\omega_{sat}) + \hat{\mathbf{T}}_{srp} + \hat{\mathbf{T}}_p + \hat{\mathbf{T}}_{RW} \quad (1)$$

$$L_a \frac{di}{dt} = -R_a i - K_m \omega_M + \mathbf{V}_m \quad (2)$$

$$J_m \dot{\omega}_M = -b\omega_M + K_m i + \hat{\mathbf{T}}_{RW} \quad (3)$$

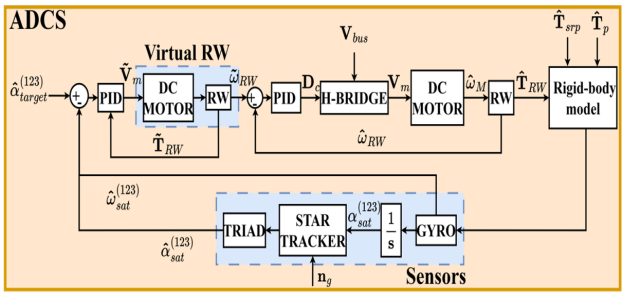


Figure 6. ADCS subsystem modeling scheme.

where in Equation 1: J is the satellite inertia matrix, and ω_{sat} is the angular velocity vector expressed in body-frame axes ({**xyz**} in Figure 3). In Equation 2: L_a and R_a are the inductor and the armature resistance of the DC motor circuit, respectively; i is the current flowing in the circuit; and K_m is the motor constant that allows the coupling of the electric domain with the mechanical one. In fact, in Equation 3, the torque produced by the motor ($K_m i$), together with friction losses ($b\omega_M$) and load torque given by the RW, determines the angular speed of the DC motor with inertia J_m. In this case, the motor shaft is directly connected with the load (RW) without gear-box in between ($\omega_M = \hat{\omega}_{RW}$). In Table 1 it is reported the *Modelica library* components used for modeling the ADCS subsystem in Dymola. The external satellite configuration (Figure 3) has been replicated with the basic rigid body element and then, through links, PV arrays have been added to the main body. The spherical joint is needed to simulate only the three degree of freedom of the satellite. The world frame in Dymola has been then connected to the spherical joint in correspondence of the satellite CM, and the gravity field option has been imposed to zero to replicate the deep space condition. The radiator mechanism is allowed to rotate only along the **z**-axis of Dymola frame thanks to the revolute joint element. Assuming an ideal gyro, it is simple an angular velocity sensor in Dymola. Regarding the DC motor speed controlled by the H-bridge electric circuit (Priyanka and Mariyammal 2018), the homonymous components already existing in the *Modelica library* have been exploited instead, for the RW, it has been modeled as inertia load element attached to the DC motor shaft. The Pulse Width Modulation (PWM) block in Modelica allows to generate the real voltage signal with discrete values: $[-\mathbf{V}_{Bus}, +\mathbf{V}_{Bus}]$ and a variable duty cycle given by a limited [0 to 1] PID. The fixed rotation/translation element, listed in the table, permits to link the main body with the PV arrays or, eventually, to arrange the satellite rigid body elements into another configuration.

In section 3, the Modelica text coding (within Dymola) will be discussed in order to simulate: the TRIAD algorithm, for estimating the attitude matrix exploiting the two measured vectors by star trackers, the Gaussian noise $\mathbf{n}_g$ model of the star tracker, and the SRP torque computation in Dymola body-fixed frame axes.

Table 1. Modelica structure of the ADCS subsystem

Physical element	*Modelica component*
Rigid-body	
World frame joint	
World torque	
Fixed rotation	
Radiators joint	
Voltage source	
DC motor	
RW	
H-Bridge	
PWM	
GYRO	
PID (limited)	

2.1.3 THR subsystem

The thermal model has been tackled using the lumped approach (Cannon 1967). According to this method, the satellite surfaces or internal instruments (like the battery) are modeled as nodes with a certain thermal capacity (depending on the material). The thermal nodes are capable to exchange heat each other, with the Sun/Earth, and with the deep space. In Figure 7 it is reported the lumped scheme of the satellite system presented in 2.1. In fact, the lumped nodes (3-4-5-6-7-8) represent the surfaces of the main body, the PV array surfaces correspond to the nodes (1-2-9-10) instead the radiators surfaces correspond to the nodes (11-12-13-14). As it can be noticed, the dominant heat exchange ways considered are conduction and radiation (the convection is not present or negligible in deep space due to lack of air). The deep space acts as a sink

10.3384/ECP207155 DOI

(average temperature of 3 [K]) with which each node exchanges heat by radiation. The PV array surfaces have been split in four nodes to model the different thermal properties between the top side (higher emissivity) and the bottom one (higher absorptivity). Similarly for the radiators with the difference of having higher emissivity at nodes (12-14) and higher absorptivity at nodes (11-13).

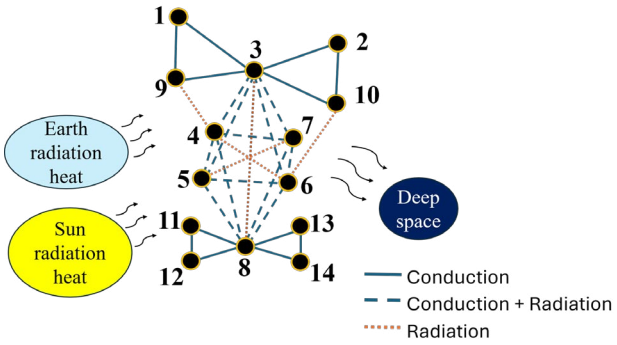


Figure 7. Satellite thermal lumped model.

The THR subsystem logic flow scheme is shown in Figure 8. Specifically, we can observe that:

1. The subsystem receives as external inputs: the satellite attitude matrix from ADCS, the sun position vector with respect to an inertial frame, and the solar irradiance $\mathbf{E}_{ir}$ with intensity depending on the satellite to Sun distance.

2. The shadow model returns the boolean vector ν_S with dimension equals to the number of thermal nodes (14 in this case). If the value is 1 means that the corresponding node is receiving the Sun or Earth radiation.

3. From the resulting satellite temperature distribution, hysteresis logic can be adopted to maintain the average nodes temperature between a desired range. The hysteresis logic will then open/close the radiators surface through the control torque $\hat{\mathbf{T}}_c$.

4. The opening of radiator will affect the heat transfer coefficients of the thermal model both for the conduction $\mathbf{G}_c$ and radiation $\mathbf{G}_r$.

5. Finally, from the radiator rotation angle $\hat{\alpha}_{radiator}$ along $\mathbf{z}$-axis of the body frame (Figure 3), the shadow condition for nodes (11-12-13-14) can be established.

Regarding the mathematical model, the main governing equations of the THR subsystem can be summarized as (Lienhard 2024):

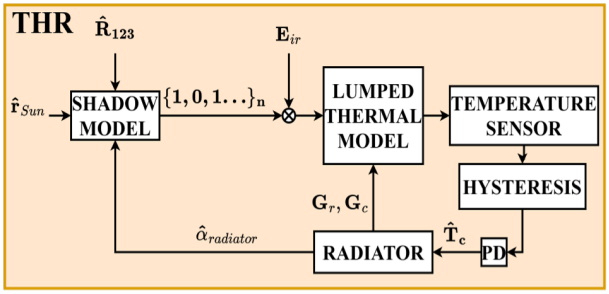


Figure 8. THR subsystem modeling scheme.

$$C_i \frac{dT_i}{dt} = \dot{Q}^i_{Sun} + \dot{Q}^i_{Earth} + \sum_{ij} \dot{Q}^{ij}_c + \sum_{ij} \dot{Q}^{ij}_r \tag{4}$$

$$\dot{Q}^i_{Sun} = \alpha_i \dot{q}_{Sun} A_i \tag{5}$$

$$\dot{Q}^i_{Earth} = \alpha_{al} \dot{q}_{al} A_i + \alpha_{ir} \dot{q}_{ir} A_i \tag{6}$$

$$\dot{Q}^{ij}_c = G^{ij}_c (T_i - T_j) \tag{7}$$

$$\dot{Q}^{ij}_r = G^{ij}_r (T_i - T_j) \tag{8}$$

$$G^{ij}_c = \frac{\lambda^{ij}_c A^{ij}_c}{s^{ij}_c} \tag{9}$$

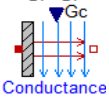
$$G^{ij}_r = \frac{\sigma_B (T_i + T_j)(T_i^2 + T_j^2)}{\frac{1-\varepsilon_i}{\varepsilon_i A_i} + \frac{1}{A_i F_{ij}} + \frac{1-\varepsilon_j}{\varepsilon_j A_j}} \tag{10}$$

where in Equation 4: T_i is the temperature of the $i-th$ node, C_i is the thermal mass capacity of the $i-th$ node, $\dot{Q}^i_{Sun}$ and $\dot{Q}^i_{Earth}$ are the Sun and Earth radiation energy falling into $i-th$ node, respectively. The Sun energy is computed through Equation 5 knowing the node absorptivity α_i, the Sun heat flux $\dot{q}_{Sun}$, and the node area A_i; instead the Earth energy is calculated using Equation 6 knowing the albedo absorptivity α_{al}, the infrared absorptivity α_{ir}, the albedo irradiance $\dot{q}_{al}$, and the infrared irradiance $\dot{q}_{ir}$ perceived on s/c. In Equation 4: $\dot{Q}^{ij}_c$ is the conduction energy exchange between the $i-th$ and $j-th$ node, and $\dot{Q}^{ij}_r$ is the radiation energy exchange between the $i-th$ and $j-th$ node. The conduction energy is governed by Equation 7 in which the thermal conductance G^{ij}_c has to be determined assuming a contact area A^{ij}_c between the nodes, a wall thickness s^{ij}_c, and the material thermal conductivity λ^{ij}_c (Equation 9). Regarding the radiation energy, it is calculated using Equation 8 knowing the radiation conductance G^{ij}_r. This latter term can be estimated through Equation 10 where: σ_B is the Boltzmann constant, $\varepsilon_{i,j}$ is the node emissivity, and F_{ij} is the radiative view factor between the $i-th$ and $j-th$ node (which can be obtained analytically applying the geometrical equations in (Martínez 2015)).

In Table 2, the Dymola structure of the just described THR subsystem is reported. Particularly, it can be noticed how the main physical variables of the mathematical model and logic in Figure 8 are acausally modeled by

a specific *Modelica library* component. In section 3, the shadow self-built function will be discussed in details.

Table 2. Modelica structure of the THR subsystem

Physical element	*Modelica component*
Thermal node	
Thermal conductor	
Radiation heat transfer	
Heat transfer coefficient	
Sun (Earth) radiation	
Hysteresis control	
Temperature sensor	
Deep space sink	

2.1.4 PROP subsystem

The PROP subsystem logic, for the considered satellite architecture in Figure 1, is shown in Figure 9. From left to right we have:

1. the target error, in terms of angle θ and angular rate error $\dot{\theta}$, are inputs for the fuzzy logic block which returns the firing time of each satellite nozzle (Figure 4).

2. The EPS bus voltage is decreased to a suitable value (~ 0.5 [V]) to be applied at the extremes of the tungsten rod. The voltage conversion is done thanks to a buck circuit (Czarkowski 2011). The final heat flow $\mathbf{Q}_{flow}$ released by tungsten, as result of Joule effect, feeds the boiler's furnace.

3. Inside the vaporizer, the liquid water flow $\dot{\mathbf{m}}_{H2O}$ turns into vapour phase expanding towards the convergent-divergent nozzle. The amount of liquid phase in the boiler is controlled by a PID that maintains the furnace on.

4. From the nozzle moment arm $\mathbf{L}_{arm}$ with respect to CM, the propulsive torques $\hat{\mathbf{T}}prop$, due to the exhausted steam mass flow, are calculated.

5. The valve element allows the steam generation and consequently the thrust delivery on satellite. It can be opened whenever an attitude maneuver with the PROP subsystem shall be performed.

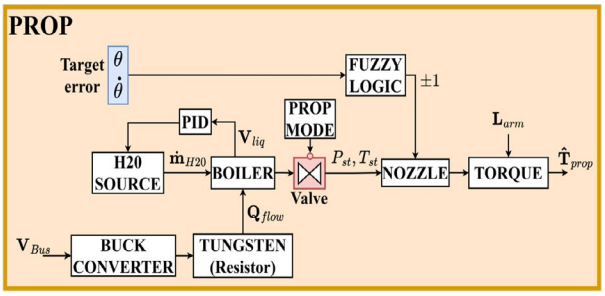


Figure 9. PROP subsystem modeling scheme.

In this case, the boiler is an in-built acausal Modelica component which implements the drum-boiler dynamics (Åström and Bell 2000). The other main equations of the mathematical model involve the nozzle, and the fuzzy logic block. The first block computes the thrust according to the rocket equations (Sutton and Biblarz 2017) (assuming the steam as an ideal gas):

$$\hat{\mathbf{F}}_p = \dot{m}_{st} V_e + p_e A_e \tag{11}$$

$$V_e = M_e \sqrt{\gamma R T_e} \tag{12}$$

$$\frac{p_e}{P_{st}} = \left(1 + \frac{\gamma - 1}{2} M_e^2\right)^{-1} \tag{13}$$

$$\frac{T_e}{T_{st}} = \left(1 + \frac{\gamma - 1}{2} M_e^2\right)^{-\frac{\gamma}{\gamma - 1}} \tag{14}$$

$$\frac{A_e}{A^*} = \left(\frac{\gamma + 1}{2}\right)^{-\frac{\gamma+1}{2(\gamma-1)}} \left[\frac{\left(1 + \frac{\gamma-1}{2} M_e^2\right)^{\frac{\gamma+1}{2(\gamma-1)}}}{M_e}\right] \tag{15}$$

where in Equation 11: $\dot{m}_{st}$ is the steam mass flow rate produced in the boiler, V_e is the steam exhaust velocity, and p_e is the pressure at the exit area A_e of the nozzle. Equation 12 allows to estimate the exhaust velocity from the exit Mach number M_e, the heat capacity ratio γ, the steam gas constant R, and the temperature condition T_e at exit nozzle area. The Mach number can be iteratively calculated, applying for example the Newton method (Galántai 2000), once the ratio between the nozzle throat area A^* and the exit one is fixed (Equation 15). The missing quantities of steam pressure and temperature conditions at the exit can be determined respectively with Equation 13 and Equation 14. The fuzzy logic has been designed using the Mamdani GUI interface in Matlab (as it will be explained in section 4). For the fuzzy rules, they have been chosen following the procedure in (Nagi, Ahmed, and al. 2009). Figure 10 shows the membership functions considered for the design of the fuzzy bang-bang controller.

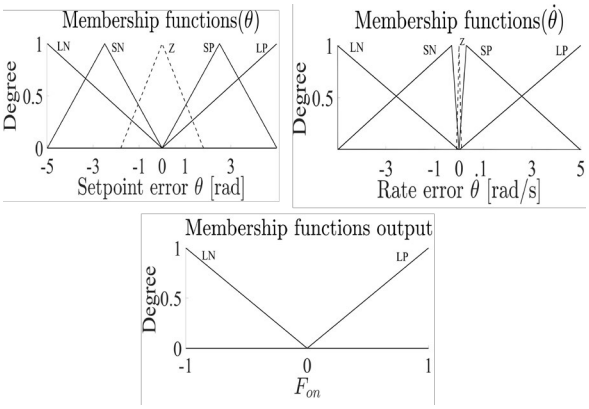


Figure 10. Fuzzy memebership functions used in Mamdani model scheme.

In Table 3, the PROP acausal version in Dymola is shown. Notice how the resistor component has the thermal port enabled to simulate the dissipation heat $\mathbf{Q}_{flow}$ released by the tungsten mass. Moreover, the heat transfer component is added to restore the nominal boiler conditions (tungsten temperature ~ 298 [K]) just after the PROP is turned off.

Finally, the nozzle I/O (Input/Output) block in Dymola will be presented in section 3 where the Modelica text coding has been applied to connect this element with the others Modelica PROP components listed in Table 3.

2.1.5 EPS subsysyem

The EPS subsystem is now analysed to understand the logic architecture assumed for this work. In Figure 11 it is reported the subsystem operation:

1. The external inputs of: the Sun heat flux over the PV array, the Sun rays incidence angle on the PV surface, and the solar panel temperature are used to estimate the photocurrent flowing in the PV array circuit.

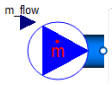
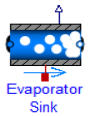
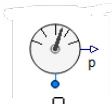
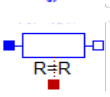
2. The MPPT algorithm regulates the PV voltage to operate at the maximum power possible. Then, the PV voltage is increased or decreased, using a two-switch buck-boost converter (Kim and al. 2022), to obtain the operative bus voltage (for 16U CubeSat assumed to be ~ 13 [V]).

3. The battery is fed nominally on bus voltage and, according to its state-of-charge (SOC) or eclipse condition ($V_{Bus} \sim 0$ [V]), the BMS (Battery Management System) unit regulates the current i_{Batt} flowing in it.

4. When an eclipse occurs or, in general, the bus voltage is too low, the switch is closed such that the battery can supply power to the loads.

5. The charge-discharge battery profile is affected by the temperature of the node associated to the battery T_{Batt}.

Table 3. Modelica structure of the PROP subsystem

Physical element	*Modelica component*
Water source	
Steam drum boiler	
Steam sink boundary	
Valve	
Steam pressure sensor	
Steam temperature sensor	
Voltage source	
Thermal resistor	
Tungsten mass	
Heat transfer	

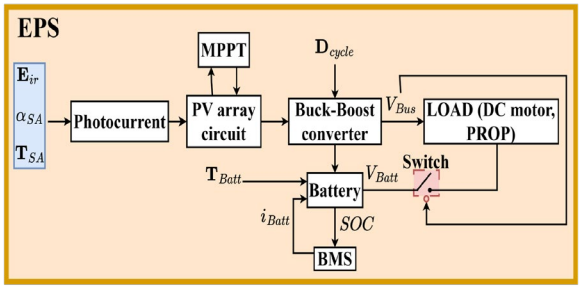


Figure 11. EPS subsystem modeling scheme.

Regarding the MPPT algorithm, Figure 12 shows the logic scheme of the applied method to make PV array operate at the maximum extractable power for different photocurrent values. Instead, the BMS applies the typical Constant Current-Constant Voltage (CC-CV) algorithm (Mostacciuolo, Iannelli, and al. 2018) during the battery charging-discharging process.

In Figure 13, the Buck-Boost converter physical model, considered for the Dymola simulation, is reported. It is worth noticing that, in the context of acausal modeling approach, only the physical representation of the real system

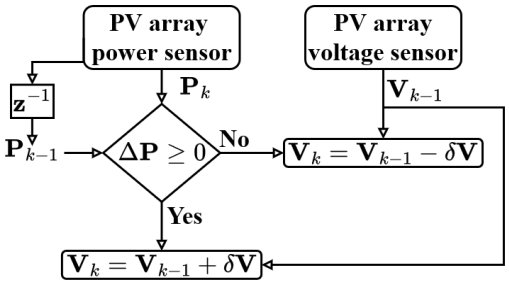


Figure 12. MPPT algorithm scheme.

has to be extracted, while the constitutive equations are already embedded in each singular component of the *Modelica library*. In this way, the modeling focus is moved more on the physical concept rather than on the solving procedure of the equations (Kulhánek and al. 2015). The mathematical model of the EPS subsystem can be summarized by the following main constitutive equations:

$$\mathbf{I}_{diode} = \mathbf{I}_{ds}\left[\exp\left(\frac{v_d}{N v_t}\right) - 1\right] \tag{16}$$

$$v_t = \frac{\sigma_B T_d}{q} \tag{17}$$

$$\mathbf{V}_{out} = \frac{d_1}{1 - d_2}\mathbf{V}_{in} \tag{18}$$

where in Equation 16: $\mathbf{I}_{diode}$ is the current flowing into a single diode, $\mathbf{I}_{ds}$ is the diode saturation current, v_d is the diode voltage drop, N is the diode emission coefficient, and v_t is the diode thermal voltage. This latter quantity can be retrieved using Equation 17 knowing the Boltzman constant, the diode (or solar array) temperature T_d, and the electron charge q.

In Equation 18, the voltage conversion of the two switch buck-boost converter as function of: the duty cycles d_1 and d_2 of the two switches (S_1 and S_2, respectively), and the input circuit voltage $\mathbf{V}_{in}$.

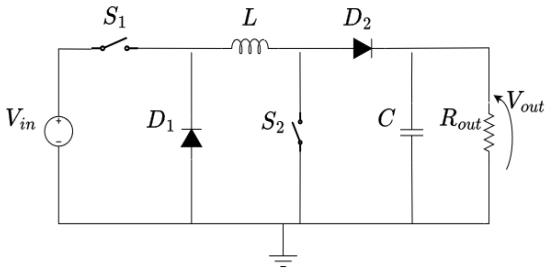
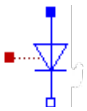
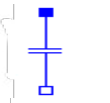


Figure 13. Buck-Boost converter electric circuit.

As it has been done with the previous satellite subsystems, the Dymola EPS translation is listed in Table 4. Specifically, the series/parallel connection of a singular thermal diode element in Modelica allows to build the entire PV array circuit with temperature as inputs. For the MPPT logic, the `Logical` library has been exploited. Regarding the BMS unit, in section 4 it will be discussed the modeling exportation from the Matlab/Simulink to Dymola environment using the *FMI* interface.

Table 4. Modelica structure of the EPS subsystem

Physical element	*Modelica component*
PV array diode	
Photo-current	
Resistor	
Capacitor	
Inductor	
Switch (BB)	
Diode (BB)	
Power sensor	
Ground	

3 Modelica coding

In this section, the generation method of the ad hoc functions presented so far is discussed. For the analyzed satellite system, they can be summarized in:

- SRP torque
- ADCS sensor
- Shadow model
- Nozzle
- Photocurrent

These functions have dual objectives: modeling all that specific algorithms or components to which a corresponding acausal translation can not be found in the *Modelica Standard Library*, and connecting a s/c subsystem to another or to the space environment (like SRP torque function). The functions generation has been performed using

the `Text` option in Dymola starting from the basic layout in the `Blocks` library (*Modelica® - A Unified Object-Oriented Language for Systems Modeling* 2014). This option allows to design the block (I/O) ports and to write inside of it the algorithm which relates the input with the output. Specifically, the main algorithms/models used are:

1. The SRP torque generated by each satellite surface, with respect to CM (Figure 3), depending on its absorption, diffuse reflection, and specular reflection coefficients (Wertz 1978).
2. The perturbing matrix as Euler sequence "123", with a rotation angle function of the gaussian noise $\mathbf{n}_g$, to determine the vectors measured by the star trackers in body-frame axes.
3. The TRIAD algorithm (Markley 1999) to estimate the s/c attitude angles as euler sequence "123" (alphaX_est, alphaY_est, and alphaZ_est).
4. The determination of the shadow condition for each lumped node (Figure 7) checking the value of the scalar product between the Sun direction $\hat{r}_{Sun}$ and the normal of each s/c surface.
5. The converging-diverging isentropic flow expansion (Sutton and Biblarz 2017).
6. The PV photocurrent mathematical model (X. H. Nguyen and M. P. Nguyen 2015).

Figure 14 shows the corresponding Dymola layout of the functions above described (highlighting the variables discussed in the previous sections). In Listing 1 it is reported an example of the Modelica text structure for the *Photocurrent block*.

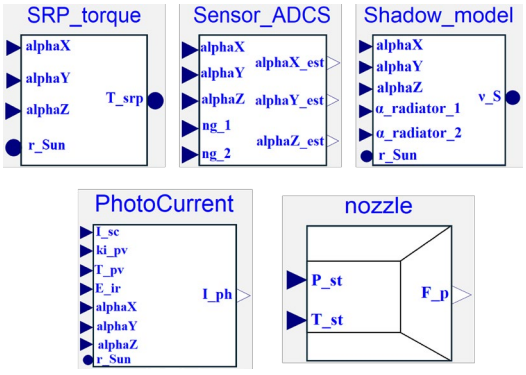


Figure 14. Dymola block functions architecture.

Listing 1. Modelica text of Photocurrent block function

```
block PhotoCurrent
  extends Modelica.Blocks.Icons.Block;
  Modelica.Blocks.Interfaces.RealInput I_sc
      ;
  Modelica.Blocks.Interfaces.RealInput
      ki_pv;
  Modelica.Blocks.Interfaces.RealInput T_SA
      ;
  Modelica.Blocks.Interfaces.RealInput E_ir
      ;
  Modelica.Blocks.Interfaces.
      RealVectorInput y_body[3];
  Modelica.Blocks.Interfaces.
      RealVectorInput r_Sun[3];
  Modelica.Blocks.Interfaces.RealOutput
      I_ph;
protected
    Real cosine_alpha_SA;
    Real nu_el;
    Real cosine_alpha_SA;
    Real F_sensitivity;
    Real I_ir_SA;

algorithm

cosine_alpha_SA := y_body[1]*r_Sun[1] +
    y_body[2]*r_Sun[2] + y_body[3]*r_Sun
    [3];

if cosine_alpha_SA < 0 then
  cosine_alpha_SA :=0;
end if;

I_ir_SA :=E_ir*nu_el*F_sensitivity*
    cosine_alpha_SA;

I_ph :=(I_sc + ki_pv*(T_SA - 298.15))*(
    I_ir_SA/1000);
end PhotoCurrent;
```

4 FMI interface

The Functional Mock-up Interface (FMI) is a powerful tool when the modeling of complex hybrid system (as a space system) is requested. Indeed, this interface allows: the interaction between different programming language (making the model more versatile), and the use of the potentialities coming from each modeling platforms. For this work, the FMI has been used for:

- The Fuzzy bang-bang logic (Figure 15).
- The Battery-BMS module (Figure 16).

Each of them has been acausally modeled exploiting the `Simscape library` within the Matlab/Simulink environment. The choice of changing the modeling language has been made for two reasons: speeding up the implementation using the fuzzy logic designer app in Matlab/Simulink, and taking advantage of the medium-high fidelity model of the battery element in *Simscape* (especially for the temperature dependency effects).

5 Simulation results

Once the hierarchical model of the satellite system is done, with the relative differential equations implicitly defined for each physical component, the next step is to simulate the system (as described in section 2). The tab

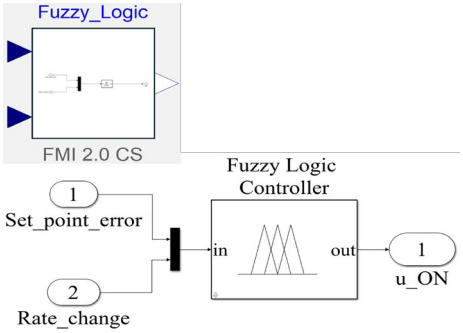


Figure 15. The Dymola FMI block (top) and the relative Simulink model (bottom) of the Fuzzy logic.

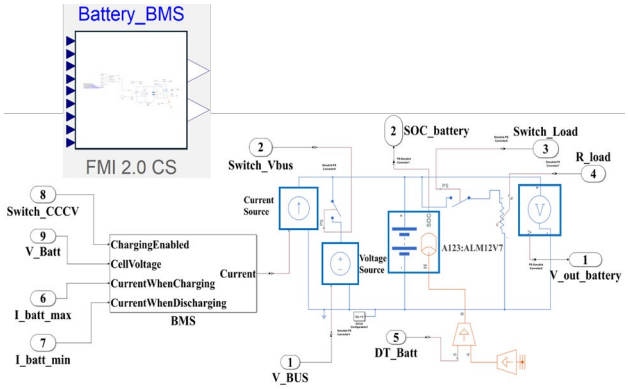


Figure 16. The Dymola FMI block (top) and the relative Simulink model (bottom) of the Battery-BMS module.

`Simulation` in Dymola allows to integrate the entire satellite mathematical model selecting a proper integration scheme. Below are the main assumptions and parameters selected to perform the simulation:

- The satellite is a 16U CubeSat with dimension: $0.2 \times 0.2 \times 0.4$ [m].
- The PV array dimension are: $0.2 \times 0.4 \times 0.01$ [m].
- The DC motor nominal voltage is ~ 12 [V].
- The RWs inertia is ~ 0.0005 [kg m^2].
- The peak power generated on board is ~ 180 [W].
- The maximum battery voltage is ~ 15 [V].
- The star tracker covariance is $1\sigma \sim 10^{-4}$ [rad].
- The Mach number M_e at each nozzle exit area is ~ 10.
- The satellite is in a heliocentric orbit receiving a constant Sun heat flux of 1370 [W/m^2].
- Only the Sun is considered as heat source for the thermal model (no Earth contribution).
- The simulation time is ~ 100 [s] in which the satellite displacement is neglected considering the inertial Sun direction fixed.
- No orbit propagation but only satellite attitude evolution in time.
- The aluminium is assumed as the satellite main body material and silicon one for the PV surfaces.
- The wall thickness considered for the thermal conduction is ~ 0.01 [m].

In the following sections, the results of a singular satellite subsystem and a scenario embedding more subsystems will be analysed. Due to the high number of physical variables involved, in section 6 it has been reported the satellite animation (obtained in Dymola) together with the evolution of the main subsystems variables. Regarding the integrator, to solve the mathematical model, the `Dassl` scheme has been used for all the simulations except for the last one (section 5) where the stiff scheme `Sdirk34hw` has been proved to be more performing.

5.1 ADCS-Sensors simulation

The ADCS subsystem is now simulated in Dymola according to the scheme presented in Figure 6. The simulation assumes to have the Sun direction fixed at $[0,\ 0,\ 1]$ and the target ramp profiles shown in Figure 23**b** to be tracked by the satellite (or spacecraft (s/c)). In Figure 17 it can be visualized the comparison between the ideal gyro measurement integration and the TRIAD algorithm output along **x**-axis body frame. Obviously, the estimated attitude (black) differs from the real one (red) unless of the Gaussian noise $\mathbf{n}_g$. The peak overshoot attitude error (Figure 23**e**) obtained during the maneuver is $\sim 0.34°$ (which can be modified tuning the PID parameters). The real voltage square signal given to the DC motor of the RW1 is depicted in Figure 23**f**. In this simulation, the PWM block switching frequency is set at 1000 [Hz]. In Figure 23**a** the satellite attitude animation, during the maneuver, is extracted thanks to the `Animation` tool in Dymola.

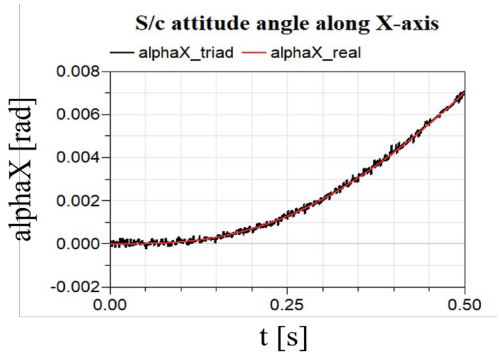


Figure 17. Satellite attitude angle comparison between the real and the estimated one by the *Sensor_ADCS* block.

10.3384/ECP207155 DOI

5.2 Fuzzy logic (FMI simulation)

The PROP architecture described in Figure 9 is now tested in Dymola. As we noticed from the previous ADCS simulation (Figure 17), the TRIAD estimation doesn't affect the PID control which is able to bring anyway the satellite at the desired attitude. So, from now on the ideal attitude angles from gyro will be considered in place of star trackers measurements. Also in this case, the Sun direction is fixed at $[0,\ 0,\ 1]$ and the final desired attitude sequence "123" is set to $\hat{\alpha}_{target} = [0.3,\ 0.8,\ 0.1]$ [rad]. Figure 18 shows the simulation results. It can be noticed how the PROP system is able to maneuver the satellite reaching a final attitude error (Figure 18**d**) of $\sim 0.06°$.

The Fuzzy FMI block outputs are reported in Figure 19 in terms of propulsive torques, expressed in Dymola *body fixed-frame*, and the corresponding firing time/direction associated to each satellite nozzle (Figure 4).

As expected from the exit vaporizer conditions in Figure 20, the resistojet gives a torque intensity in the order of 10 [mNm] with a nominal steam pressure $P_{st} = 13$ [bar] and a water mass flow rate, to maintain the vaporizer on, of $\dot{\mathbf{m}}_{H2O} \sim 0.13$ [kg/s]. It is worth highlighting that the mass of water storable inside the satellite is constrained by the dimensions of the system itself so, according to the mission design, the PROP subsystem parameters have to be refined accordingly.

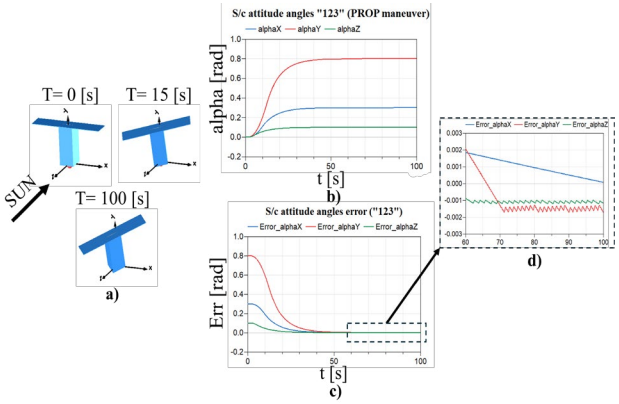


Figure 18. Fuzzy FMI block outcome: **a)** Satellite attitude animation during the propulsive maneuver; **b)** Satellite attitude Euler angles $\hat{\alpha}_{sat}^{(123)}$; **c)** Error profile between the target attitude angles and the real ones; **d)** Zoom in of the attitude error in the time interval [60-100] [s].

5.3 Battery-BMS (FMI simulation)

In this section, the Battery-BMS FMI block presented in section 4 is simulated separately to visualize its behavior in time. For this simulation, the simulink model has been augmented of a load resistance of 0.1 [Ω] to replicate the power absorption event when the battery feeds the loads. Regarding the satellite bus voltage signal, it has been imposed to be squared to simulate periodical eclipse effects. Figure 21 shows the overall response of the battery. Specifically, when $\mathbf{V}_{Bus}$ is zero, the load switch is activated with the effect of a drop battery voltage and consequently discharge. Then, a sensitivity analysis has been done for three different battery temperature: 283 [K], 293 [K], and 313 [K]. In Figure 22**a** it can be noticed how the higher battery temperature accelerated the discharge time (red SOC curve) against the case of a lower temperature (green SOC curve). For the battery voltage instead, it can be caught the effects of an increment of the battery charging voltage and higher loaded drop voltage at lower temperature (Figure 22**b**).

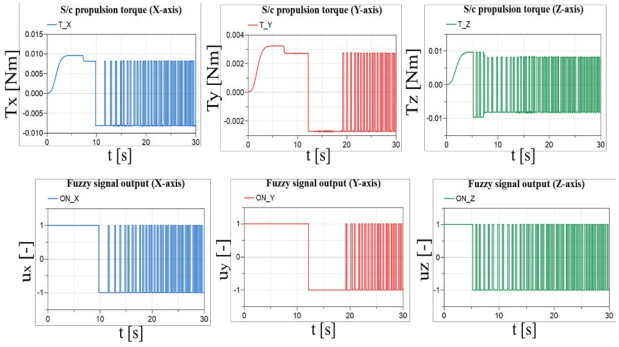


Figure 19. Fuzzy logic FMI outputs: at the top the propulsive torques profile $\hat{\mathbf{T}}_p$ in the Dymola *body fixed-frame*, and at the bottom the firing time/direction for each nozzle in the time interval [0-30] [s].

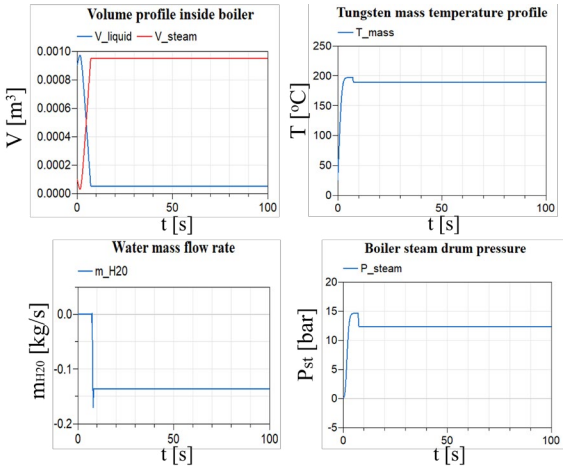


Figure 20. Vaporizer state variables evolution with time during the PROP maneuver.

5.4 THR-Radiator simulation

The THR subsystem, with the radiators active control architecture shown in Figure 8, is simulated here in Dymola considering: the inertial Sun direction at $[0,\ -1,\ 0]$ and then, after ~ 6.9 [h], it changes at $[0,\ 0,\ 1]$; the control is based on checking the temperature of node 8 and trying to maintain it below 70 [°C]. Moreover, when the radiators are closed, they don't exchange heat at all keeping constant their temperature. The simulation results are re-

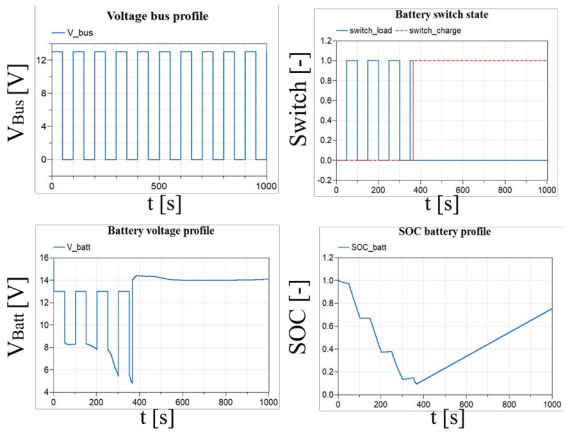


Figure 21. Battery-BMS simulation outputs.

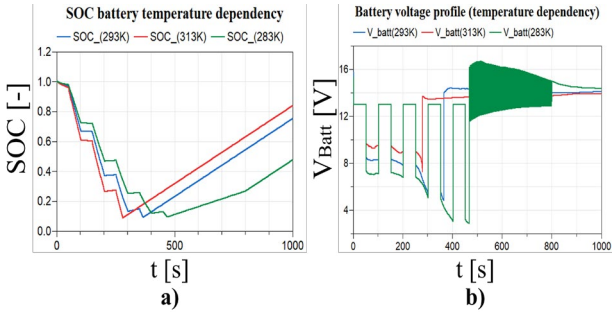


Figure 22. Battery response for different temperature conditions.

ported in Figure 24. Specifically, it can be noticed how the radiators are opened when T_8 reaches 60 [°C](Figure 24**c**), and they are closed again when T_8 goes below 40 [°C] to avoid overcooling of the system. It is worth remarking that due to the small satellite dimensions (16U CubeSat), the symmetry of the thermal scheme (Figure 7), and the homogeneity of the material used, the temperature difference among the nodes is small (as it can be seen in Figure 24**d** and in Figure 24**g**). Finally, Figure 24**e** shows the increment of the heat radiation exchange to deep space when high emissivity nodes (8-12-14) are active.

5.5 Multyphysics scenario

The last simulation involves all the satellite subsystems except for PROP (the attitude maneuver is performed only by RWs). The analysed multiphysics scenario is the following: the satellite (or s/c) shall perform a rotation of 180° around the fixed Dymola **x**-axis; the Sun direction is fixed at $[0,\ 1,\ 0]$. In Figure 25 the overall system response is reported. In particular, it can be noticed how the power generation is higher at the beginning of the maneuver (Figure 25**b**) and then zero when the nodes (1-2) are in shadow (Figure 26**a**). In this latter case, the battery feeds the RWs by discharging (Figure 25**d**) and delivering them a voltage $\sim$ 13 [V] (Figure 25**g**). For this simulation, the PWM signal frequency has been set to 100 [Hz].

In Figure 26**b**, the MPPT logics allows to make the main bus operate at the nominal value $\sim$ 13 [V] despite the PV array is working at higher voltage. Finally, Figure 26**c** shows the real-time satellite attitude angle trajectory controlled by the PID, and the corresponding angular velocity of the RW1 in Figure 26**d**.

6 Conclusions

The main objectives of this work were: modeling and simulating an example of complex space system using the *Modelica* tools; creating a platform where different tests and failure analysis could be carried out. The future work will be to expand the satellite model including also the structure subsystem to study the flexibility effects (especially for large satellites). Lastly, the FMI option will help on exporting the models into others programs to perform parametric system identification (Gupta and al. 2018) or nonlinear system identification in frequency domain (Pintelon and Schoukens 2012), using the real system telemetry, and to obtain higher fidelity models.

References

Åström, K.J. and R.D. Bell (2000). "Drum-boiler dynamics". In: *Automatica* 36.3, pp. 363–378. DOI: 10.1016/S0005-1098(99)00171-5.

Cannon, Robert H. (1967). *Dynamics of physical systems*. McGraw-Hill. ISBN: 0486428656.

Czarkowski, Dariusz (2011). *Power Electronics Handbook (Third Edition)*. Butterworth-Heinemann, pp. 249–263. DOI: 10.1016/B978-0-12-382036-5.00013-6.

Galántai, A. (2000). "The theory of Newton's method". In: *Journal of Computational and Applied Mathematics* 124.1-2, pp. 25–44. DOI: 10.1016/S0377-0427(00)00435-0.

Gupta, Sapna and et al. (2018). "Parametric system identification and robust controller design for liquid–liquid heat exchanger system". In: *IET Control Theory & Applications* 12.10, pp. 1474–1482. DOI: 10.1049/iet-cta.2017.1128.

Kim, Sunghwan and et al. (2022). "Modified Design of Two-Switch Buck-Boost Converter to Improve Power Efficiency Using Fewer Conduction Components". In: *Applied Sciences* 13.1, p. 343. DOI: 10.3390/app13010343.

Knospe, C. (2006). "PID control". In: *IEEE Control Systems Magazine* 26.1, pp. 30–31. DOI: 10.1109/MCS.2006.1580151.

Kulhánek, Tomáš and et al. (2015). "Experiences in teaching of modeling and simulation with emphasize on equation-based and acausal modeling techniques". In: *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 3683–3686. DOI: 10.1109/EMBC.2015.7319192.

Lienhard, John (2024). *A Heat Transfer Textbook*. 6th ed. Phlogiston Press.

Markley, F. Landis (1999). *Attitude Determination Using Two Vector Measurements*. URL: https://www.researchgate.net/publication/4706531_Attitude_Determination_Using_Two_Vector_Measurements/link/0c960525b7927e11c5000000/download.

Martínez, Isidoro (2015). *Radiative view factors*. URL: http://imartinez.etsiae.upm.es/~isidoro/tc3/Radiation%20View%20factors.pdf.

Modelica® - A Unified Object-Oriented Language for Systems Modeling (2014). URL: https://modelica.org/documents/ModelicaSpec33Revision1.pdf.

Mostacciuolo, E., L. Iannelli, and et al. (2018). "Modeling and power management of a LEO small satellite electrical power system". In: *2018 European Control Conference (ECC)*, pp. 2738–2743. DOI: 10.23919/ECC.2018.8550095.

Nagi, Farrukh, S.K. Ahmed, and et al. (2009). "Fuzzy bang-bang relay controller for satellite attitude control system". In: *Fuzzy Sets and Systems* 161.15, pp. 2104–2125. DOI: 10.1016/j.fss.2009.12.004.

Nguyen, Xuan Hieu and Minh Phuong Nguyen (2015). "Mathematical modeling of photovoltaic cell/module/arrays with tags in Matlab/Simulink". In: *Environmental Systems Research* 4.24. DOI: 10.1186/s40068-015-0047-9.

Pintelon, Rick and Johan Schoukens (2012). *System Identification: A Frequency Domain Approach*. 2nd ed. John Wiley & Sons Inc.

Priyanka, K. and A. Mariyammal (2018). "DC Motor Speed Control Using PWM". In: *International Journal of Innovative Science and Research Technology* 3.2.

Singh, Maulshree, Evert Fuenmayor, and et al. (2021). "Digital Twin: Origin to Future". In: *Applied System Innovation* 4.2, p. 36. DOI: 10.3390/asi4020036.

Song, Sua and et al. (2018). "Design and Implementation of 3U CubeSat Platform Architecture". In: *International Journal of Aerospace Engineering* 2018. DOI: 10.1155/2018/2079219.

Sutton, George P. and Oscar Biblarz (2017). *Rocket Propulsion Elements*. John Wiley & Sons Inc. ISBN: 1118753658.

Tiller, Michael (2001). *Introduction to Physical Modeling with Modelica*. Vol. 615. Springer Science & Business Media.

Tummala, Akshay Reddy and Atri Dutta (2017). "An Overview of Cube-Satellite Propulsion Technologies and Trends". In: *Aerospace* 4.4, p. 58. DOI: 10.3390/aerospace4040058.

Umez-Eronini-Eronini (1999). *System dynamics and control*. Pacific Grove : PWS Publishing Company. ISBN: 0534944515.

Wei, Jin, Dengqing Cao, and et al. (2017). "Dynamic modeling and simulation for flexible spacecraft with flexible jointed solar panels". In: *International Journal of Mechanical Sciences* 130, pp. 558–570. DOI: 10.1016/j.ijmecsci.2017.06.037.

Wertz, James R. (1978). *Spacecraft Attitude Determination and Control*. D. Reidel. ISBN: 9027712042.

Appendix: Simulation Graphs

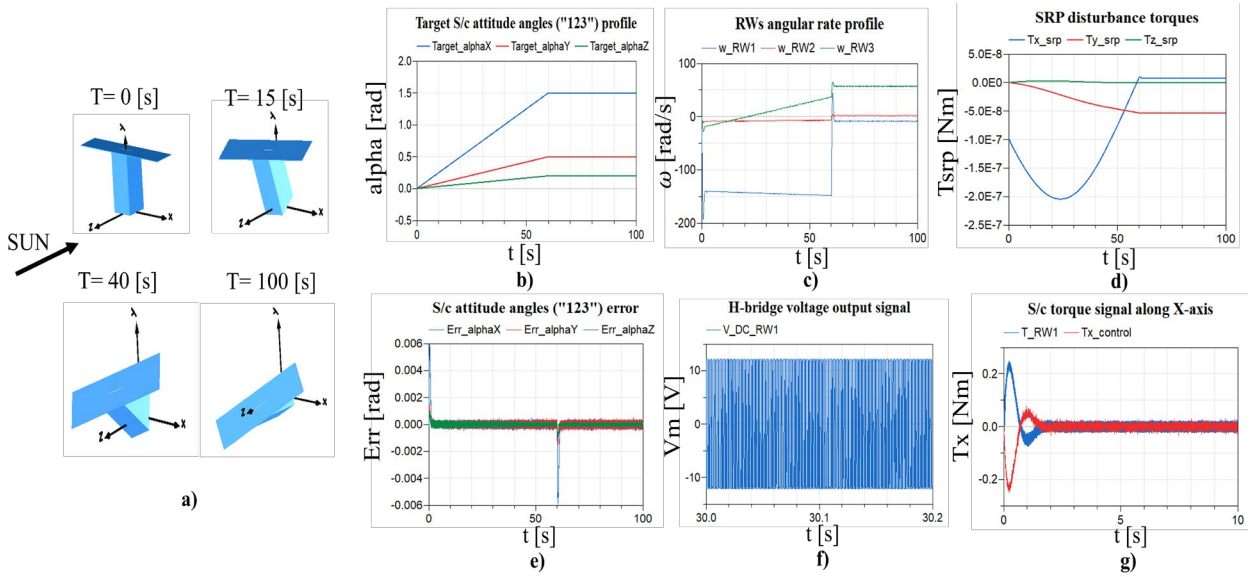


Figure 23. ADCS subsystem simulation outcome: **a)** Satellite attitude animation at four different times; **b)** Target angles profile $\hat{\alpha}_{target}^{(123)}$; **c)** RWs angular rate profile ω_M; **d)** SRP torques profile in the Dymola inertial frame {**xyz**}; **e)** Error profile between the desired attitude and the estimated one with TRIAD algorithm; **f)** The real voltage profile $\mathbf{V}_m$ applied to the DC motor of the RW1 in the time interval [30 - 30.2] [s]; **g)** Control torque profile from PID (red) and RW1 torque released along *body* **x**-axis (blue) in the time interval [0 - 10] [s].

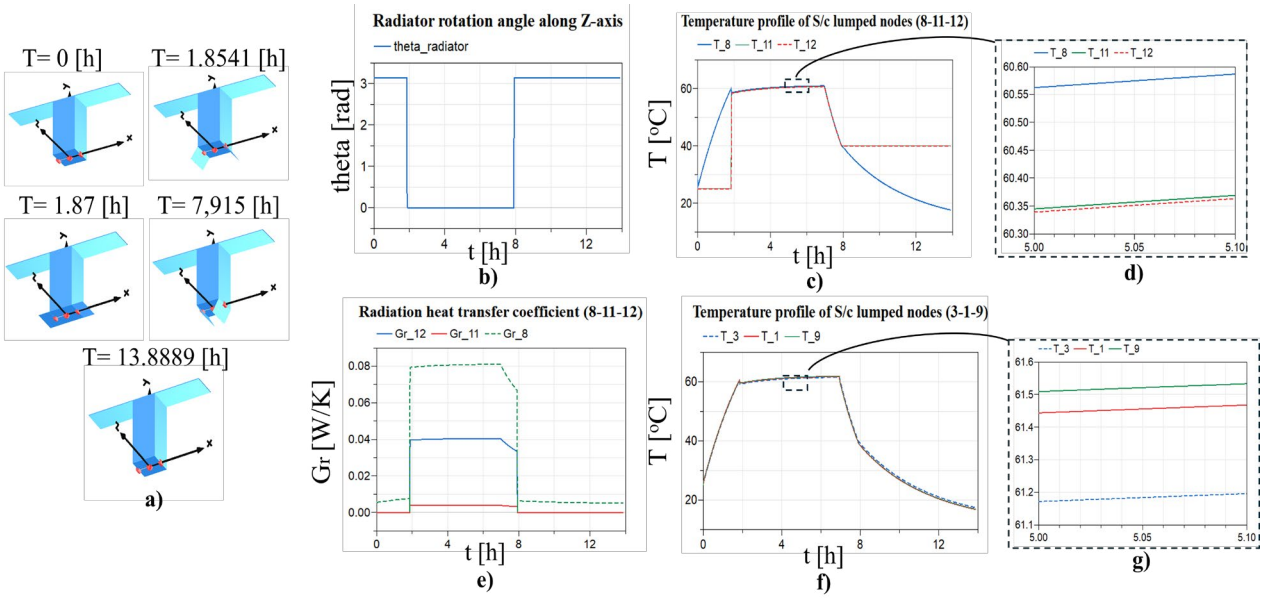


Figure 24. THR subsystem simulation outcome: **a)** Satellite radiators opening/closing animation; **b)** The evolution of the radiator angle $\hat{\alpha}_{radiator}$ with time; **c)** The lumped temperature profile of nodes (8-11-12) as shown in Figure 7; **d)** Zoom in of the temperature difference between nodes (8-11-12) in the time interval [5 - 5.1] [h]; **e)** The radiation heat transfer coefficient variation towards deep space for nodes (8-11-12); **f)** The lumped temperature profile of nodes (1-3-9) as shown in Figure 7; **g)** Zoom in of the temperature difference between nodes (1-3-9) in the time interval [5 - 5.1] [h].

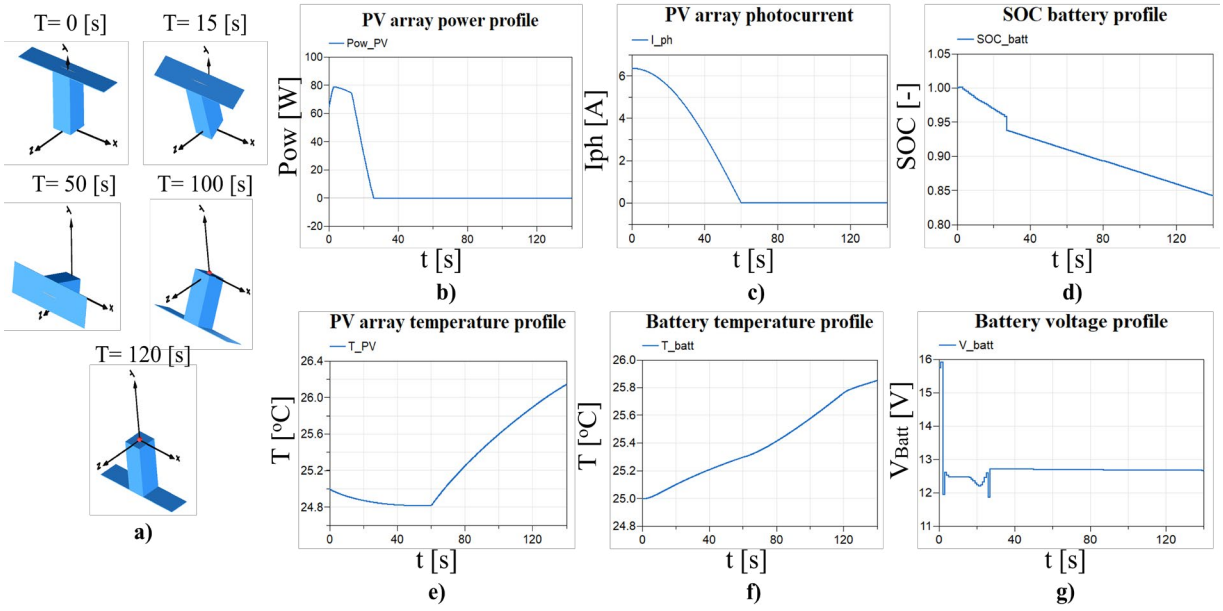


Figure 25. Satellite multiphysics simulation outcome: **a)** Satellite animation during the attitude maneuver; **b)** Power generated by a singular PV array surface; **c)** Photocurrent flowing into a singular PV circuit with time; **d)** SOC battery evolution with time; **e)** PV array temperature evolution with time; **f)** Battery temperature profile with time; **g)** Battery voltage response with time.

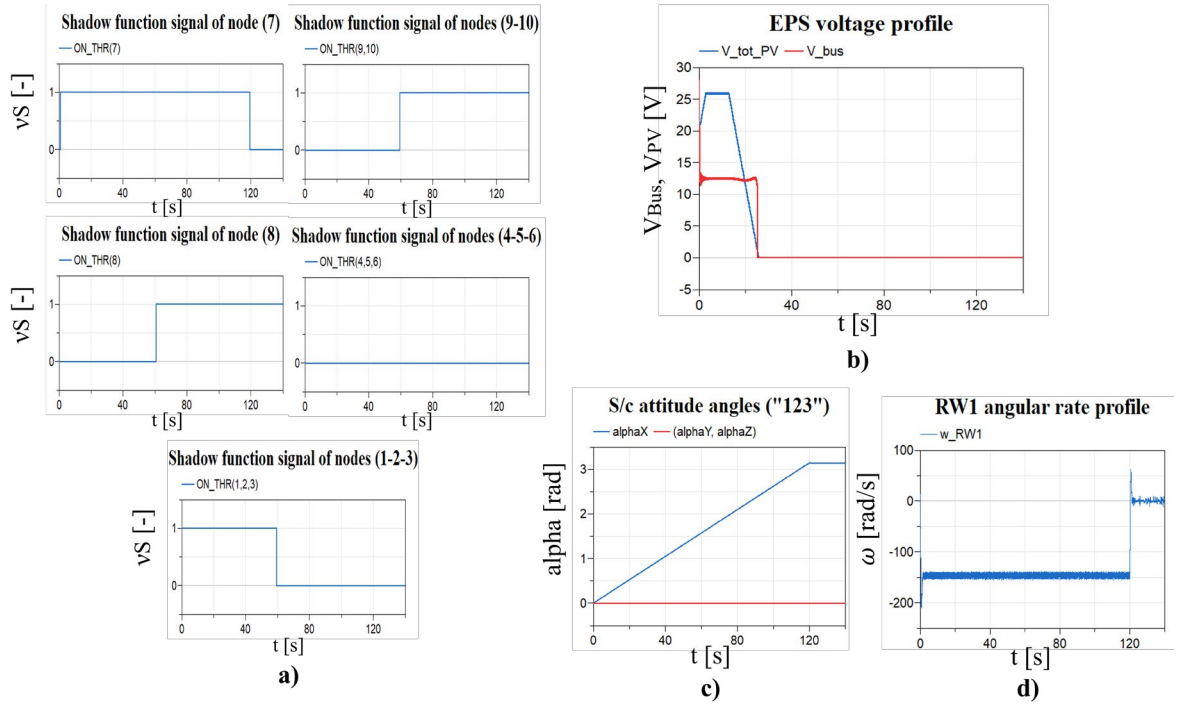


Figure 26. Satellite multiphysics simulation outcome: **a)** The shadow conditions of the satellite lumped nodes coming from the *Shadow_Model* block; **b)** The EPS voltage signals controlled by the MPPT algorithm; **c)** The satellite attitude Euler angles ("123") evolution with time ; **d)** RW1 angular rate during the maneuver

Advanced Edge Deployment: Abstracting Cyber-Physical Models via FMU Mastery

Fanping Bu[1] Mikalai Filipau[1] Nikolay Baklanov[1]
[1]Integrated Productivity and Conveyance Center, SLB, USA,
{fbu2, mfilipau, nbaklanov2}@slb.com

Abstract

Deploying cyber-physical models at the edge or in the cloud as software components is the key step of model-based-design. Depending on run-time environment, an extensive customization often needs to be made. To streamline and facilitate the deployment of models and simulators in production, a unified framework is developed. The implementation utilizes functional mockup units (FMUs) as the executable binary for the models and JavaFMI as the simulation engine. Each model deployment is encapsulated inside a microservice with all the software dependencies, with communication realized through RabbitMQ. A generalized approach to manage the model namespace has been implemented, ensuring that the FMU executor remains agnostic to changes in both model and application, as long as the AsyncAPI specification includes a mapping of the model's input-output space to the protocol's topics. Two examples are presented to illustrate the convenience and effectiveness of the proposed framework: a winch controller at the edge for oil and gas wireline operation and a wireline logging unit simulator in the Azure DevOps pipeline for software-in-the-loop testing.

Keywords: FMU, Edge, Wireline, Oil&Gas, FMI, Cyber-Physical Systems, Deployment, Microservices

1 Introduction

Rapidly evolving edge computing prioritizes convenience of deployment of advanced physical models designed for real-time control applications and data processing. The shift from monolithic software architecture to microservices has been facilitated by containerization tools like Docker and Kubernetes, which allow isolation of applications into distinct environments, thereby enhancing the scalability and manageability aspects via smaller and independently deployable services.

The workflow requires careful handling of parameters, inputs, and outputs. Dealing with unique namespaces is a part of a larger challenge - the need to manually adjust naming conventions and identifiers for each functional mockup unit (FMU) import, which significantly complicates the deployment process. We present a solution of using an interim Java layer to abstract the FMU's namespace that addresses the "at the edge integration" challenge by standardizing the interface between the FMUs and the microservices architecture.

Before discussion of specifics of our proposed framework, it is essential to provide an overview of the current state of the art. This will contextualize our work within the broader landscape of this technology block and highlight gaps and opportunities that our approach aims to address. The papers analyzed below stress the complexity and challenges involved in FMU integration and deployment, especially when FMUs from different tools form a single simulation environment.

The functional mockup interface (FMI) has been instrumental in advancing interoperability and integration within the modeling and simulation community (Gomes et al. 2018; Blochwitz et al. 2011). Multi-year efforts from various cross-domain institutions have explored diverse FMI applications. One of the earliest studies (Chen et al. 2011) introduces a generic FMU interface for Modelica for enhanced reusability and interoperability within the OpenModelica framework for multiple instances of an imported FMU. While this approach effectively facilitates FMU integration and connection within the designated simulator engine, it lacks interoperability extension to a wider range of modeling environments. The work by Cabral et al. (2018) explores FMI applications in industrial automation by enabling co-simulation (Gomes et al. 2018) per the IEC 61499 standard for distributed systems, which facilitates the virtual commissioning process by allowing co-simulation of physical plants and their PLC-based control by elevating mapping of internal variables, parameters and inputs/outputs between IEC 61499 models and the FMI. Despite its contributions to Industry 4.0 automation via paying great attention to correlation between the inter-standard data types, this research does not scale up deployment scenarios and model types and thus avoids the context of cloud and edge computing.

The co-simulation FMU-proxy framework (Hatledal et al. 2019) achieves language and platform independence using a remote procedure call (RPC) technique in a client-service architecture and offers FMU discovery. The solution significantly contributes to collaborative modeling and heterogeneous simulation expanding array

of previously unsupported languages and on incompatible platforms. However, it primarily focuses on co-simulation and intellectual property protection and does not address the emerging need for flexible and scalable model deployments, such as microservice-based architectures. A recent work (Juhlin et al. 2022) breaks long-standing lack of interoperability at the system level and presents a cloud-enabled simulation platform for drive-motor-load systems using asset administration shells [1] (AAS) and FMUs. This approach significantly enhances the flexible deployment of asset models in complex simulations by leveraging containerization. However, it does not fully exploit the potential of microservices for heterogeneous applications, as it relies on a more rigid RESTful API server architecture.

A noteworthy paper by Stüber and Frey (2021) presents a cloud-native simulation as a service (SIMaaS) implementation utilizing FMUs for co-simulation, leveraging the FMPy[2] framework. This implementation is realized as a microservice in the form of a RESTful API. In our development, though, we have identified that JavaFMI[3] is a more performant alternative (Hatledal et al. 2018). Our generalized approach for model namespace management ensures that the FMU executor remains agnostic to model-application mapping changes, with much less restrictions on FMU parametrization as offered in the analyzed paper. Furthermore, our study showcases a practical, real-time, complex industrial automation systems example (Segura et al. 2023), offering a significant advancement in solution integration over the SIMaaS demo.

2 Concepts

The method described below abstracts models from Simulink or other modeling environments, enhancing workflow efficiency and user-friendliness from inception to Dockerized edge deployment using a microservices, RabbitMQ, Linux VM, Kubernetes, and Rancher ecosystem. The revealed methodology utilizes FMI and streamlines model's input/output space, to suit better microservice deployments outside the original, often Windows-based, software ecosystem. The FMI concept, combined with our mapping explained below, ensures that abstracted and vectorized models maintain their functional integrity and ease integration with various computational environments. The JavaFMI engine enables cross-platform configuration and execution of models, regardless of the originating modeling tool and allows for scalable complexity. The presented technology extends beyond the edge, allowing physical models to be embedded as FMU objects in web applications and cloud platforms or even be invoked via command line interface during quick prototyping.

In our approach, we introduce a novel concept of model anonymization [4], which allows integrators to use the model without needing prior knowledge of the exact namespace of its inputs and outputs. We employ I/O vectorization specification and an inter-system mapping layer on top of FMU, which generalizes the interface and allows flexible interaction. This approach simplifies the integration process and also somewhat obfuscates sensitive details, while enhancing flexibility and ease of integration in complex systems. Application teams can now work with standardized interface definitions focusing on a single or a limited set of specification files, such as YAML for AsyncAPI/OpenAPI or similar, instead of navigating through specific I/O names. This technique is particularly beneficial in environments where model reusability and interoperability are paramount, providing a seamless cross-platform method for deploying and interacting with models in real-world physical systems.

The abstraction of models from Simulink or other model-based design tools for Dockerized edge deployment using FMI is still an emerging concept, particularly in the context of Kubernetes on Linux VM platforms. This area appears to be underexplored in the current literature, highlighting the avenue for our work. While the use of FMU/FMI for model exchange and co-simulation is well-adopted (Modelica Association 2022), the specific edge deployments in containerized environments is less discussed (Schranz et al. 2021).

Abstracting models to enhance workflow efficiency and deploying them as microservices on the edge, while preserving functional integrity through vectorized mapping of I/O specifications, introduces a less explored approach, particularly for the oil and gas industry, and when integration with "system of systems" architecture solutions is considered.

3 System Design and Implementation

In this section, system architecture and software stacks chosen to enable easy deployable software for control and simulation at edge or cloud are introduced. First, a microservice based system architecture is described in Section 3.1. Section 3.2 provides descriptions on how we encapsulate the designed model in an FMU and how the FMU is executed with a JavaFMI library. Section 3.3 illustrates how interfaces are defined among different microservices with AsyncAPI, and data (or messages) are shared among different microservices. Section 3.4

[1] A central concept in the context of Industry 4.0, the digital representation of an asset.

[2] https://pypi.org/project/FMPy/

[3] https://bitbucket.org/siani/javafmi/src/master/

[4] Not widely mentioned in literature in the context of Modelica, Simulink, or other modeling languages. The anonymization concept for hiding model's namespace particularities seems has little to no explicit discussions.

describes how a JavaFMI-based wrapper and RabbitMQ message passing are combined and executed inside a Docker container-based microservice. Finally, the development process of modeling and simulation microservice, and how we can utilize DevOps pipeline to automate it, is presented in Section 3.5.

3.1 System Architecture

The microservice-based system architecture is depicted in Fig. 1. An FMU, combined with a JavaFMI-based wrapper, is encapsulated within a Docker container-based microservice. Data sharing and communication among different microservices is realized through the open-source message-broker software, RabbitMQ.

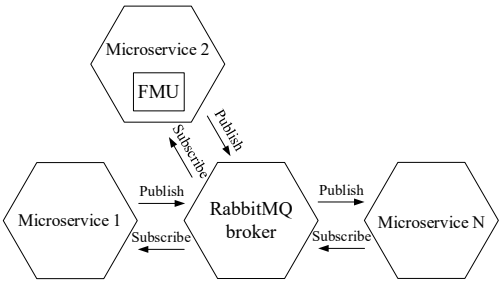


Figure 1. System architecture.

Microservice-based architecture has many advantages and is popular for both cloud and edge development. Docker container-based microservices package all the runtime dependencies and thus can be deployed across different platforms. At a high level, developers can choose appropriate modeling software tools for the development of physical system models, plants, or control algorithms. If the selected software tool supports FMU export, the exported FMU can be plugged into the proposed framework for cloud or edge deployment. In a previous iteration of our framework, we directly employed a binary Linux shared library generated by Simulink to represent the system model and control algorithm. This approach constrained development to Matlab/Simulink and tied it to a specific version of the software. With FMU, the tool selection is more flexible if it conforms to the standard.

3.2 FMU Wrapper with JavaFMI

As shown in Fig. 2, the resulting core of the modeling simulation microservice is a wrapper dealing with binary inside an FMU. There are many existing libraries that can run FMU simulations. We adopted JavaFMI for its fast execution capabilities to meet the real-time requirements of our field application deployments.

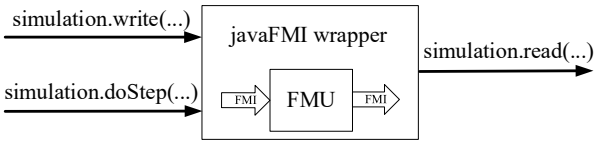


Figure 2 JavaFMI wrapper interface.

JavaFMI library is a suite of components to interact with the FMU interface. The FMU wrapper is a key component for easy access to the FMU models in co-simulation mode. It provides access to the abstract simulation class with a set of methods to interact with inputs, outputs, and parameters. A typical algorithm of application to run an FMU includes:

- Creation of simulation class with a pointer to the FMU file: *Simulation simulation = new Simulation("path/to/file.fmu").*
- Initialization with a start time and, optionally, an end time using *simulation.init (startTime, stopTime)* method.
- Writing parameters with *simulation.write* method.
- Updating inputs in a loop with desired update rate.
- Updating tunable parameters in a loop.
- Running one simulation step in a loop with the *simulation.doStep (stepSize)* method.
- Reading outputs in a loop with the *simulation.read* method.
- Resetting or terminating the FMU simulation.

Since the co-simulation mode is used, FMU has its internal sampling rate, defined during the FMU compilation stage. Method *simulation.doStep (stepSize)* requires *stepSize* time to be a result of multiplying the internal sample time by an integer number. To achieve continuous model simulation, the process involves reading inputs, updating tunable parameters, running the FMU synchronously with a specified step size, and generating outputs, as illustrated in Fig. 3.

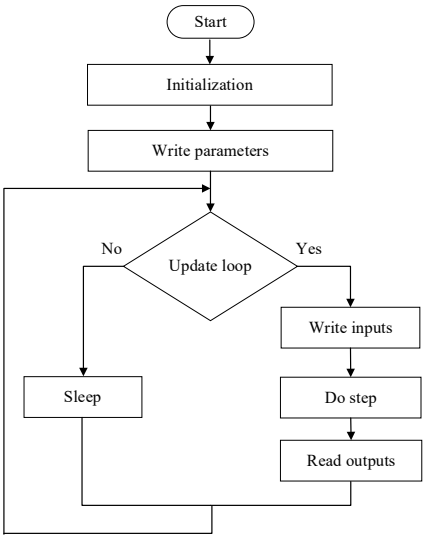


Figure 3 JavaFMI-based Java wrapper execution loop.

3.3 RabbitMQ and AsyncAPI

For proper execution and results exchange during the active phase of FMU simulation, communication with other microservices is essential. To fulfill this requirement, we have adopted RabbitMQ, an open-source message broker software. Initially designed to implement the advanced message queuing protocol, RabbitMQ has evolved through a plug-in architecture to support additional protocols such as the streaming text-oriented messaging protocol and MQ telemetry transport.

To define the specific content and format (schema) of messages exchanged among different services, AsyncAPI is used for RabbitMQ communication. The schema may reference other files for additional details or shared fields, but it is typically a single, primary document that encapsulates the API description. Furthermore, the AsyncAPI schema acts as a communication contract between receivers and senders within an event-driven system. It specifies the payload content required when a service sends a message and offers clear guidance to the receiver regarding the message's properties.

3.4 Modeling and Simulation Microservices

Modeling and simulation microservices are created by combining JavaFMI-based execution wrapper for FMU simulation and RabbitMQ communication with other microservices. The integration code is implemented in a Java loop, as shown in Fig. 4.

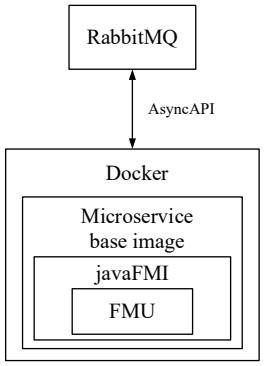


Figure 4 Modeling simulation microservice structure.

During the initialization phase, information about inputs, outputs, and parameters are extracted from the FMU's *ModelDescription.xml* to compare against the AsyncAPI payload schema. All inputs and outputs are mapped to periodically updated "state" topics to reproduce continuous input/output signals. Tunable parameters are mapped to "configuration" topics updatable by request using RPC calls; for example, to achieve the anonymization of a model, we map the I/O of an arbitrary FMU to the AsyncAPI schema referring to the *ModelDescription.xml* content. The internal structure, calculation algorithms, and the origin of the FMU will not impact RabbitMQ communication and other services. The modeling and simulation microservice with the FMU receives messages with inputs and parameters and sends messages with outputs from the model.

An example of the definition for an input port of a model in AsyncAPI properties is shown below. In this example, topic "*model.input.command.v1*" contains a link to the "*input_port"* model input of array type with four elements.

```
channels:
  model.input.command.v1:
    subscribe:
      summary: Model input topic.
      message:
        $ref: "#/components/messages/model_input"

  schemas:
    model_input:
      type: object
      required:
        - input_field
      properties:
        input_field:
          type: array
          minItems: 4
          maxItems: 4
          items:
            type: number
```

x-units: NA
x-inport: "input_port"
x-initialvalue: [1.0, 1.0, 1.0, 1.0]

With a defined AsyncAPI document, it is easy to find the corresponding "*input_port*" in ModelDescription.xml and update the *"input_port"* of FMU with the values in the message received through RabbitMQ.

The main update loop runs at a defined update rate and is shown in Fig. 5 below.

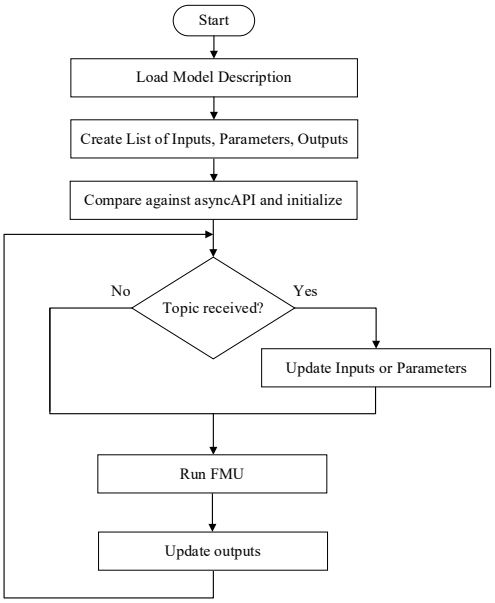


Figure 5 Modeling simulation microservice loop.

The microservice code is responsible for

- Handling received configuration messages and updating tunable parameters of the FMU by request.
- Handling received state messages and updating inputs according to the update rate and, in this way, mapping sampled time series input signal to the predefined input of the FMU.
- Executing the FMU by running the *simulation.doStep (stepSize)* method.
- Updating RabbitMQ message topics mapped to the outputs of FMU with desired update rate.
- Providing service information and statistics as periodic state messages: model time and model states (running/stopped/paused).
- Handling received control messages to start, stop, pause, or reset the FMU.

3.5 Microservice Development Process for Modeling and Simulation

Developers can choose any preferred software modeling tools to develop models for physical systems or control algorithms. FMUs can then be exported. Modeling and simulation microservice can be built with following steps, as shown in Fig. 6:

- Access is added to the FMU utilizing wrapper with the JavaFMI library.
- Custom microservice implementation provides an interface to the AsyncAPI and synchronization of FMU execution and API state messages to and from the microservice.

The last stage is deployment of the microservice as a Docker container as a part of complex software application.

All steps in building the pipeline may be automated and integrated into DevOps pipeline (for instance, Azure DevOps) with automated integration tests and API validation, providing safe and robust application deployment.

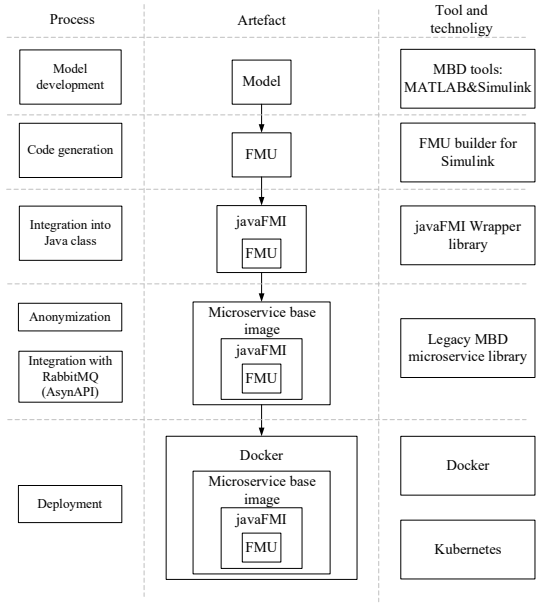


Figure 6 Modeling simulation microservice development process.

4 Case Studies

To illustrate the effectiveness, convenience, and versatility of the proposed framework in real production software deployment, wireline automation development for assisted conveyance in oil and gas operations at SLB is used as an example. Two design cases are presented. In the cloud, a wireline winch and cable simulator is deployed in the Azure DevOps pipelines for software-in-the-loop testing. At the edge, a winch controller is deployed for an automatic winch control. Although the

application environments are different between cloud and edge, the same framework can be used due to the portability of the FMU and microservice.

Wireline operation is widely used in the oil and gas industry to measure the properties of a formation using electronic instruments. Fig. 7 illustrates a typical wireline logging operation. A drum of electric cable is driven by a hydraulic winch with a toolstring packed with different formation measurement sensors attached at the free end of the cable. The winch drum is rotated by a hydraulic motor that moves the toolstring up and down along the wellbore. Sensors packaged inside the toolstring conduct sensing measurements while moving and send back measurement results through the connected cable. During operation, an operator is required to control the hydraulic winch manually so that the toolstring movement will follow a desired motion profile.

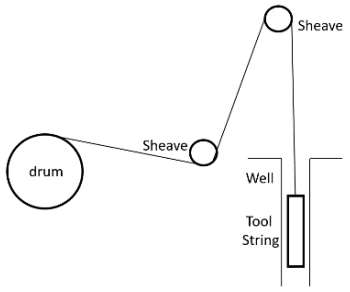


Figure 7 Wireline logging operation for oil and gas industry.

4.1 Winch Simulator

The hydraulic winch drives the drum via a gear transmission. As shown in Fig. 8, the hydraulic winch is a hydrostatic transmission system consisting of a variable displacement pump, a variable displacement motor, and a charge pump. The pumps are driven by a vehicle engine through gears. The drum is driven by variable displacement motor through transmission gears.

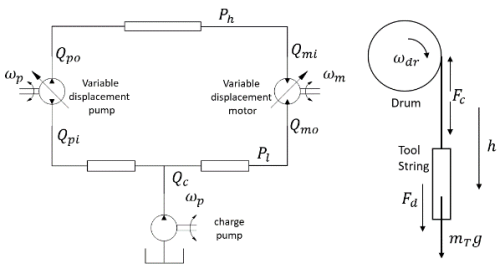


Figure 8 Schematics of a hydraulic winch.

The winch simulator model is developed using the Simulink/Simscape package from MathWorks. The hydraulic circuit is modeled using components from the MathWorks' fluids library and is derived from the hydrostatic transmission example[5] shown in Fig. 9.

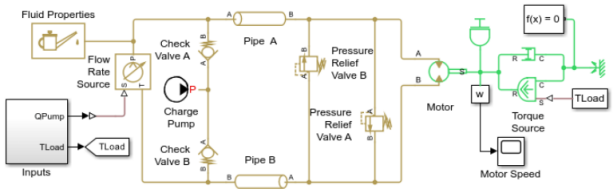


Figure 9 An exemplary model of a hydraulic circuit.

During wireline operation, the released cable can be over thousands of feet long. Therefore, it is necessary to model the dynamic effect of cable elasticity. The entire cable is discretized into serialized mass-spring-damper blocks, as shown in Fig. 10.

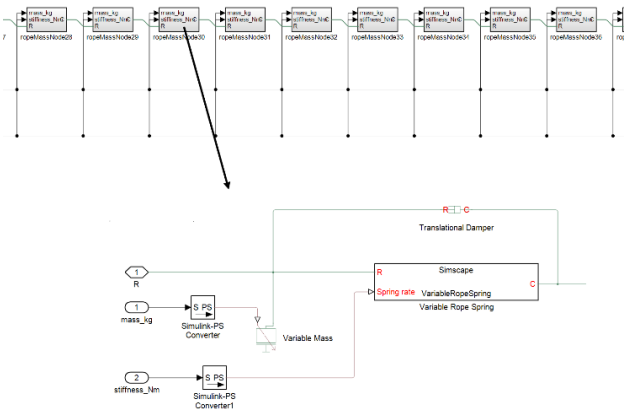


Figure 10 Mass-spring-damper node and serialized cable model.

Calibration tests are conducted to collect data and identify system parameters. The calibrated system response matched actual system behavior, as shown in Fig. 11.

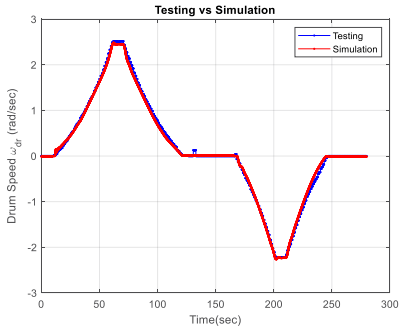


Figure 11 Simulation vs testing data.

[5] https://www.mathworks.com/help/hydro/ug/hydrostatic-transmission.html

Following the procedure described in previous sections, FMU is exported from the Simulink/Simscape model and packaged into a microservice. The resulting simulator microservice was used for the software-in-the-loop testing of the winch controller and other software services of wireline automation. Software developers with no knowledge of modeling software such as Matlab/Simulink/Simscape could easily incorporate the simulator microservice into their test need to mimic hydraulic winch behavior if they follow the interface specified in AsyncAPI. Those tests can be made automatic and regressive and can run in Azure DevOps pipeline as is done today for wireline automation development in SLB. Other than software testing, the FMU-based simulator microservice can be deployed in the cloud or at the edge as the core of digital twin applications for predictive maintenance, operation planning, and optimization.

4.2 Winch Controller

As the first step toward automation of the wireline logging operation, it is necessary to control the hydraulic winch, or toolstring motion, following a desired motion profile, automatically, without operator intervention. A nonlinear model based adaptive robust controller (ARC) is designed for this purpose. The detailed controller design can be found in Bu (2020). The designed controller is constructed in Simulink, as shown in Fig. 12.

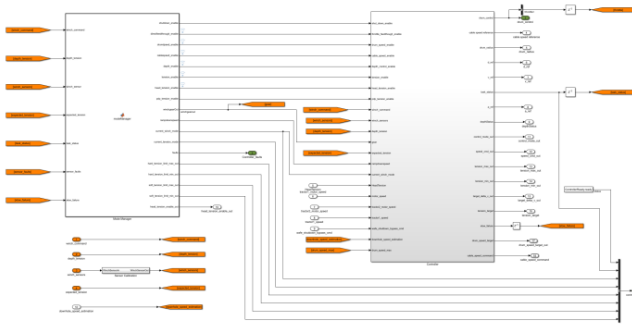


Figure 12 Simulink diagram of winch controller.

The winch controller developed in Simulink can be exported to FMU and packaged into the microservice the same way as the winch simulator in previous sub-section. The winch controller can be deployed via two scenarios, as shown in Fig. 13:

- Software-in-the-loop testing is executed in the Azure DevOps pipeline in the cloud together with the winch simulator as a virtual winch. In this case, a software "switch" will map winch simulator inputs/outputs to the proper RabbitMQ messages. From the winch controller point of view, it is receiving sensor inputs, and sending out actuator commands, from/to the real hardware.
- In the “edge at wellsite” application, the winch controller microservice is deployed at the edge, namely at the automation server physically installed inside the wireline logging unit at a wellsite. The software "switch" will map real sensors and actuator signals from the hardware interface microservice to the proper RabbitMQ messages. The winch controller will be able to control the actual winch drum for automation purposes.

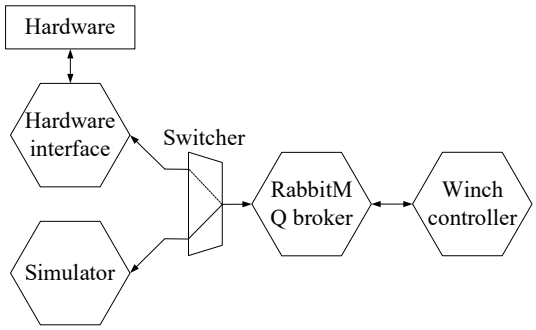


Figure 13 Winch controller deployments.

It should be noted that by adopting a model-based-design process, the winch controller has been matured to the product level and deployed in SLB’s wireline logging units globally, completing over several million feet in automated conveyance services.

5 Conclusions

Our work presents a versatile unified framework that is not bound to specific platforms able to generate FMUs, expanding FMU support to a broad range of environments beyond Simulink, Modelica, and similar systems. It covers the deployment needs of arbitrary multidomain simulation engines. By fusing FMUs, JavaFMI, and an AsyncAPI-based anonymization layer[6], we provide a standardized platform for co-simulation and verification suitable for industrial automation systems and well beyond, demonstrated with real-world O&G oil and gas applications. This could be applied, but is not limited to, advanced wireline conveyance assistance systems comparable to automotive ADAS (MathWorks 2024; Dassault Systemes 2024) and autonomous driving functionalities.

Our microservice-based architecture is both flexible and scalable, serving the needs of both edge and cloud model

[6] not limited to, can be a RESTful API wrapping

deployment, promoting collaboration and efficiency in composite asset engineering. The presented comprehensive approach enhances interoperability and streamlines the model delivery process in production software. It allows the automated wrapping and execution of highly arbitrary models at the small cost of introduction of a very thin model-to-RabbitMQ I/O mapping.

Our results demonstrate that the proposed solution enables scalable, flexible, and practical modular deployment of models as software components in cyber-physical and control systems, with a particular focus on Docker and Kubernetes for real-world commercial products in modern computing environments (Segura et al. 2023). Cross-platform model wrapping, configuration, and execution enable the reuse of models in various deployment scenarios.

Our future work focuses on scalability and performance optimization for large-scale deployments, enhancing security measures for edge and cloud, and integrating alternative communication protocols for broader interoperability. The introduced solution utilizes the FMI 2.0 standard, and we are moving onto FMI 3.0, which lets us naturally reduce restrictions for data types. Additionally, we are exploring automated mechanisms for model updating and versioning, with deeper DevOps pipeline integration.

Acknowledgements

The authors would like to thank Ken Ditlefsen (SLB) for his initial “abstract executor” architecture and non-FMU code implementation, which we developed and elevated to the solution presented in this paper. We also express our appreciation to MathWorks for providing consultancy services for their tools, which significantly contributed to the success of our project.

References

Blochwitz, Torsten et al. (2011). “The Functional Mockup Interface for Tool independent Exchange of Simulation Models”. In: *the 8th International Modelica Conference,* 105-14. Dresden, Germany. DOI: 10.3384/ecp11063105

Bu, Fanping. (2020). "Nonlinear adaptive robust motion control for hydraulic winch in oil and gas wireline operation." In: *21st IFAC World Congress*, 8991-96. Berlin, Germany. DOI: 10.1016/j.ifacol.2020.12.2015

Cabral, J. et al. (2018). "Enable Co-Simulation for Industrial Automation by an FMU Exporter for IEC 61499 Models." In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 449-55. Turin, Italy. DOI: 10.1109/ETFA.2018.8502654

Chen, Wuzhu et al. (2011). "A Generic FMU Interface for Modelica." In: *the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, 19-24. Zurich, Switzerland.

Dassault Systemes. 2024. "Dymola". URL: https://www.3ds.com/products/catia/dymola.

Gomes, Cláudio et al. (2018). "Co-Simulation: A Survey", In: *ACM Computing Surveys*, 51: 1-33. DOI: 10.1145/3179993

Hatledal, L. I. et al. (2018). "FMI4j: A Software Package for working with Functional Mock-up Units on the Java Virtual Machine." In: *The 59th Conference on Simulation and Modelling (SIMS 59)*. Oslo, Norway. DOI: 10.3384/ecp1815337

Hatledal, L. I. et al. (2019). "FMU-proxy: A Framework for Distributed Access to Functional Mock-up Units." In: *the 13th International Modelica Conference*. 79-86. Regensburg, Germany. DOI: 10.3384/ecp1915779

Juhlin, P. et al. (2022). "Cloud-enabled Drive-Motor-Load Simulation Platform using Asset Administration Shell and Functional Mockup Units." In: *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1-8. Stuttgart, Germany. DOI: 10.1109/ETFA52439.2022.9921678

MathWorks. (2024). 'What Is ADAS? 3 things you need to know'. URL: https://www.mathworks.com/discovery/adas.html.

Modelica Association. (2022). "Functional Mock-up Interface for Model Exchange and Co-Simulation. Version 3.0 " Modelica Association.
URL: https://fmi-standard.org/docs/3.0.1/

Schranz, Thomas et al. (2021). "Portable runtime environments for Python-based FMUs: Adding Docker support to UniFMU." In: *14th Modelica Conference 2021*. 419-424 Linköping, Sweden. DOI: 10.3384/ecp21181419

Segura, J., Tran, V.V., Meirkhan, J. et al. (2023). "Autonomous Slickline and Wireline Conveyance Improves Performance of Offshore Interventions". Paper presented at the SPE Offshore Europe, Aberdeen, Scotland, 5–8 September. SPE-215586-MS. DOI: 10.2118/215586-MS

Stüber, Moritz, and Georg Frey. (2021). "A Cloud-native Implementation of the Simulation as a Service-Concept Based on FMI." In: *14th Modelica Conference*, 393-402. Linköping, Sweden. DOI: 10.3384/ecp21181393

Integrating Generative Machine Learning Models and Physics-Based Models for Building Energy Simulation

Luigi Vanfretti[1] Christopher R. Laughman[2] Ankush Chakrabarty[2]

[1]ECSE Department, Rensselaer Polytechnic, Troy, NY, USA, vanfrl@rpi.edu
[2]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA, {achakrabarty,laughman}@merl.com

Abstract

This paper describes the integration of generative deep learning models for data-driven building energy simulation. The generative models (GMs) are trained to learn distributions of building input signals from data using Python and PyTorch and interfaced with physics-based Modelica models. The developed integration requirements provide background on typical needs that focus on building energy simulation performance. Simulation examples using models from the Buildings library, refactored to receive GM inputs, are presented to illustrate the benefits of the proposed integration approach and how GMs can be used for building energy performance analysis.

Keywords: Machine learning, generative models, Buildings library, building energy simulation

1 Introduction

Building simulation tools are frequently used during the design phase to size equipment and perform simulation-based studies that help estimate annual energy use or carbon emissions. The demand for such simulation studies, combined with the emergence of new design scenarios such as building electrification, has driven the creation of advanced physics-based building simulation models. The Modelica Buildings library (Wetter, Wangda Zuo, T. S. Nouidui, et al. 2014) is one of the best-known collections of such models, which enable the simulation of the coupled dynamic behavior of building envelopes and heating, ventilation, and air conditioning systems (Chakrabarty, Maddalena, Qiao, et al. 2021; Zhan, Wichern, Laughman, et al. 2022). Modelica-based tools offer distinct benefits in analyzing building performance, as they facilitate systematic controller design (Wetter, Ehrlich, Gautier, et al. 2022) and realistic closed-loop control performance assessment (Stoffel, Maier, Kümpel, et al. 2023).

While such physics-based Modelica models can effectively simulate the energy and mass transfer processes for the building envelope, together with the thermofluid physics of HVAC systems, there are other processes that influence the heating and cooling load that the HVAC system will experience that are not driven by physics alone, but also by human actions. Building occupants generate and absorb latent, sensible and radiant heat, and their actions can significantly impact the efficiency of an HVAC system in terms of energy usage, comfort levels, and indoor air quality, among other factors (Mirakhorli and Dong 2016). As models of such behavior are also required for building design and performance analysis, a common practice is to make engineering assumptions that define a 'nominal' behavior for variables such as occupancy (number of people occupying a zone), activity level and schedule, and then augmenting this nominal model by representing the time-varying behavior as an input disturbance (e.g. a constant or ramp) during simulation. The reliability of simulation outcomes is compromised by such a limited representation of human behavior variables influenced by human behavior, as these variables are challenging to model using physics-based or first-principles methods.

Data-driven approaches have demonstrated their efficacy in characterizing the observed distribution of single-output operational building profiles, such as energy usage (Ye, Strong, Lou, et al. 2022), thermal comfort (Das, Tran, Singh, et al. 2022), and occupancy patterns (Chen and Jiang 2018). A diverse set of building simulation scenarios, including typical or extreme occupancy patterns for a specific building, could be created by integrating these machine learning-based generative models with Modelica-based building models. Such tools could assist in pinpointing improvement opportunities in the existing HVAC system (i.e. retrofitting) or assess the effectiveness of a particular control scheme for existing buildings. and would also enable the creation of data-driven occupancy models to be used in building design when specifying and calibrating the HVAC system to be deployed.

Although explicit neural network (NN) models have previously been created in Modelica (Codeca and Casella 2006), this approach does not enable integration with major ML platforms for NN design (e.g., PyTorch), and attempting it would require a ground-up reimplementation in Modelica. This would require a parallel effort to that in the discipline of machine learning, where advancements in ML platforms are made rapidly and by a large community compared to that of Modelica specialists. Recent efforts have also been made to integrate NNs with physics-based simulators, including the use of the Functional Mock-Up Interface (Modelica Association 2019) to exchange the trained NN model with other frameworks (The MathWorks n.d.). Two approaches of note that

support the use of trained NN models directly in Modelica include the `SmartInt` (XRG Simulation GmbH n.d.) library, which allows the use of TensorFlow TFLite models (XRG Simulation GmbH n.d.) and the `NeuralNet` library, which supports the Open Neural Network Exchange (ONNX) format (Wolfram Research n.d.). Unfortunately, limitations in the implementations of both of these libraries make them unsuitable for this work. In particular, the lack of support for NN models trained in `PyTorch` and the added complexity caused by its dependencies (e.g. TensorFlow API) made it difficult to use the `SmartInt` library, whereas the `NeuralNet` library is only available for use in the Wolfram SystemModeler Modelica tool where, as of the time of writing, models from the `Buildings` library cannot be compiled. Moreover, this tool requires the use of the ONNX Runtime library and its C API, which adds not only complexity but also overhead in software integration.

In this study, we use trained Generative Models (GMs) to provide input signals for building models, such as building occupancy and power demands. These GMs are specified via a set of input parameters and provide causal outputs to the building models, and as such provide valuable "component models" for the overall building representation. While this could have been integrated with the building models via FMI, we chose to build a direct connection between the GMs and Modelica tools to facilitate the iterative refinement of the building models and avoid the inherent trade-offs that model exchange or co-simulation has on simulation performance (Schweiger, Gomes, Engel, et al. 2019). We thus implement a requirements-based light-weight integration of generative models trained in Pytorch with building simulation models in a Modelica environment, which is accomplished via the external function standard interface, as is also done in the `SmartInt` and `NeuralNet` libraries. This approach can be of value to other simulation researchers or practitioners as, for simulation purposes, it maintains a minimum number of dependencies and attempts to prioritize simulation performance.

The remainder of this paper is organized as follows. Section 2 lists the requirements considered for the proposed implementation shown in Section 3. The simulation results obtained through this integration approach are illustrated using a model from the Buildings library, and a GM trained in PyTorch using real-world data are presented in Section 4. Conclusions and further work are summarized in Section 5, which concludes the paper.

Conventional Modelica notation is used extensively in this paper. The `typewriter` font is used along the dot notation to reference the syntax of the Modelica language, including the names of Modelica libraries, models names, etc. Furthermore, the `typewriter` font is used to refer to the names of other software packages. Meanwhile, the dot notation is used to specify hierarchy in object-oriented modeling. As an example, consider the model of a `building` containing a zone named `zon`, which itself is composed of a room named `roo`. To access a parameter value, for example, the constant convection coefficient for room-facing surfaces of opaque constructions, `hIntFixed`, the dot notation would be `building.zon.roo.hIntFixed`. Finally, syntax in code listings follows same as that in the Modelica Language Specification[1].

2 Model Integration Requirements

To combine GM models with building simulation models based on Modelica, various factors have to be taken into account. We outline the scope in three main categories: 1) training and modeling of GM, 2) integration of GM models with building models, and 3) automation of the simulation workflow. Figure 1 illustrates these categories and shows how they interact to support building simulation.

2.1 GM Training and Modeling

An important consideration is that the GM models must be designed to interact seamlessly with the building simulation model. The following requirements (Req.) must be met by the GM models and their integration. These requirements result in the implementation shown in Figure 1(A), which is discussed in Section 3.

Req. 1: Deep generative networks should be easily trained (for example, in PyTorch) with real building data. This requirement emerged from the need for a research-focused framework on machine learning that provides flexibility and ease of experimentation to test new methods such as the one in (Salatiello, Wang, Wichern, et al. 2023). Furthermore, the design of the GM neural architecture should be such that the length of the output (e.g., number of days a signal is generated) can be easily provided as a user-input.

Req. 2: The parameters of the trained generative model must be exchanged in a manner that ensures they are stored in the smallest possible file formats. Reading the files must be fast and efficient, especially because modern GM architectures contain a very large number of trained parameters. In this work, our GM has multiple sub-network components to be trained, but for signal generation (i.e., at inference), only a small sub-module of the deep network is required: therefore, only a small subset of the GM weights need to be stored.

Req. 3: The generative model must be incorporated into the simulation environment and should operate with high computational efficiency. The computational load of running the generative models in conjunction with the building model should be minimal (or insignificant) when compared to running the building model by itself.

[1] See Ch. 1.4 Notation in the Modelica Language Specification: `https://specification.modelica.org/master/introduction1.html`.

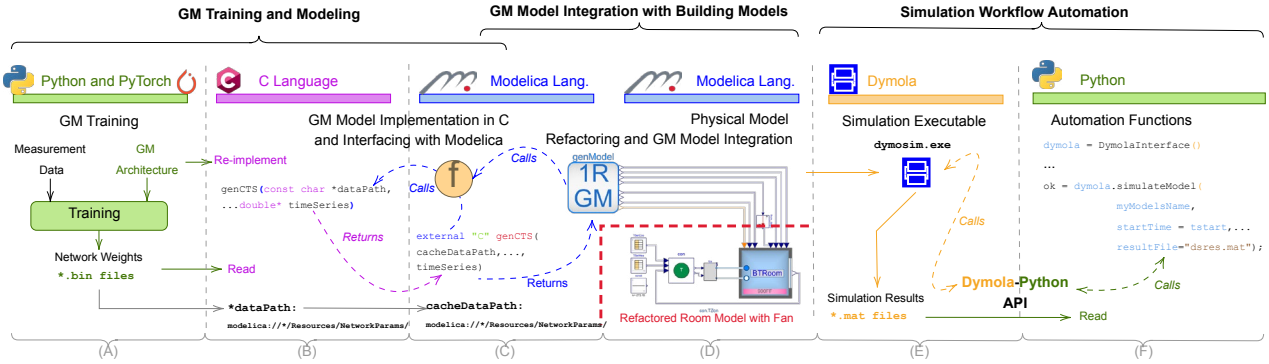


Figure 1. Overview of the Desired Integration of a Generative Model with a Single-Zone Thermal Model of a Building

2.2 Interfacing Deep Generative Networks with Building Simulation Models

Because the building models are implemented in the Modelica language; thus, it is necessary to integrate the GMs with them. To interface them the following requirements are made:

Req. 4: The GMs need to be incorporated into the building simulation model using a tailored `block`. Inputs to the GM should be routed through the Modelica model. Outputs from the GM should be connected via `RealOutput` or `IntegerOutput` interface blocks from the MSL. These interfaces can, for instance, be used to provide the occupancy, equipment, ventilation, and lighting loads of the building.

Req. 5 The GM's output should generate a time series that fills a `CombiTimeTable` from the MSL with predicted variable values at specified time intervals. The lookup table must implement a sample-and-hold mechanism and suitable methods for extrapolation for values outside its defined range for each variable. These features should be included within the `block` described in Req. 4.

Req. 6: The `block` from Req. 4 must also provide the parameters for the deep generative network. In the case of the weights obtained through training, a string parameter will indicate the location of the file(s) storing the weights.

These requirements guided the implementation shown in Figures 1 (see labels (B)-(D)) and 2, which are discussed in Section 3.

2.3 Simulation Workflow Automation

Referring to Figure 1 (see labels (E) and (F)), one final aspect to consider in the scope of this work is that of simulation. As illustrated above label (E), once the GM and building models are integrated, it is possible to create a simulation executable and obtain simulation results. It is beneficial to offer a method for automating the workflow, specifically to alter parameters in the Modelica model programmatically, herein reflected by:

Req. 7: When possible, the simulation executable shall be reused, i.e. limit the re-translation/compilation of the Modelica model, to perform trade-off analysis studies. Such studies shall include changing any of the parameters of the Modelica model, and allow for post-processing of the simulation results.

3 Prototype Implementation

To meet the requirements emerging from the three aspects considered in the previous section, the design choices and implementation pursued are defined next. For illustrative purposes, Figure 1 shows how the implementation was carried out to meet the requirements. A detailed description of the implementation to meet the requirements above labels (B)-(D) in Figure 1, explaining how the interfacing between C and Modelica of the GM models was done, is shown in Fig. 2.

The main goal of the software integration approach used was to minimize dependencies (for both ease of portability and simulation performance purposes) on external software tools other than the C compiler and the Modelica tool, in this case Dymola (Bruck, Elmqvist, Olsson, et al. 2002; Dassault Systemes AB 2023), which requires a `C` compiler itself. Hence, an attractive feature (as discussed in the Introduction) is the use of the standardized external function feature of the Modelica language. Consequently, to minimize dependencies, the integration of GMs with the simulation model must be done solely using `C`. In turn, this requires one to interface the `C` implementation of the GMs with a Modelica model that can call the `C` code. Meanwhile, simulation workflow automation can be achieved by interacting with the Dymola-built simulator executable, which in the case of the approach shown in Figure 1, contains *both* the GM and the building models, with a suitable scripting tool supported by Dymola.

3.1 GM Training and Modeling

Training the NN of the GM was conducted using Python and PyTorch, as illustrated in Figure 1, in the portion over the under-brace labeled with (A), which helps to meet Req. 1. To train the models, measurement data and the designed architecture for the NN are provided in PyTorch to perform the training. This results in the weights of the network, which are stored in `*.bin` files, i.e. binary data, which helps to meet Req. 2.

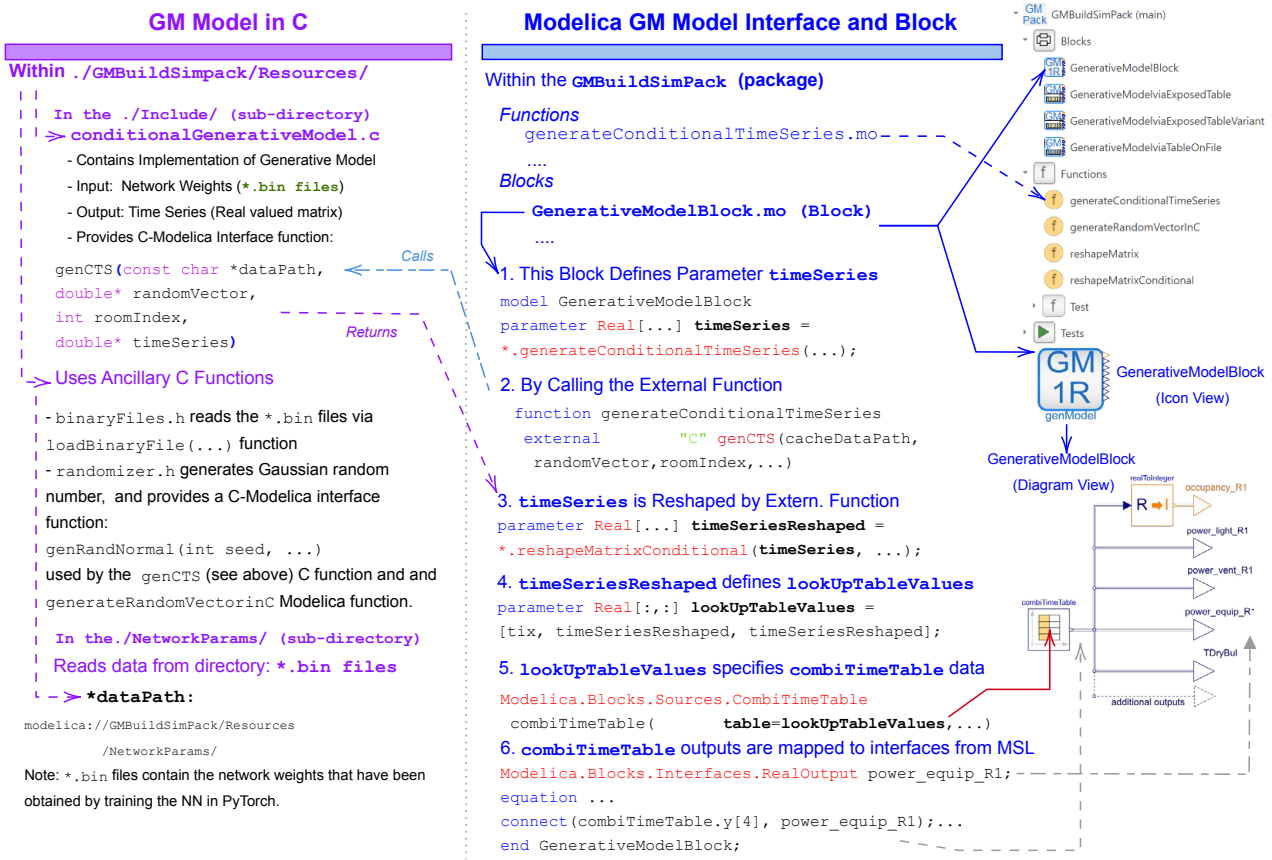


Figure 2. Integration between C and Modelica

To meet the third requirement (Req. 3), the GM needs to be translated to C, as illustrated above label (B) of Figure 1. This implies that the network architecture and different activation layers need to be coded, while the weights resulting from training only need to be read. To illustrate this, consider Listing 1. In lines 2-4, the NN's graph is defined via a structure, i.e., the architecture is defined. Here, only one hidden layer is shown; however, the NN has others (see (Chakrabarty, Vanfretti, et al. 2024)). Next, in lines 6-10, the activation functions are defined, here only the Softplus(x) function is shown; however, other functions are used (see (Chakrabarty, Vanfretti, et al. 2024). Finally, in lines 11-23, the fourth layer's output is calculated using the output of the previous layer, applying the weights for this layer, adding biases, and finally applying the Softplus(x) function.

Listing 1. Excerpt of the GM Implementation in `conditionalGenerativeModel.c`

```
/* Structure to represent network graph */
typedef struct {...
    cLayer hidden4_output;
} cGenerativeModel;
...
/* Sample Activation Functions */
double c_SoftPlus(double x) {
    return log(1 + exp(x));
}
...
/* Sample Network Layer and Output */
double* c_forward(cGenerativeModel network,
    double *input){
...
    // Hidden Layer to Output
    for (int i = 0; i < c_layerSizes[5]; i++) {
        output[i] = 0;
        for (int j = 0; j < c_layerSizes[4]; j
            ++) {
            output[i] += hidden4[j] * network.
                hidden4_output.weights[i *
                c_layerSizes[4] + j];}
        output[i] += network.hidden4_output.
            biases[i];
        output[i] = c_SoftPlus(output[i]);
```

While C-implementation of such kind of NN's is not trivial, this choice was intentionally made taking into account the needs to minimize dependencies and maximize simulation performance. With the proposed approach, the GM model becomes part of the source code of the simulation executable, in Dymola (called `dymosim.exe`, see Figure 1(F)). In addition to this, other C functions are needed, e.g., to load the binary files, and provide other functionalities (see Figure 2). Finally, in addition to implementing the network, the C code needs to include a function to interface with Modelica, as discussed next.

3.2 Interfacing GM Models with Building Models

To fulfill the requirements pertaining this aspect, i.e. Req. 4-6, it is useful to refer to Figure 1, paying

attention to what is presented above the labels (B)-(D). This gives a high-level overview of the main parts of the interfacing. As can be observed, R4 and R5 are fulfilled by invoking a Modelica function, `generateConditionalTimeSeries` (see above label (C) 1), which provides input parameters and invokes the C-function, `genCTS` (see above label (B) 1), evaluating the GM and asking for its output to fill a lookup table called `timeseries`. Meanwhile, to fulfill Req. 6, the `timeseries` output is provided to a table within the `genModel`, which is a `block` that is interfaced with the building model (see above label (D) in Figure 1).

To expand on this overview and understand the integration, refer to Figure 2, which focuses only on the integration between C and Modelica for the GM alone. In the LHS, the C implementation is shown, in the middle the Modelica text is shown, and in the RHS the Modelica graphical views for the developed package and the main block component are shown. This figure should be perused from left to right to understand the implementation details and from right to left to understand how the different pieces work together from a user perspective.

Reading from the right-hand side (RHS), the RHS corner of Figure 2 shows the structure of the `GMBuildSimPack` package. It contains the Modelica function and `block` that help to integrate the GM model with the building. For the user, the `GenerativeModelBlock` would be used to drag, drop, and connect with the building model inputs. When graphically instantiated as `genModel`, as shown by the icon view, one `IntegerOutput` and several `RealOutput` interfaces that route the output of the GM model. This fulfills Req. 4. Meanwhile, the diagram view of this block helps to see them in more detail, and to observe that they are connected to a `CombiTimeTable`, which was what Req. 5 requested. This table will be populated with the `timeseries` output of the GM model, which requires a few steps that are explained next.

Now, to understand how the `CombiTimeTable` gets the GM model data, it is necessary to understand how the Modelica function, `generateConditionalTimeSeries`, interfaces with the C function `genCTS`. This is illustrated in Listing Listing 2 that shows the call to the `external "C"` function in line 5, while also passing input parameters to run the NN (lines 2-4), and obtaining the GM's output in line 5 of the listing, i.e., `Real[384] timeSeries` [2].

Using the `GenerativeModelBlock` in Listing 3, will call `genCTS` in line 7, while providing it with different required parameters, see lines 2-5. The function `generateConditionalTimeSeries` in Listing 3 provides the `timeSeries` output that will pass its data to `lookUpTableValues` in Line 9. Next, in line 7, `CombiTimeTable` is instantiated and data are provided through the `lookUpTableValues` parameter. There are several steps required in this process, which are listed as steps 3 and 4 in Fig. 2. Note that in 10 of Listing 3, several modifiers that are needed have been ommitted; these include those required to set-up the sample-and-hold and periodic extrapolation (i.e. `smoothness=...` and `extrapolation=...` modifiers), and the `timeScale=...` modifier defines the GM's output rate, which will be 15 min in the examples in the forthcoming section. Finally, in line 11, `power_equip_R1` instantiates a `RealOutput` interface that is connected in Line 14 to the corresponding output of the table, i.e. `CombiTimeTable.y[4]`.

Listing 2. External Function in Modelica Linking the GM's Output

```
function generateConditionalTimeSeries                          1
  input String cacheDataPath = Modelica.                        2
      Utilities.Files.loadResource("modelica://
      GMBuildSimPack/Resources/NetworkParams/");
  ...                                                           3
  output Real[384] timeSeries;                                  4
  external "C" genCTS(cacheDataPath,                            5
      randomVector, conditionalInputs,
      timeSeries) annotation (
    IncludeDirectory="modelica://GMBuildSimPack/                6
        Resources/Include",
    Include="#include \"                                        7
        conditionalGenerativeModel.c\"");
end generateConditionalTimeSeries;                              8
```

Listing 3. Excerpt of the Source Code of the Generative Model Block

```
model GenerativeModelBlock                                      1
    constant String cacheDataPath=Modelica.                     2
        Utilities.Files.loadResource("modelica
        ://GMBuildSimPack/Resources/
        NetworkParams/");
    /* Network Input Parameters */                              3
    constant Integer latentDim = 8;                             4
    ... /* Other parameters ommited.                            5
    /* Network Output */                                        6
    parameter Real[nSamplesPerDay*nSignals]                     7
        timeSeries = GMBuildSimPack.Functions.
        generateConditionalTimeSeries(
        cacheDataPath, randomVector,
        conditionalInputs);
    ... /* Ommitting: reshape timeSeries into                   8
        lookUpTableValues */
    Modelica.Blocks.Sources.CombiTimeTable                      9
        combiTimeTable( table=lookUpTableValues
    ... /* other modifiers ommitted */)                         10
    Modelica.Blocks.Interfaces.RealOutput                       11
        power_equip_R1;
    ... /* Other interface instantiations                       12
        ommitted */
equation                                                        13
 connect(combiTimeTable.y[4], power_equip_R1);                  14
 ... /* many connect statmenets ommited */                      15
end ConditionalGenerateTimeSeriesModel;                         16
```

Under this category, only one requirement needs to be addressed, Req. 6. To understand how this is fulfilled, it can be observed in both Listings 2 (see line 2) and 3 (see

[2]Note that in this prototype implementation the size of the output `timeSeries` is fixed to 384 for illustration purposes. In a more generic implementation, this parameter could be propagated to make it easier for the user to modify it.

line 2) that the string called `cacheDataPath` points to a specific directory where the `*.bin` files are located. It should be noted in this Listing 2 the `annotation` points Dymola to the location where the file that includes `genCTS` is located, so that it can be included as part of the integrated simulator code.

Finally, additional Modelica and C functions are built to support the workflow. For example, the NN must be initialized with a random vector, this is done through ancillary C code whose functions are depicted on the RHS of Fig. 2, one of which has an accompanying Modelica function, i.e. `generateRandomVectorInC`. Meanwhile, as it can be observed in steps 3-4 in Fig. 2, additional Modelica functions (i.e., `reshapeMatrix`), reshape the GM's output before feeding it to the table. We omit further details about the integration of these features, as it is similar to that explained for `genCTS` and `generateConditionalTimeSeries`.

3.3 Simulation Workflow Automation

Finally, to facilitate the automation of the simulation workflow and meet Req. 7, the only design choice to make is the selection of one of the available scripting interfaces provided by Dymola. Among the various interfaces, the most attractive is the Python Interface for Dymola (Dassault Systemes AB 2023), an API to execute Dymola commands using a Python program. This choice was made because Python is already being used to train NN models via PyTorch. Using this interface, the models parameters, weather data files, etc., can be specified and used for specific simulation cases.

As shown in Figure 1 the Dymola-Python Inteface allows to change the value of the models parameter within the simulation executable, by instantiating the interface (i.e. `dymola = DymolaInterface()`), and running a simulation through one of its commands (i.e. `dymola.simulateModel(...)`. To avoid the need of retranslating/compiling the model, one option is to first translate the model (see Chp. 1.3 of the Modelica Specification) using the `translate` command of the Dymola-Python interface, which generates the code of the simulator that can simulate the model. Thus, every time that parameters are changed[3] within a look, the model does not need to be translated; i.e. code generator is avoided, reducing time.

4 Results

4.1 Building Models and GM Training Data

4.1.1 Building and System Model

In this section, examples demonstrating the integration of GM and building models will be presented using the system model depicted in Figure 3, which is divided into three parts. The segment labeled (A) is designated for setting the temperature setpoints (`TSetCoo` and `TSetHea`), the segment labeled (B) includes the physics-based and GM models that will be elaborated on later, and finally, the segment (C) carries out calculations to track building performance metrics such as the zone's temperature (`TRoom`).

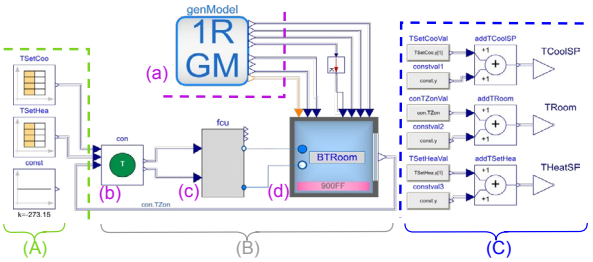


Figure 3. GM integrated with a Re-factored Single-Zone Building Model including a Fan Control Unit

Let us now describe the physics-based models. Above the segment labeled (B) in Figure 3, there are four components. GM models are identified with the label (a) and have been described in detail in Section 3.2. Labeled (c), a simple fan coil unit (FCU) is included to condition the building, which is shown in Figure 4. The FCU is regulated by a simple thermostat, which is modeled by a dual proportional and integral (PI) controller with dual set point, shown in Figure 5, to maintain room temperature within the set points of heating and cooling. When the

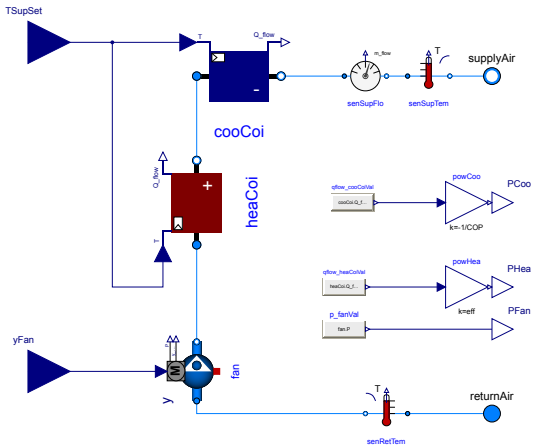


Figure 4. Fan Coil Unit labeled (c) in Figure 3

FCU is activated, the supply fan runs at a constant speed to circulate air through the heating and cooling coils, indicated in Figure 4 as `heaCoi` and `cooCoi`, respectively. The heating and cooling set points are converted to the supply air temperature set point by the PI controller shown in Fig. 5[4], and the coils are activated to reach the set point. The conditioned air is then supplied to the building through the `supplyAir` interface in Figure 4, where it is assumed to be well mixed. For the illustrative purposes of the examples herein, the energy impact of FCU is

[3] Provided that the parameter to be changed is non-structural (Modelica Association 2017).

[4] Observe that the goal here was to provide a simple implementation of the thermostat. To avoid numerical issues that could appear due to the use of the `Modelica.Blocks.Logical.GreaterThreshold` block set to > 0 could be made.

simplified, i.e., simple electrical models are used to determine the consumed power. The electric heating coil has a constant efficiency of 0.9, and the cooling coil operates at a constant coefficient of performance (COP) of 3.0.

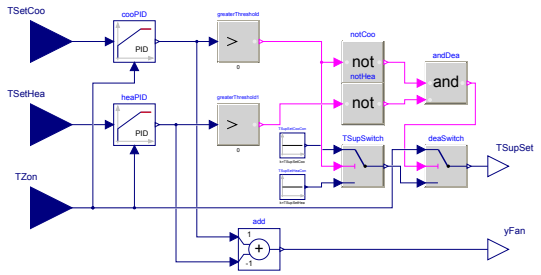


Figure 5. Thermostat Model labeled (b) in Figure 3

Finally, consider the component labeled (d) in Figure 3, this is the building model, which is expanded in Figure 6. The modeling hierarchy is shown for three layers. In the layer labeled (A), it is shown how the GM model is interfaced, and certain variables are scaled to match the model's units. In layer (B), further refactoring and computations are performed, mainly the interfacing with a weather data block with the GM and the computation of the total radiant, convective, and latent heat gains from the prediction of the GM model. Finally, layer (C) the underlying model refactored from the `Buildings` library. Although the specific case shown here corresponds to `Case900FF` of the ASHRAE BESTEST validation models (ASHRAE 2007), the refactoring provides an object-oriented hierarchy that allows one to easily modify other models by matching the interfaces for radiant, convective, and latent heat flow (which are constant in the original model), and move the weather data block to layer (B) to adapt weather conditions based on input from the GM.

The building represents a single zone with a window on the south wall and a constant infiltration mass flow rate. For the examples considered here, there are two variations of construction, the light weighted `Case600FFF` and the heavy weighted `Case900FFF`. The exterior walls and roof of `Case600FFF` and 900 are, respectively, plaster board with fiberglass insulation and concrete block with foam insulation. The floor of `Case600FFF` is timber construction and the floor of `Case900FFF` is concrete slab.

4.1.2 Measurement Data for GM Training

To train generative models, we use measurement data collected from SUSTIE, a cutting edge net zero-energy commercial office building located in Japan[5]. The name SUSTIE combines the words "Sustainability" and "Energy" and the building is designed to investigate and demonstrate technologies that can lead to energy savings and worker health and comfort. The four-story SUSTIE building has a total floor area of approximately 6456 m^2 which includes nine office spaces (experimental rooms) regularly used by around 260 office workers, an open atrium area, a cafeteria and a gym.

The building management system at SUSTIE gathers data on electrical energy usage, weather conditions, indoor environmental parameters, occupancy levels, and equipment operations to monitor and manage energy consumption and comfort throughout the building's operations. The electrical energy consumption is measured separately for different types of equipment (air conditioning, ventilation, lighting, hot water supply, and elevators) and for each room. The occupancy, i.e. the number of people in each room, is counted by the access control system using card readers installed in each area. This constitutes hundreds of sensing instruments installed throughout the building measuring more than 2,500 unique data signals throughout the year, 24 hours a day, with a sampling rate of 1 minute.

In this work a data set collected at SUSTIE over 20 consecutive months from January 2021 to August 2022 is used for training the GM's used in the examples below. For more information on the steps required for pre- and post-processing of data and GM training, see (Chakrabarty, Vanfretti, et al. 2024).

4.2 Illustrative Example

Let us now present some simulation results obtained by simulating the model shown in Figure 3.

First, we present the GM predictions in Figure 7, which correspond to the signals from `genModel`, (a) in Figure 3 that are fed to the building model (d) in Figure 3. These figures display the mean of the distribution from the measured data (*nom*) as well as a realization from the generative model (GM). The output of the GM corresponds to several variables (e.g., occupancy, equipment power, etc.) that influence the radiant, convective, and latent heat gains of the room shown, which is represented by a light blue square with a gray edge in Figure 6 (C). As can be seen in Figure 7, the GM model provides a time series that reflects what is expected of such types of building operations. For example, in Figure 7 (a) the occupancy increases in the morning and falls to zero during the day, while in Figure 7 (b) the power consumed by the equipment is highest in the morning and drops to a minimum at night. This illustrates the expressiveness of these generative models, as multiple simulations can be used to characterize the effect of the uncertainty in these input quantities.

The influence of the GM model upon the radiant, convective, and latent heat gains is shown in Figure 8. Observing the flow of radiant heat in Figure 8 (a) while at the same time observing both the power of the equipment in Figure 7 (b) and the global horizontal radiation in (f), it can be observed that during the beginning of the day both variables influence the radiant heat. Meanwhile, the convective heat flow in Figure 8 (b) is not as drastically affected at the beginning of each day by these variables. Furthermore, it should be noted that the latent heat flow in

[5]See `https://www.mitsubishielectric.com/en/about/rd/sustie/index.html`.

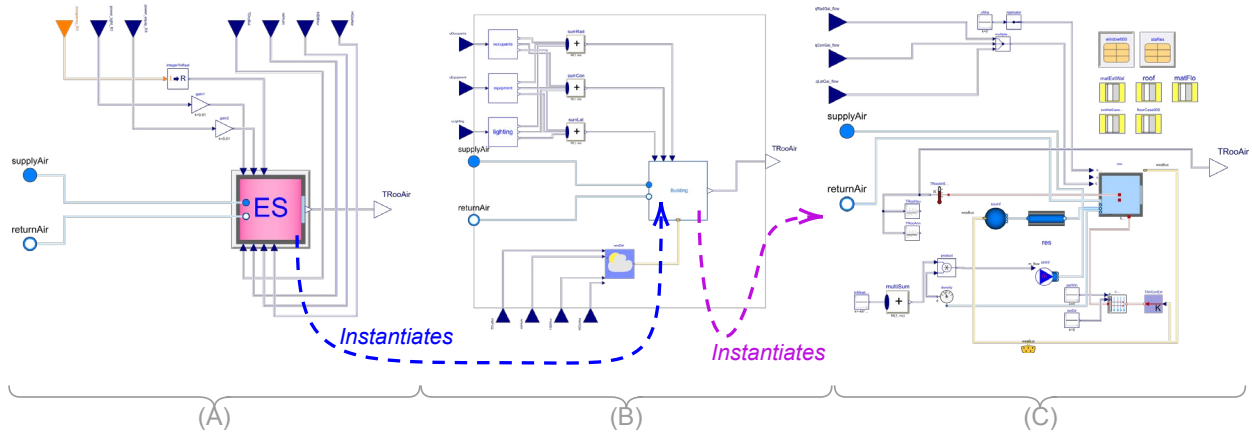


Figure 6. Hierarcichal Layers of the Single-Zone Building-Model labeled (d) in Figure 3. Observe that layer (C) is a refactored model from the `Buildings` library, corresponding to `Case900FF` of the ASHRAE BESTEST validation models (ASHRAE 2007).

Figure 8(c) is dominated by occupancy. It is worth noting that in the case of the latent heat flow, not having a good estimate such as those from the GM can lead to a substantial underestimation of latent heat, as shown by the case where the variables are assumed to be some 'nominal' heat loads obtained from a nominal schedule of energy use in the zone.

The GM's output also influences the heat and mass balance in the moist air of the room. Within the room in Figure 6 (C), the component `MixedAirHeatMassBalance` determines the heat and mass balance of moist air (M., W. Zuo, and T. Nouidui 2011), as can be seen in the sensed air temperature and flow rate of water added to the air using the `MixingVolumeMoistAir` component from the `Buildings` library. The resulting effect of the difference between the nominal inputs and a realization from the generative model can can be observed in Figure 9(a)-(c); the GM output allows a better estimation of the resulting heat flow in the room (see (a)) and a more realistic temperature estimate (see (b)), which can serve in the sizing of the HVAC system and/or improving its control. It should be noted that, although small, it is also possible to quantify the rate of extraction of water from moist air in Figure 9 (c), which would otherwise be underestimated when using nominal values instead of those of the GM.

Finally, to maintain the room temperature shown in Figure 9 (b) within the specified set points, the FCU and the thermostat in Figures 4 and 5 must cool the air. The resulting set point provided by the thermostat to regulate the cooling is shown in Figure 10 (a), with the air flow from the FCU shown in Figure 10 (b). From these figures, it can be seen that the impact of including the GM serves to adapt the performance of the cooling system according to the operating needs of the building. Observe in Figure 10 (a) that the new setpoints adapt to changes in the building conditions that are not prescribed by the mean of the experimental measurements, allowing the user to quantify the uncertainty in the system performance related to variations in the building operation.

5 Conclusions

Bringing together physics-based buildings models with models that describe variables driven by human interactions has the potential to substantially improve the performance of existing buildings or to develop better informed building designs, particularly when considering heating and cooling requirements that impact HVAC systems. To explore this potential, this paper has presented the requirements and a prototype implementation of the integration of machine learning generative models and physics-based building models. Once the generative models were trained, they were linked to a building model by exploiting the external function interfaces for C defined in the Modelica language specification. This enabled the reuse of Modelica building models from the `Buildings` library, while simultaneously leveraging real-world occupancy, power consumption, and other data for building energy simulations, as demonstrated by the provided examples.

Although the prototype implementation proposed here has proven beneficial in the development of novel building control performance analysis techniques (Chakrabarty, Vanfretti, et al. 2024), it has several other application domains. For example, it can be used similarly to characterize load patterns and perform power system control performance evaluations (Bombois and Vanfretti 2024). There is also a great deal of room for improvement, for example, to be able to use multiple and different types of generative model architectures, which will be subject to future work.

Acknowledgements

Part of the work reported in this article was carried out by the first author during his sabbatical at Mitsubishi Electric Research Laboratories (MERL). The author thanks MERL for the opportunity and support.

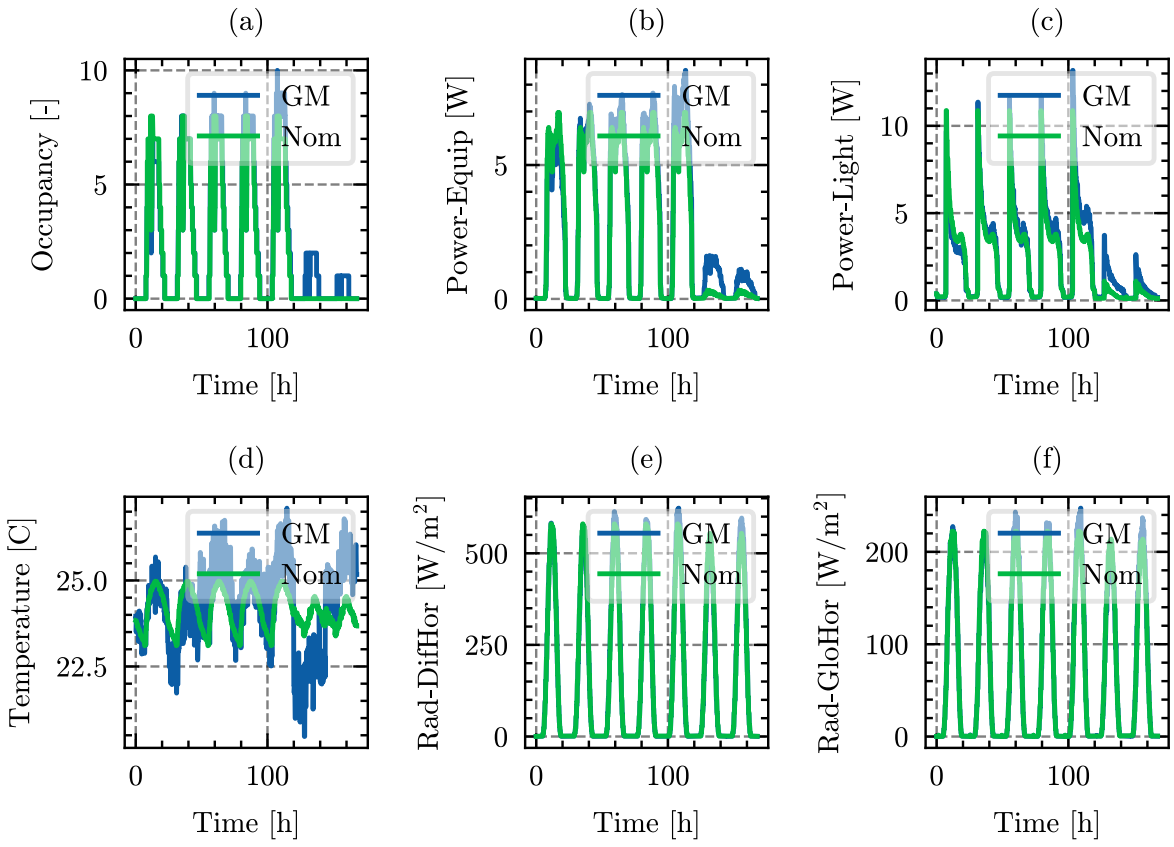


Figure 7. Outputs of the GM model, (a) in Figure 3, for a 7-day period. (a) Occupancy. (b, c) Power consumed by equipment and lighting. (d) Outdoor temperature. (e,f) Solar radiation, scattered and global.

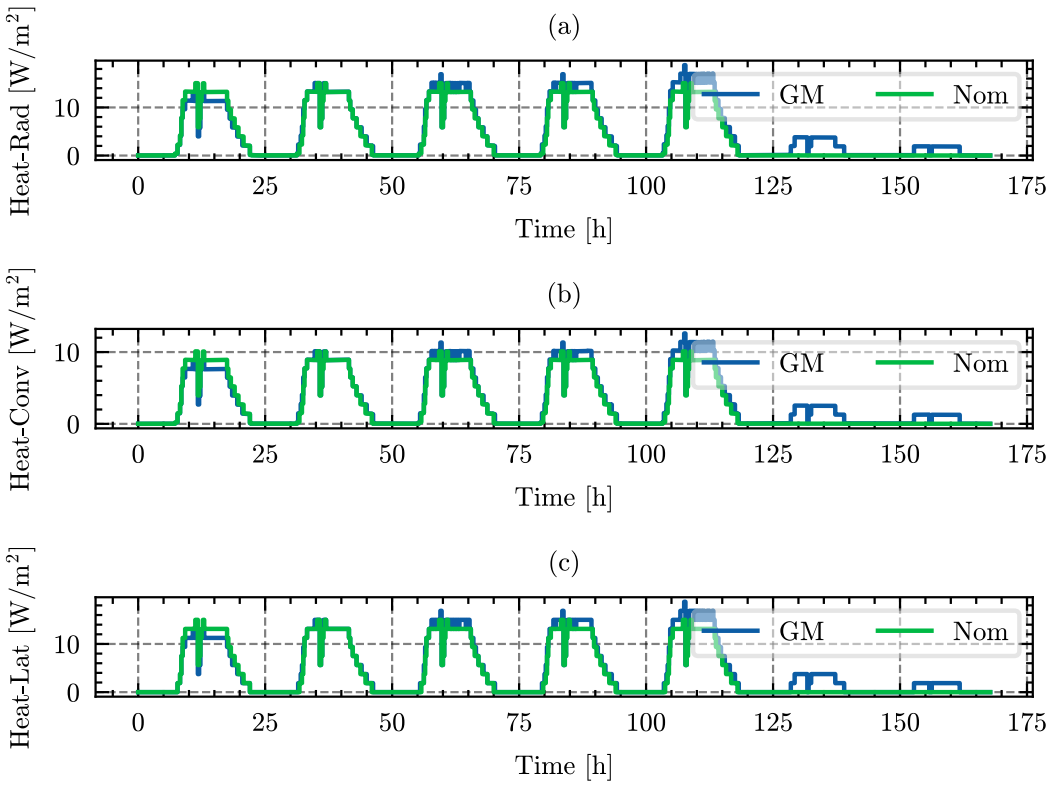


Figure 8. (a) Radiant, (b) convective and (c) latent Heat Flow resulting from the GM and a Nominal signal.

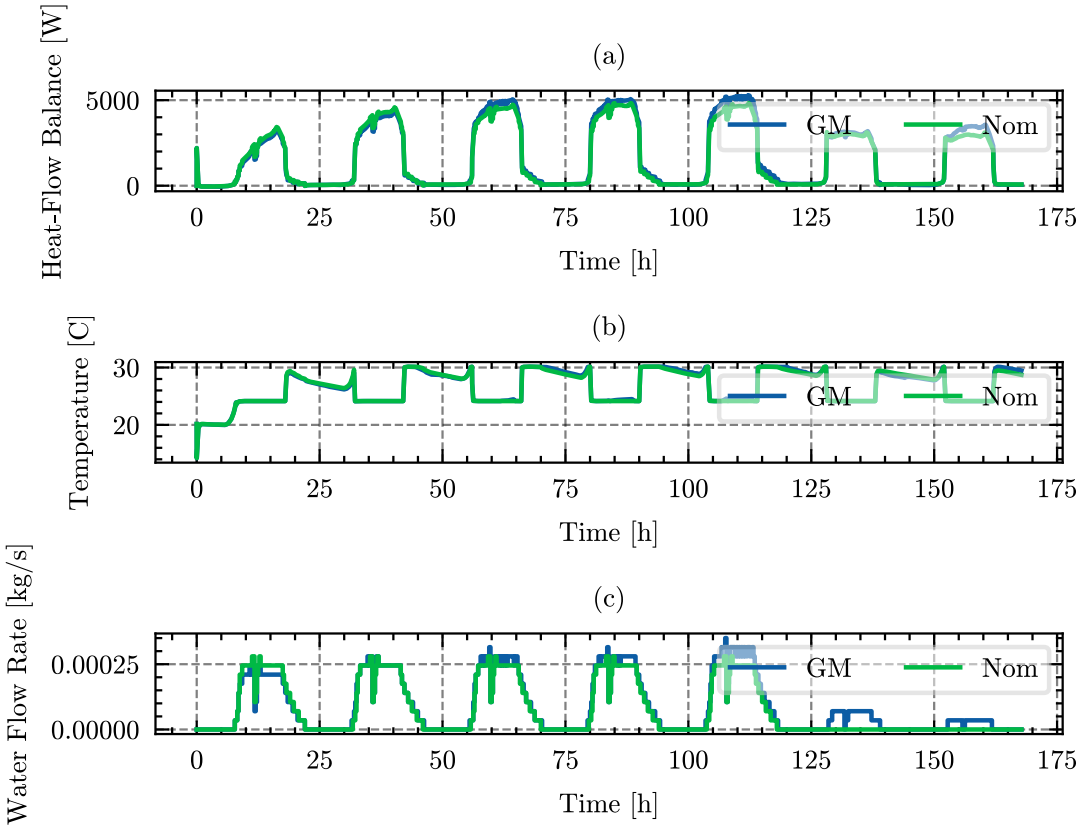


Figure 9. Mixed air conditions in the room in Figure 6 (C). (a) Heat flow balance, (b) temperature and (c) water flow rate.

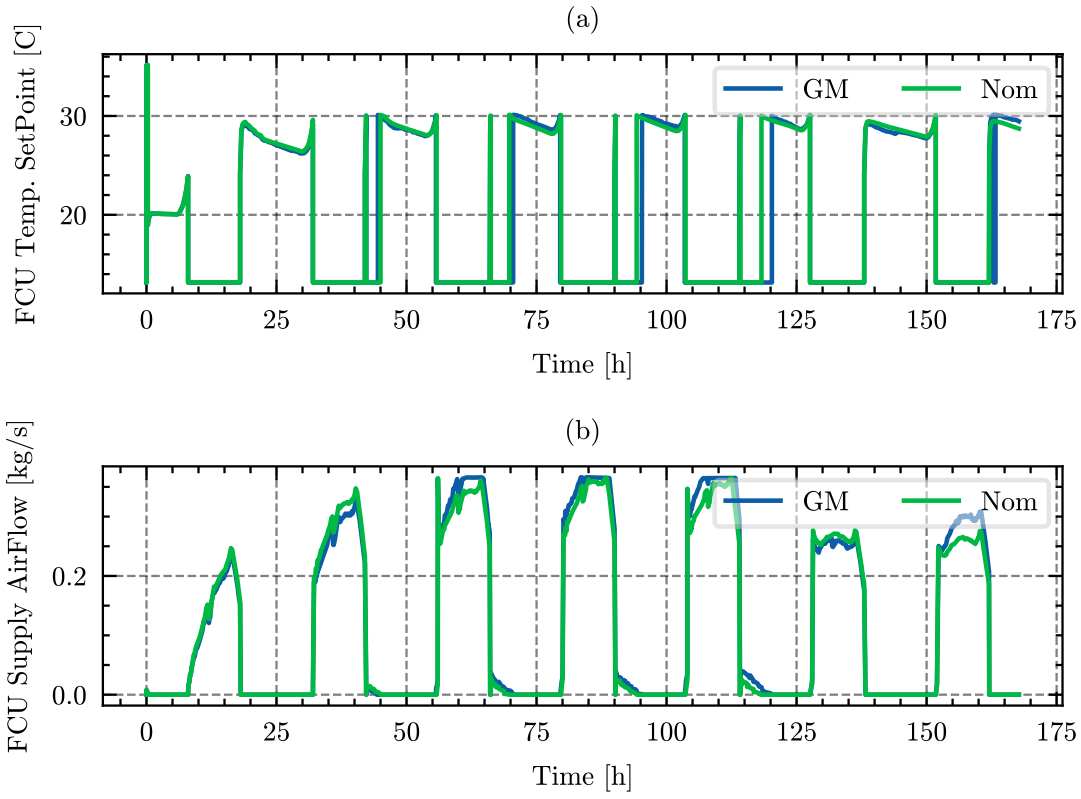


Figure 10. (a) Thermostat `TSupSet` output (see Figure 5) and (b) Supply air output measured from `senSupFlo` (see Figure 4)

References

ASHRAE (2007). *140: Standard Method of Test for the Evaluation of Building Energy Analysis Computer Program*. Tech. rep. ASHRAE.

Bombois, Xavier and Luigi Vanfretti (2024). "Performance monitoring and redesign of power system stabilizers based on system identification techniques". In: *Sustainable Energy, Grids and Networks* 38, p. 101278. ISSN: 2352-4677. DOI: https://doi.org/10.1016/j.segan.2024.101278.

Bruck, D., H. Elmqvist, H. Olsson, et al. (2002). "Dymola for Multi-Engineering Modeling and Simulation". In: *2nd International Modelica Conference*, pp. 55–1 — 55–8. URL: https://modelica.org/events/Conference2002/papers/p07_Brueck.pdf.

Chakrabarty, Ankush, Emilio Maddalena, Hongtao Qiao, et al. (2021). "Scalable Bayesian optimization for model calibration: Case study on coupled building and HVAC dynamics". In: *Energy and Buildings* 253, p. 111460.

Chakrabarty, Ankush, Luigi Vanfretti, et al. (2024). "Assessing Building Control Performance using Physics-Based Simulation Models and Deep Generative Networks". In: *8th IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, pp. 1–8.

Chen, Zhenghua and Chaoyang Jiang (2018). "Building occupancy modeling using generative adversarial network". In: *Energy and Buildings* 174, pp. 372–379.

Codeca, F. and F. Casella (2006). "Neural Network Library in Modelica". In: *5th International Modelica Conference*, pp. 549–557. URL: https://modelica.org/events/modelica2006/Proceedings/sessions/Session5c3.pdf.

Das, Hari Prasanna, Ryan Tran, Japjot Singh, et al. (2022). "Conditional synthetic data generation for robust machine learning applications with limited pandemic data". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 11, pp. 11792–11800.

Dassault Systemes AB (2023-09). *Dymola — Dynamic Modeling Laboratory — Full User Manual*. Lund, Sweden.

M., Wetter, W. Zuo, and T. Nouidui (2011). "Modeling of Heat Transfer in Rooms in the Modelica "Buildings" Library". In: *12th Conference of International Building Performance Simulation Association*, pp. 1096–1103.

Mirakhorli, Amin and Bing Dong (2016). "Occupancy behavior based model predictive control for building indoor climate—A critical review". In: *Energy and Buildings* 129, pp. 499–513.

Modelica Association (2017). "Modelica Specification, Version 3.4". In: URL: www.modelica.org.

Modelica Association (2019). "Functional Mockup Interface for Model Exchange and Co-Simulation, Version 2.0.1". In: URL: www.fmi-standard.org.

Salatiello, Alessandro, Ye Wang, Gordon Wichern, et al. (2023). "Synthesizing Building Operation Data with Generative Models: VAEs, GANs, or Something In Between?" In: *Companion Proceedings of the 14th ACM International Conference on Future Energy Systems*, pp. 125–133.

Schweiger, Gerald, Cláudio Gomes, Georg Engel, et al. (2019). "An empirical survey on co-simulation: Promising standards, challenges and research needs". In: *Simulation modelling practice and theory* 95, pp. 148–163.

Stoffel, Phillip, Laura Maier, Alexander Kümpel, et al. (2023). "Evaluation of advanced control strategies for building energy systems". In: *Energy and Buildings* 280, p. 112709.

The MathWorks (n.d.). *Export Network as FMU*. Accessed: Feb. 2, 2024. Available since v. 2023b. URL: https://www.mathworks.com/help/deeplearning/ug/export-network-to-fmu.html.

Wetter, Michael, Paul Ehrlich, Antoine Gautier, et al. (2022). "OpenBuildingControl: Digitizing the control delivery from building energy modeling to specification, implementation and formal verification". In: *Energy* 238, p. 121501.

Wetter, Michael, Wangda Zuo, Thierry S Nouidui, et al. (2014). "Modelica buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Wolfram Research (n.d.). *NeuralNet: Library that provides support for the use of neural networks in modeling*. Accessed: Feb. 2, 2024. URL: https://reference.wolfram.com/system-modeler/libraries/SystemModelerExtras/SystemModelerExtras.NeuralNet.html.

XRG Simulation GmbH (n.d.). *SmartInt: Simple Modelica Artificial Intelligence Interface*. Accessed: Feb. 2, 2024. URL: https://github.com/xrg-simulation/SMArtIInt.

Ye, Yunyang, Matthew Strong, Yingli Lou, et al. (2022). "Evaluating performance of different generative adversarial networks for large-scale building power demand prediction". In: *Energy and Buildings* 269, p. 112247.

Zhan, Sicheng, Gordon Wichern, Christopher Laughman, et al. (2022). "Calibrating building simulation models using multi-source datasets and meta-learned Bayesian optimization". In: *Energy and Buildings* 270, p. 112278.

Pipeline-based Automated Integration and Delivery Testing of Simulation Assets with FMI/SSP in a Railway Digital Twin

Ozan Kugu[1] Shiyang Zhou[1] Stefan H. Reiterer[2] Mario Schwaiger[2] Lukas Wurth[1] Manfred Grafinger[1]

[1]Institute of Engineering Design and Product Development, TU Wien, 1060 Vienna, Austria, ozan.kugu@tuwien.ac.at
[2]Virtual Vehicle Research GmbH, 8010 Graz, Austria, stefan.reiterer@v2c2.at

Abstract

Railway infrastructure systems have recently been enhanced through the use of the digital twin (DT) concept, enabling visualization and control in a virtual environment while effectively mitigating life cycle costs. This work provides insights into the development and operations (DevOps) of a railway DT platform and highlights the automation and management of asset integration and processing based on the FMI and SSP interface standards through the use of the Continuous Integration / Continuous Delivery pipeline technology. This offers long-term durability, pausability, remote triggering, open-source and workflow design capabilities, and connectivity to other tools such as version control systems and code analysis tools. In this research paper, we present an anti-slip co-simulation model of a railway vehicle as a use case example to demonstrate the pipeline-oriented automation and management in combination with a version control system and code analysis tool within the platform.

Keywords: CI/CD Pipeline, DevOps, FMI, SSP, Automation and Management, Asset Integration and Processing

1 Introduction

Railways play a crucial role in modern public and freight transportation due to their cost-effectiveness, energy-efficiency and eco-friendliness. To reduce life-cycle costs, conserve energy, and streamline maintenance and monitoring for railway operators and infrastructure managers, railway infrastructure systems are being brought into the virtual world through the use of the DT concept. As an example, (Zhou et al. 2022) conceptualized a DT platform called Rail for Future (R4F), where digital assets (models and data) from different railway subsystems, including vehicles, tracks, turnouts, bridges and tunnels, can be integrated and interoperated with each other. This enables end-users to control and visually analyze the railway system.

There are challenges and limitations to overcome in railway digitalization. For instance, raw simulation assets of the subsystems cannot be run and easily managed in the DT platform due to their software tool dependence, operating system (OS) incompatibility, relatively complex model and data structure. The use of interface standards, offered by Modelica Association, shows great potential for dealing with the adaptation of the simulation assets to the platform. Some of these standards are Functional Mock-up Interface, which provides an interface between dynamic simulation models and software as a ZIP-formatted container (simulation unit called Functional Mock-up Unit (FMU)), is open-source and supported by more than 200 tools (*Functional Mock-up Interface (FMI) Standard* 2024), and System Structure & Parameterization, which is used to containerize complex simulation systems containing one or more FMUs and ideal for co-simulation use cases (*System Structure and Parameterization (SSP) Standard* 2024). This adaptation process allows the assets to be seamlessly integrated and processed in the platform. (Kugu et al. 2023) successfully demonstrated this in their work by using the FMI and SSP standards. In order to ease the asset integration and processing task, we prefer to automate it and improve its management. This is a challenging mission, that requires comprehensive knowledge and experience about the assets, the task and automation techniques to apply it to the platform. This paper demonstrates the use of the Continuous Integration / Continuous Delivery and/or Continuous Deployment (CI/CD) pipeline technology for automated integration, processing and management of the assets with the FMI and SSP standards in the R4F Platform. The technology is foreseen as an effective method for the automation, because it provides workflow design, open-source, delivery and/or deployment capabilities, remote-triggering functions, pausability (ability to stop and wait for human response), long-term durability and platform compatibility. Besides, the pipeline software tool can interoperate and communicate with other software tools such as version control systems and code analysis tools, which help track and store the version of the assets and detect potential errors, vulnerabilities and redundancies in the codes belonging to these assets. Thus, version control systems and code analysis tools are applied to the pipeline in this work, which effectively boosts the automation and management of the asset integration and processing task as a part of the DevOps practice in the platform. Moreover, there are open-source tools available for these two tech-

nologies. Considering all the features and open-source materials, mentioned above, the CI/CD pipeline technology is highly preferred for this research work.

Another issue to address concerns license management, which is necessary to obtain permissions for running simulations of the assets in the pipeline of the platform, since these assets were previously designed and configured with an appropriate solver in commercial software tools. Additionally, a thorough understanding of pipeline installation, configuration, and design is needed, which also depends on the physical domain according to (Zampetti et al. 2023)'s comprehensive study on CI/CD pipeline implementation in a DT for Cyber Physical Systems (CPS) incl. railways. Moreover, proficiency in installing, configuring version control systems and code analysis tools, and effectively interoperating the pipeline with them in the platform is essential.

In this research paper, first, in Section 2, we give insights about the benefits and limitations of railway DTs, CI/CD pipeline technology, FMI and SSP standards through different research examples. Second, in Section 3, we define the FMI- and SSP-based asset integration and processing task and its underlying goals. Then, in Section 4, we present what kind of advantages the automation and management of the task bring to stakeholders, how we apply it to the R4F Platform, how we keep the stakeholders updated through the use of the pipeline, version control system & code analysis tool methodologies and which processes we defined and directly applied to the pipeline in the platform. After that, in Section 5, we show an interesting demonstration of how we automate and manage the integration and processing of a co-simulation model consisting of a multibody simulation (MBS) model of a railway vehicle, and a Proportional-Integral-Derivative (PID) controller model for anti-slip traction and vehicle speed control of the vehicle. In Section 6, we discuss the simulation comparison results of the use case, and point out limitations and challenges we encountered while integrating, processing the assets, then automating and managing them in the pipeline. Finally, in Section 7, we briefly mention the conclusion of this work and outline future work.

2 Related Work

2.1 Railway Digital Twin

In recent years, railway DTs have increasingly been designed and developed by numerous researchers and engineers to enhance operational, monitoring and maintenance tasks within the virtual railway environment. For instance, (Zhang et al. 2021) presented a DT-assisted approach for fault diagnosis of railway point machines used to operate turnouts. They emphasized the significance of DTs in enhancing fault diagnosis processes, thereby improving the reliability and efficiency of railway operations. In 2022, (Hamarat, Papaelias, and Kaewunruen 2022) introduced a Peridynamics-based DT approach, which could predict potential fatigue damage in railway turnout crossings by integrating real-time data and simulations, facilitating proactive maintenance and improved safety. This enhancement contributed to assessment and management of railway infrastructure. Additionally, (Kaewunruen et al. 2023) proposed employing DTs for managing railway bridge maintenance, where they monitor and analyze the structural integrity of railway bridges in real time. Their study highlighted the role of DTs in improving decision-making processes related to the maintenance, thereby enhancing the overall safety and reliability of railway infrastructure systems. As mentioned in Section 1, (Zhou et al. 2022) conceptualized a model-based DT platform called R4F, capable of simulation, visualization, and predictive analytics, enabling stakeholders to optimize operations, maintenance, and resource allocation for comprehensive management of large-scale railway infrastructure systems. This advancement improved the efficiency and reliability of the system. This paper aims to ease stakeholders' aforementioned tasks by demonstratively automating and managing the integration and processing of various railway simulation assets within the platform.

2.2 CI/CD Pipeline for Digital Twins

Many researchers and software engineers prefer CI/CD pipelines to automate and manage integration, simulation, validation, delivery and deployment processes in a DT of a physical system as DevOps practices. This approach facilitates easier analysis, monitoring, and evaluation of the system. For example, (Hugues et al. 2020) proposed the TwinOps process, which is a combination of DevOps, DTs and model-based engineering, and used it for automated code generation, condition monitoring and data analysis of CPSs. (Villa et al. 2024) used the CI/CD pipeline technology to reproduce protocol stacks (e.g. cellular, WiFi) in both physical and digital environments in real time, which helps researchers to efficiently and automatically test the protocols in a conceptual DT for large-scale wireless networking. (Barbie, Hasselbring, and Hansen 2023) enhanced the automated testing of their DT prototype for smart farming applications through the use of the pipeline technology. They noted relatively high cost and time consumption of the hardware used for the applications, making simulations a preferable choice over the hardware for gaining virtual insights into the physical system via the CI/CD pipeline. Consequently, we aimed to work with multiple railway simulation assets, intending to automatically integrate and process them within our CI/CD pipeline for this work.

Another crucial aspect to consider is the intercommunication of the pipeline with other tools, which helped us to further improve the management of our automated asset integration and processing task occurring in the pipeline. For instance, (Kiran et al. 2021) suggested to work with code analysis tools in their CI/CD pipeline for more reliable and secure software development life cycle and DevOps. Similarly, (Zampetti et al. 2017) did a relatively extensive comparative study, where they investigated the us-

age of several static code analysis tools in a CI pipeline for static analysis of many different open-source software applications. They also gave many insights about how effectively the static code analysis tools should be used with the CI pipeline to detect bugs, errors and warnings in the application examples, which is important for us to be aware of possible failures, redundancies and vulnerabilities in our asset applications. Besides, (Sethi 2020) proposed to apply version control system to their CI/CD pipeline for business intelligence solutions in order developers to keep tracking and saving source code changes collaboratively. Based on this, we found the version control system very promising for tracking and storing our asset applications with a version control system tool directly connected to our CI/CD pipeline for this work.

2.3 FMI and SSP Standards for Railways

The FMI and SSP standards are preferred to be used in many sectors including the railway sector, because these strongly assist researchers to integrate and manage different railway simulation models in a virtual environment by providing more tool-independence, file-portability, co-simulation capability and less data complexity in spite of particular limitations noted by railway experts.

(Pieper and Obermaisser 2018) introduced a distributed co-simulation approach for conducting software-in-the-loop tests of networked railway systems. In this approach, various subsystems of the railway network could be simulated independently and in parallel by leveraging FMI. They also highlighted the potential of FMI in enabling collaborative and scalable simulation of the systems, which is crucial for ensuring their reliability and safety in operations. (Hotzel Escardo et al. 2021) designed and developed a train driver behaviour model for railway co-simulations, demonstrating how train driver behavior models can be seamlessly integrated with other simulation components, such as infrastructure and rolling stock models by utilizing FMI. This approach allows for comprehensive simulations that capture the interactions between different elements of railway systems, ultimately enhancing the understanding of system dynamics, supporting decision-making processes in railway operations, and planning. Besides, (Zhou et al. 2023) proposed to use FMI to seamlessly integrate their machine learning based surrogate model in the R4F Platform. This enables tool-independency and interoperability with other future models. (Golightly et al. 2022) noticed several FMI-compliant simulation tools (e.g. MATLAB/Simulink), used for rail applications and thus showing great potential of the FMI standard for the rail sector, while they studied the practicability of the multi-modelling approach for rail decarbonisation systems. On the other hand, they listed a couple of limitations of the FMI for railways such as lack of clear presentation, data incompatibilities and intellectual property (IP) issues related to railway simulation models, which are surely to be considered while using FMI for railway applications as well. (Hällqvist et al. 2021) discussed the utilization of the SSP standard to achieve engineering domain interoperability, particularly focusing on its application within railway systems. The study highlights how SSP facilitates interoperability between various engineering domains, including railway systems, by providing a common standard for describing system components and their interactions. By adopting SSP, engineers can enhance collaboration, optimize model integration processes, and increase the efficiency of complex railway system development and analysis. Finally, (Kugu et al. 2023) proposed to use both FMI and SSP to integrate and simulate different railway simulation models such as an MBS model of a railway vehicle, and residual life time calculation model of a railway steel bridge in the R4F Platform. They also pointed out the high potential of these standards for the railway sector, which gave us enormous inspiration to use the FMI and SSP technologies for our asset integration and processing task in the CI/CD pipeline on the platform in this work.

3 Asset Integration and Processing

The asset integration and processing task plays a significant role in enabling various railway use cases to operate within the R4F Platform. This task comprises two primary components: Asset Integration and Asset Processing.

The Asset Integration in this work means direct adaptation of the simulation assets consisting of model and data to the platform through the use of interface technologies such as FMI and SSP. Before the Asset Integration part, these assets were manually designed and processed in a Graphical User Interface (GUI) - supported software tool (e.g. MATLAB, Simpack,...), which is very handy to create, configure, optimize and simulate a physical system. The system can be a railway vehicle as an example from the railway sector. Of course, these assets need to be further prepared to properly use it with the interfaces in the platform. In this preparation, input parameters, input and output channels are defined and assigned to the simulation assets. This is necessary to address the right inputs and outputs for the asset simulation, so that the input parameterization and output generation work in a right manner during the asset simulation executed by the pipeline in the platform. After that, these assets are packed into FMU and lastly into SSP, where the FMUs are connected to each other resp. co-simulated. This helps to achieve the assets as a container in ZIP format including the entire asset description (model metadata, parameters, connections, connectors). This container can technically be analyzed, simulated in the platform, and therefore makes the assets independent from their GUI-based default tool as found out in this work.

Asset Processing occurs subsequent to Asset Integration. In this case, the assets, integrated with FMI and SSP, are tested, optimized and then released to be completely sure that these function in a right manner. For this, necessary software tools, libraries, packages and licenses are

pre-installed on a computer as first step. After that, codes of the assets are manually analyzed to find out errors and then fix bugs. After the code analysis, these assets are simulated and finally their outputs are generated by executing a simulation code script on the computer (see the model simulation approach of (Kugu et al. 2023)). Prior to delivery, all aspects related to the simulation test undergo peer review by the Asset Integrator to evaluate the test and determine whether to release the assets.

4 Automation & Management

4.1 Benefits

The asset integration and processing are automated and managed by the Asset Integrator through the use of the pipeline, version control system and code analysis tool in the R4F Platform as previously mentioned in Section 1. The reason for this is that it brings significant advantages as follows:

- **Time-efficiency** in the asset integration and processing through the automation,
- **Money-saving**, because open-source tools are implemented for the automation and management work,
- Stakeholders need **less prior knowledge** about the asset integration and processing through the Asset Integrator's great contribution to the automation and management,
- **Long-term durable, pausable and workflow-based** asset integration and processing through the pipeline technology,
- **Better quality control** through analysis and validation processes,
- **Better control of the tracking and storage** of the simulation assets between different stakeholders by using the version control system, therefore **easier collaborative work** between them.

4.2 Environment Overview

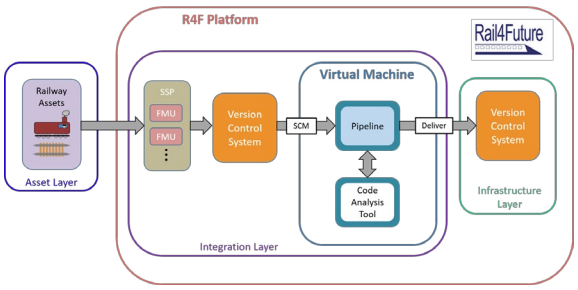


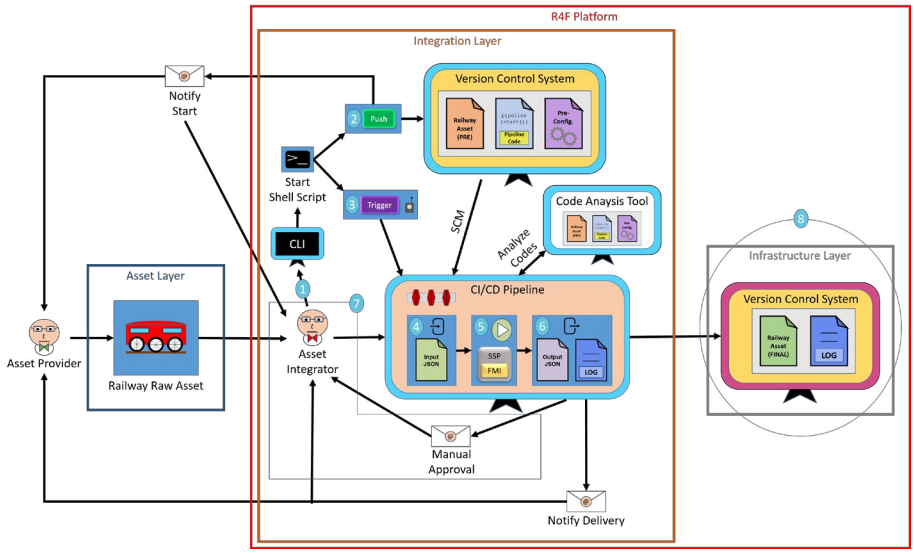
Figure 1. Simplistic Landscape of the CI/CD Pipeline-based Automation and Management of the Asset Integration and Processing in the R4F Platform.

Figure 1 provides an overview about how, in what kind of environment and among which layers of the R4F Platform landscape (as outlined in (Zhou et al. 2022)) the asset integration and processing task is automated and managed by using the CI/CD pipeline technology. As first step, the railway raw assets, which are provided from the asset layer into the integration layer of the platform, are adapted to the platform through the use of the FMI and SSP within the scope of the Asset Integration part. Then, as first of the Asset Processing part, these integrated assets are stored and tracked in the Asset Integrator's version control system, which is directly connected to a pipeline by using a Source Code Management (SCM) plugin in order the pipeline to keep and track these assets in its own server as well. Of course, the pipeline should be comparable with the ones in the R4F Platform function layer, where pipelines for particular functions (e.g., predictive maintenance) are used, because all the assets must work in the function layer's pipelines to be able to visualize and control these assets in the visualization layer of the platform. Thus, the Asset Integrator initiates a virtual machine (VM) with Linux OS and then installs a software tool where they created the pipeline. Besides, the pipeline is directly connected to a code analysis tool installed in the same VM, so that the tool extracts the codes of the assets from the pipeline, and then publishes the code analysis results to the Asset Integrator. In these results, they can detect potential errors, redundancies, bugs and vulnerabilities, which helps them to further improve the code quality of the assets in advance. Finally, the refined assets are delivered to the version control system of the R4F Platform infrastructure layer, where the Asset Integrator conducts simulations and oversees function layer pipelines through pipeline execution, subject to manual approval.

4.3 Followed Approach

Figure 2 reveals the exact methodology to realize the automation and management of the asset integration and processing task, where the Asset Integrator plays their main part, in the pipeline. First, they get the raw prototypical simulation assets from the Asset Provider as usual. After manually preparing these assets as mentioned in Section 3, the Asset Integrator uses their Command Line Interface (CLI) tool, which is very handy to execute commands without any GUI in background. In the CLI, they start a shell script, which is a text file and contains a sequence of commands to be executed for process automation in the CLI. As first step, this script pushes all the assets (pre-asset), configuration files (pre-configs), including necessary software libraries and packages for the asset simulation, and a pipeline code, defining the workflow of the pipeline, to the integration layer's version control system software repository by using the Git command (see *Git Documentation* (2024)). Besides, the shell script pushes notification to all the stakeholders via email by using an open-source mail transfer agent software to inform them about the pipeline execution start. One notification email is sent to the Asset Integrator and includes four links with corresponding port numbers belonging to the pipeline, code analysis tool, version control system and pipeline console output, helping them to directly monitor and handle the whole progress of the asset integration

10.3384/ECP207189 DOI



Main Procedure Followed	
1	Start the Shell Script
2	Push Pre-Assets and Notification
3	Remotely Trigger Pipeline
4	Provide Input for Simulation
5	Start the FMI-based SSP Simulation
6	Generate Results
7	Manually Approve the Delivery
8	Deliver Asset

Figure 2. Overview of the Automation and Management Environment.

and processing. Another email is redirected to the Asset Provider, and includes only the pipeline start message, not the four links due to the IP protection of the pipeline automation methodology. After the pushing step, the shell script triggers the pipeline remotely through the use of the pipeline server's token so that the pipeline starts automatically and extracts all the pre-asset, pre-configs and pipeline code through the SCM from the Asset Integrator's version control system. After that, the principal input parameterization, asset simulation and its output generation occur one after the other in the pipeline. In the meantime, all the codes related to the pre-asset are automatically reviewed by the Asset Integrator's code analysis tool. Then, the Asset Integrator checks the code analysis results, pipeline console outputs and simulation results, generated as curves and output files for data analysis and validation purposes, after getting the Manual Approval message with the pipeline link as an email automatically sent from the pipeline. It should be noted that the manual checking process can be fully automated in future as (Reiterer, Schiffer, and Schwaiger 2023) did Key Performance Indicator evaluation and quality check by using post-processing tools in their work. After they finish to adapt the simulation of the FMI- and SSP-standardized asset to the pipeline, they decide to deliver the asset as final asset, and its belonging log file (pipeline console output) directly to the infrastructure layer's version control system through another push command execution by the pipeline like the previous one in the shell script. If the pushing succeeds, a Delivery message is automatically sent to all the stakeholders via email in order to inform them about the asset delivery.

4.4 Designed Workflow of the Pipeline

In this subsection, the workflow, designed and applied to the pipeline through the use of the pipeline code, is further concretized and described by defining main processes and their sub-processes. First, we automatically integrate our asset application into the R4F Platform through the use of the version control system, FMI and a couple of Linux commands in the pipeline within the Build process in order to be able to test the application there. After the Build, in the Test process, we do semi-automated testing and validation of the whole application by using the FMI, SSP, code analysis tool and Linux commands in the pipeline for quality assurance of the asset simulation in the platform. If the simulation shows relatively high resilience, code quality and result consistency, the Asset Integrator decides to deliver the final asset to the infrastructure layer's version control system software repository, which happens in the Deliver process. Otherwise, the asset application requires further improvement and development, leading to the entire automated asset integration and processing starting again from the Build process.

4.4.1 Build

1) Checkout SCM: Connection of the Asset Integrator's version control system to the CI/CD pipeline through SCM.
2) Update Software: Update necessary software packages, libraries and tools for the asset simulation in the pipeline.
3) Build FMU: Automated packaging railway simulation assets into the FMUs.

4.4.2 Test

4) Analyze Codes: Scan source codes related to the assets with the code analysis tool.
5) Validate FMU: Check, if the FMUs work properly.
6) Gather Info FMU: Extract all metadata from the FMUs to display these data in the pipeline console output.
7) Update SSP: Load the FMUs into an example SSP file.
8) Simulate SSP: Test the FMI-based SSP co-simulation of the assets.

9) Generate Results: Publish results in different formats for data analysis and result validation.
10) Manual Approval: Notify the Asset Integrator about the completed asset simulation in order them to approve the asset delivery.

4.4.3 Deliver

11) Deliver Asset: Release the whole asset application to the infrastructure layer's version control system and then inform all the stakeholders about it as confirmation.

5 Use Case: Anti-Slip Traction & Vehicle Speed Control System

In the following use case, we established a traction control alongside a vehicle speed control for an existing MBS train model, having two bogies with four wheel sets (two wheel sets per bogie), based on the Manchester Benchmark (Iwnicki 1998), designed in the commercial software tool Simpack from Dassault Systèmes, and provided by Virtual Vehicle Research GmbH. This control is able to keep the longitudinal slip of each train wheelset on a constant user input value, while simultaneously controlling the vehicle speed based on a linear function with user input variables. The purpose of this use case is to enhance the extent of usage of the already existing train model by providing the option of simulating scenarios that are closer related to actual situations, such as accelerating and braking the vehicle for arrival and departure at different train stations.

For the anti-slip traction control, the MBS model was co-simulated with a control model based on a PID controller and designed in MATLAB/Simulink. It was established through the use of the SIMAT method, which serves as a co-simulation interface between the Simpack server and MATLAB client. This interface was implemented as a SIMAT block, representing the MBS model, and directly connected its input and output channels to the input and output ports of the PID controller model in Simulink. The controller model uses the current vehicle speed and four wheel speeds of the four wheel sets in the two respective bogies as inputs. By calculating the current slip and the deviation to the desired slip, which is processed by the PID controller, the controller model generates an output signal. This output signal then results in an additional torque on the axis of the respective wheel sets, thus braking or accelerating the wheels. The vehicle speed control was implemented as a polynomial 1st order MATLAB function, with acceleration input variable as the slope parameter and the initial vehicle speed input variable as the offset parameter. The eventual outputs of this control were the vehicle position, speed and acceleration, which were handed over to the MBS vehicle model as constraints for its speed control. (for more information about the physical structure and implementation of the co-simulation model see (Zhou et al. 2024))

To integrate this model into the R4F Platform we used an open-source library for Simulink called FMI Kit for Simulink (see *GitHub - FMI Kit for Simulink* (2024)). This facilitated the packaging of the PID-based Simulink controller model into an FMU file, which is directly executable in the platform. After defining the input and

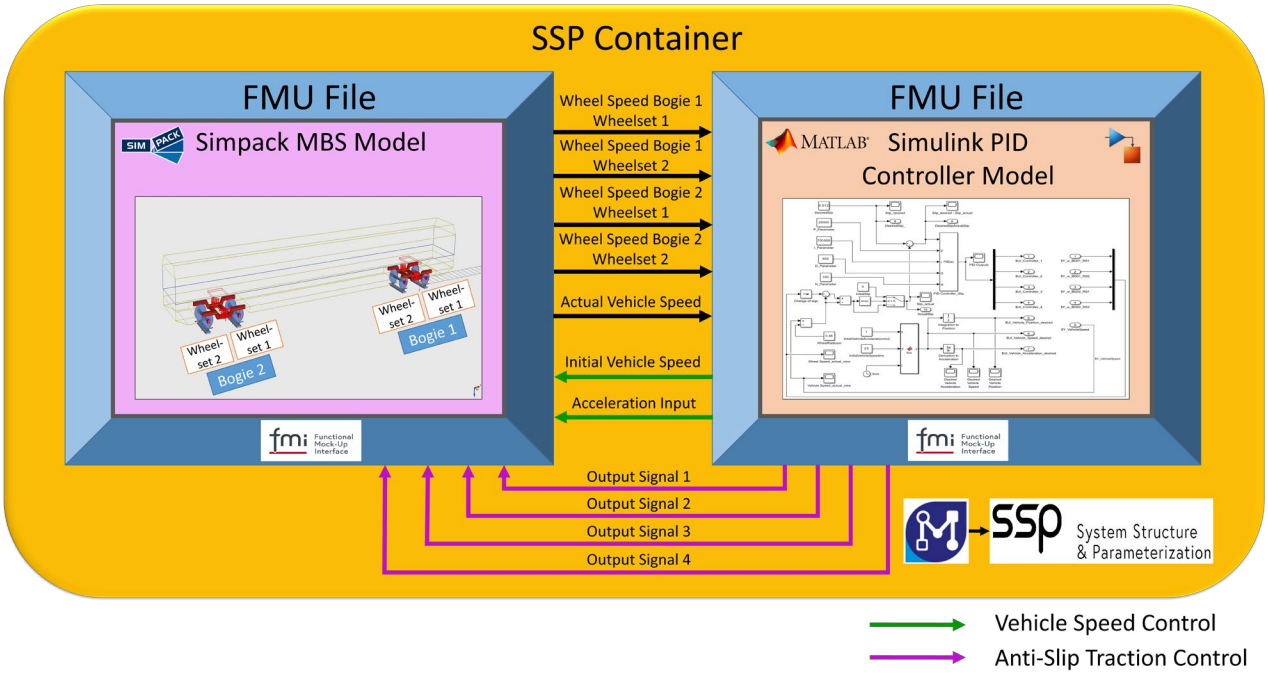


Figure 3. An Overview of the Physical Structure of the Anti-Slip Co-Simulation Model Adapted to the R4F Platform with FMI/SSP. (see (Zhou et al. 2024))

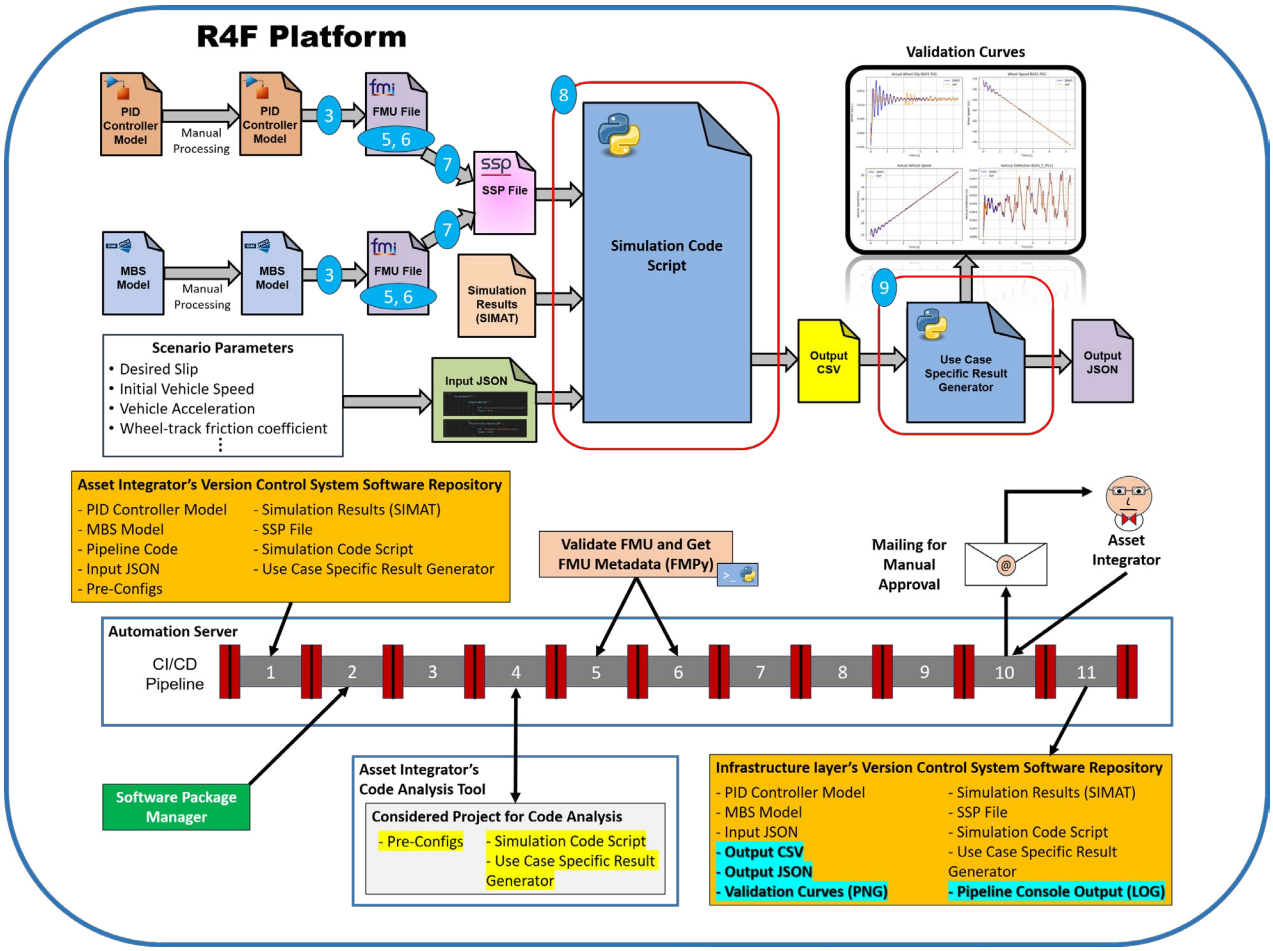


Figure 4. Automated Integration and Processing of the Anti-Slip Co-Simulation Model.

output channels of the MBS model in Simpack, we easily exported the model to the FMU by using Simpack itself. Then, we connected these two FMU files in Model.CONNECT, which is a co-simulation tool and provided from AVL List GmbH, to each other. Within the tool, we conducted tests to verify the co-simulation of these two models. After that, we used Model.CONNECT to pack the entire co-simulation model into an SSP example file, which contains the whole system structure and parameter description files with system metadata, connectors (input and output channels), connections and two FMU components belonging to these two models. Figure 3 shows an overview of the physical structure, input and output flow, described in the previous paragraph and occurring between the two FMU components in the SSP, of the entire co-simulation model.

Figure 4 illustrates the complete implementation of the use case in our CI/CD pipeline with the FMI, SSP standards, code analysis tool and version control system software repositories. Before the pipeline execution, first, the Simpack MBS model and Simulink PID controller model are manually processed by the Asset Integrator, by which scenario parameters (e.g., initial vehicle speed, desired slip, vehicle acceleration, etc.) given by the user are defined and configured. Besides, necessary input and output channels are created for their further interconnection and output generation. Following the manual processing, the pipeline extracts all asset files, including pre-configs, the PID controller model, MBS model, pipeline code, input JSON file, simulation results from the SIMAT, SSP example file, simulation code script, and use case specific result generator, from the Asset Integrator's repository through the SCM. Then, the pipeline updates all the software libraries and packages from the pre-configs in its server with a software package manager. For the Build FMU step, we preferred to use libraries and modules from the pre-configs, belonging to the open-source FMI Kit for Simulink, the commercial software tools Simpack and MATLAB without GUI, which helps us to overcome the OS dependency of the FMUs in the platform. In the next step, the source codes of the asset (incl. the simulation code script, use case specific code generator and some of the pre-configs) are analyzed and right after that their analysis reports are published in the code analysis tool. By using the FMPy Python package (see *GitHub - FMPy* (2024)), the two FMUs are easily validated and then their

metadata are directly extracted from themselves. After the pipeline updates the example SSP with these FMUs through Linux command executions, it directly executes the simulation code script to simulate the SSP as a black-box. For simulation purposes, we once again employ FMPy again and further developed the simulation code script, belonging to the SSP subfolder of the FMPy, for input parameterization and output generation. During the simulation testing progress, first, the input JSON, simulation results from SIMAT, and the SSP, containing the SSD system structure, SSV parameter files and FMUs, are read by the code script, then the simulation runs with a fixed step size and solver. Finally, the CSV results are generated for data analysis through the execution of the use case specific result generator code script, which generates validation curves and output JSON based on R4F standards. Penultimately, the Asset Integrator receives an email from the pipeline with its link, where they check all the console outputs, code analysis and simulation results. If satisfied with the entire asset integration and processing test, they deliver all asset files, including the SSP simulation results, and pipeline console outputs (excluding the pipeline code due to IP protection), to the version control system repository of the R4F Platform infrastructure layer.

6 Results and Discussion

In this section, first, we shortly discuss the simulation results coming from the FMI-based SSP simulation of the anti-slip traction & vehicle speed control co-simulation use case executed by the pipeline. Then, we address difficulties and challenges we faced related to the license management, pipeline configuration, automation and management of the asset integration and processing task in the pipeline on the R4F Platform.

6.1 Simulation Results

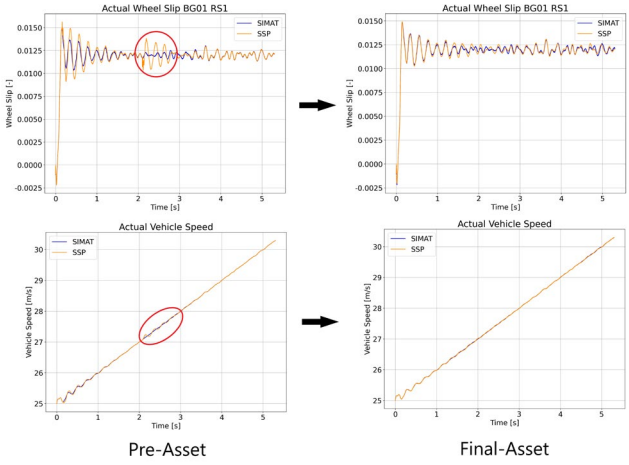


Figure 5. Result Validation and Optimization of the Anti-Slip Use Case in the CI/CD Pipeline.

In the left side of Figure 5, two different SIMAT and SSP simulation results of the pre-asset, belonging to the anti-slip use case, are displayed. These are actual values of the vehicle speed and longitudinal wheel slip of the first wheel set of the vehicle, which characterize the anti-slip use case. In the right side, the new SSP simulation results, coming from the final asset, are compared with the same SIMAT outputs after a demonstrative optimization of the SSP results by decreasing the simulation step size by a factor of ten, as previously discovered for this work. By this optimization, we aimed to reduce the unexpectedly arising little oscillations, differing from the SIMAT results and addressed with red circles in the figure. Besides that, the slip outputs give comprehensive insights about the anti-slip behavior of the model, which actually works well in the CI/CD pipeline, as realized in the figure. Lastly, it is remarkable that the constant acceleration of the vehicle functions in a right manner according to the relatively linear increase of the vehicle speed outputs.

To generate the whole picture in Figure 5, we used matplotlib, which is a Python library for visualization (see (Hunter 2007)). In general, the SSP simulation in the pipeline shows relatively consistent outputs with the SIMAT results by successfully optimizing the simulation. This also proves the success of our work with the pipeline at the end.

6.2 Challenges and Limitations

Software License Management: As mentioned in Section 1, we need software licenses, which are commercial and allow us to run the asset simulations after modeling the assets in their software. In the use case example, we needed the Simpack license to simulate the MBS model, and the MATLAB license to simulate the PID model for the SIMAT co-simulation. For the SSP co-simulation in the pipeline, we needed only the Simpack license and had to connect our VM to the Simpack license server through the use of the Virtual Private Network service by logging in with our username and password. Moreover, we used these licenses to automatically build the FMUs of these models in the pipeline without opening any GUI window. In addition, we had a Model.CONNECT license, by which we built the SSP example file of the complete anti-slip co-simulation model once as mentioned in Section 5.

Pipeline Installation and Configuration: To implement the pipeline in a right manner, we as Asset Integrator needed much know-how and experience with DevOps practices regarding to the CI/CD pipeline technology. Especially, it was very important to get to know how to install and configure the VM, software tools, packages, libraries, and then interoperate them with each other in harmony. For this work, we decided to use plugins in the pipeline server for the interoperation, code analysis tool for code analysis, version control system for asset tracking and storing, CLI for pipeline remote control, mailing, asset pushing, and basic execution commands, belonging to the software tools, packages and libraries, to apply the sub-processes defined in Subsection 4.4 to the pipeline.

Automation Testing: After providing the necessary soft-

ware licenses, installing and configuring our pipeline, we needed to test the entire asset integration and processing incl. the asset simulation as automated in the pipeline. Therefore, we always needed to do research, find out possible ways and then try them step by step to make every sub-process, implemented in the pipeline code, working in a right manner. Besides, there are use case specific limitations for the simulation test in the pipeline. For example, the anti-slip use case shows smaller oscillations by refining the simulation step size, which significantly improves the quality of the simulation results while increasing the simulation runtime on the contrary as previously discovered in this work.

7 Conclusion and Outlook

Based on our experience, the CI/CD pipeline technology is practical and interactive for automating and managing the entire FMI- and SSP-based asset integration and processing with the version control system and code analysis tool in a collaborative work environment on the R4F Platform. In addition, the FMI and SSP standards greatly facilitated the adaptive simulation of the assets in the platform, specifically in terms of tool independence, asset description and file portability. This was also demonstrated in the work of (Kugu et al. 2023). Furthermore, we successfully co-simulated multiple FMUs as one SSP for the anti-slip use case, which has shown relatively consistent results based on the validation curves in Figure 5. For this succession, we encountered challenges and restrictions related to the license management, pipeline configuration, automation and management of the asset integration and processing task in the pipeline on the platform, which should not be neglected.

In future, we plan to fully automate the asset integration and processing task by automating the model preparation and validation processes in the pipeline. In addition, we consider combining the automation and management approach with the dynamical and auto-pipeline generation (see (Reiterer, Schiffer, and Benedikt 2022)) and visualization prototype to enhance the DevOps practice in the platform. Furthermore, we plan to find out solutions to completely ensure the IP protection of the simulation models in the platform. We plan to simulate these models as servers connected to the pipeline without uploading them into the platform.

Acknowledgements

The authors would like to acknowledge the financial support of the COMET project Rail4Future (882504) within the COMET Competence Centers for Excellent Technologies from the Austrian Federal Ministry for Climate Action (BMK), the Austrian Federal Ministry for Digital and Economic Affairs (BMDW), the Vienna Business Agency and the Styrian Business Promotion Agency (SFG). The Austrian Research Promotion Agency (FFG) has been authorised for the COMET program management. We further thank for the support from ÖBB-Infrastructure AG and Siemens Mobility Austria GmbH.

References

Barbie, Alexander, Wilhelm Hasselbring, and Malte Hansen (2023). "Enabling Automated Integration Testing of Smart Farming Applications via Digital Twin Prototypes". In: *2023 IEEE Smart World Congress (SWC)*. IEEE, pp. 1–8. DOI: 10.1109/SWC57546.2023.10449240.

Functional Mock-up Interface (FMI) Standard (2024). URL: https://fmi-standard.org/ (visited on 2024-08-09).

Git Documentation (2024). URL: https://git-scm.com/docs/git (visited on 2024-04-18).

GitHub - FMI Kit for Simulink (2024). URL: https://github.com/CATIA-Systems/FMIKit-Simulink (visited on 2024-04-18).

GitHub - FMPy (2024). URL: https://github.com/CATIA-Systems/FMPy (visited on 2024-04-18).

Golightly, David et al. (2022). "A feasibility assessment of multi-modelling approaches for rail decarbonisation systems simulation". In: *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit* 236.6, pp. 715–732. DOI: 10.1177/09544097211039395.

Hällqvist, Robert et al. (2021). "Engineering domain interoperability using the system structure and parameterization (ssp) standard". In: *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*, pp. 37–48. DOI: 10.3384/ecp2118137.

Hamarat, Mehmet, Mayorkinos Papaelias, and Sakdirat Kaewunruen (2022). "Fatigue damage assessment of complex railway turnout crossings via Peridynamics-based digital twin". In: *Scientific reports* 12.1, p. 14377. DOI: 10.1038/s41598-022-18452-w.

Hotzel Escardo, Tomas et al. (2021). "Modelling train driver behaviour in railway co-simulations". In: *Software Engineering and Formal Methods. SEFM 2020 Collocated Workshops: ASYDE, CIFMA, and CoSim-CPS, Amsterdam, The Netherlands, September 14–15, 2020, Revised Selected Papers 18*. Springer, pp. 249–262. DOI: 10.1007/978-3-030-67220-1_19.

Hugues, Jerome et al. (2020). "Twinops-devops meets model-based engineering and digital twins for the engineering of cps". In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. New York, NY, USA: Association for Computing Machinery, pp. 1–5. DOI: 10.1145/3417990.3421446.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Iwnicki, Simon (1998). "The Manchester Benchmarks for Rail Vehicle Simulation". In: *Vehicle System Dynamics* 30.3-4, pp. 295–313. DOI: 10.1080/00423119808969454. URL: https://api.semanticscholar.org/CorpusID:110412927.

Kaewunruen, Sakdirat et al. (2023). "Digital Twins for Managing Railway Bridge Maintenance, Resilience, and Climate Change Adaptation". In: *Sensors* 23.1. ISSN: 1424-8220. DOI: 10.3390/s23010252. URL: https://www.mdpi.com/1424-8220/23/1/252.

Kiran, Kumar H K et al. (2021). "An Approach to basic GUI-enabled CI/CD pipeline with Static Analysis tool". In: vol. 23. 6. Journal of University of Shanghai for Science and Technology, pp. 683–693. DOI: 10.51201/JUSST/21/05317.

Kugu, Ozan et al. (2023). "An FMI-and SSP-based Model Integration Methodology for a Digital Twin Platform of a Holistic Railway Infrastructure System". In: *Proceedings of the 15th International Modelica Conference 2023, Aachen, October 9-11*, pp. 717–726. DOI: 10.3384/ecp204717.

Pieper, Tobias and Roman Obermaisser (2018). "Distributed co-simulation for software-in-the-loop testing of networked railway systems". In: *2018 7th Mediterranean conference on embedded computing (MECO)*. IEEE, pp. 1–5. DOI: 10.1109/MECO.2018.8406023.

Reiterer, Stefan H, Clemens Schiffer, and Martin Benedikt (2022). "A Graph-Based Metadata Model for DevOps in Simulation-Driven Development and Generation of DCP Configurations". In: *Electronics* 11.20, p. 3325. DOI: 10.3390/electronics11203325.

Reiterer, Stefan H, Clemens Schiffer, and Mario Schwaiger (2023). "A graph-based meta-data model for devops: Extensions to ssp and sysml2 and a review on the dcp standard". In: *Proceedings of the 15th International Modelica Conference 2023, Aachen, October 9-11*, pp. 159–166. DOI: 10.3384/ecp204159.

Sethi, Farhana (2020). "Automating software code deployment using continuous integration and continuous delivery pipeline for business intelligence solutions". In: *Authorea Preprints*. DOI: 10.22541/au.160373745.57814465/v1.

System Structure and Parameterization (SSP) Standard (2024). URL: https://ssp-standard.org/ (visited on 2024-08-09).

Villa, Davide et al. (2024). "Colosseum as a digital twin: Bridging real-world experimentation and wireless network emulation". In: *IEEE Transactions on Mobile Computing*, pp. 1–17. DOI: 10.1109/TMC.2024.3359596.

Zampetti, Fiorella, Simone Scalabrino, et al. (2017). "How open source projects use static code analysis tools in continuous integration pipelines". In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, pp. 334–344. DOI: 10.1109/MSR.2017.2.

Zampetti, Fiorella, Damian Tamburri, et al. (2023). "Continuous integration and delivery practices for cyber-physical systems: An interview-based study". In: *ACM Transactions on Software Engineering and Methodology* 32.3, pp. 1–44. DOI: 10.1145/3571854.

Zhang, Shiyao et al. (2021). "A digital-twin-assisted fault diagnosis of railway point machine". In: *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*, pp. 430–433. DOI: 10.1109/DTPI52967.2021.9540118.

Zhou, Shiyang, Stefan Dumss, et al. (2022). "A conceptual model-based digital twin platform for holistic large-scale railway infrastructure systems". In: *Procedia CIRP* 109, pp. 362–367. DOI: 10.1016/j.procir.2022.05.263.

Zhou, Shiyang, Ozan Kugu, et al. (2024). "A Reinforcement-Learning-based Parameter Tuning Methodology for Traction Control in the Holistic Railway Digital Twin System". In: *Procedia CIRP* 128, pp. 828–833. DOI: 10.1016/j.procir.2024.06.040.

Zhou, Shiyang, Alexander Meierhofer, et al. (2023). "A Machine-Learning-based Surrogate Modeling Methodology for Submodel Integration in the Holistic Railway Digital Twin Platform". In: *Procedia CIRP* 119, pp. 345–350. DOI: 10.1016/j.procir.2023.02.141.

Thermo-Fluid Modeling Framework for Supercomputer Digital Twins: Part 1, Demonstration at Exascale

Vineet Kumar[1] Scott Greenwood[1] Wesley Brewer[2]
David Grant[3] Nathan Parkison[3] Wesley Williams[1]

[1]Fusion and Fission Energy and Science Directorate, Oak Ridge National Laboratory, USA, {kumarv, greenwoodms, williamswc}@ornl.gov
[2]Computing & Computational Sciences Directorate, Oak Ridge National Laboratory, USA, brewerwh@ornl.gov
[3]Facilities & Operations Directorate, Oak Ridge National Laboratory, USA, {grantdr, parkisonjn}@ornl.gov

Abstract

A thermo-fluid modeling framework is being developed for ExaDigiT—an open-source framework for developing comprehensive digital twins of liquid-cooled supercomputers. The work is being conducted in two parts, and discussion is divided into two companion papers. The work documented in this paper focuses on the development of a cooling system library in Dymola for the Frontier supercomputer at Oak Ridge National Laboratory. The second part, outlined in a companion paper, focuses on a templating structure called *Auto-CSM* for easily creating model-agnostic, physics-based thermo-fluid cooling system models for liquid-cooled supercomputers using a text-based schema. The cooling model is being developed using primarily the open-source Transient Simulation Framework of Reconfigurable Models (TRANSFORM) library. The library follows the templating architecture developed within the TRANSFORM library for modeling subsystems. A full-system validation was performed to validate a very simple model that is integrated with the system controls, and the results are presented herein.

Keywords: cooling system, liquid-cooled supercomputers, Frontier, Modelica

1 Introduction

The electricity consumption of data centers is projected to increase in the United States from around 200 TWh in 2022—which represents about 4% of the country's total electricity demand—to 260 TWh, which is expected to be around 6% of the total electricity demand (IEA 2024). Additionally, water consumption is expected to be significant for both direct and indirect liquid-cooled supercomputing clusters. It is projected that approximately 15–27% of the energy consumed by data centers can be reduced via advanced cooling technologies, such as natural and liquid cooling techniques (Zhu et al. 2023). Therefore, there is an acute need for dynamic modeling of liquid-cooled supercomputing clusters using open-source tools like Modelica. The potential use cases for such a model could include the design and commissioning phase of new liquid-cooled data centers, as well as for operational optimization of existing facilities (Todd et al. 2021). ExaDigiT is a comprehensive open-source framework under development at Oak Ridge National Laboratory (ORNL) that focuses primarily on liquid-cooled supercomputers. Figure 1 shows the high-level architecture of the ExaDigiT framework, which consists of three main modules: (1) the cooling model discussed here, (2) a resource allocator and power simulator (RAPS), (3) a visual analytics module consisting of both an augmented reality component for 3D interactive visualization and a web-based dashboard for launching experiments and creating 2D plots of power and cooling behavior. The development of ExaDigiT is currently centered around the 2 exaflop Frontier supercomputer, which was deployed in 2022 at ORNL's Oak Ridge Leadership Computing Facility (OLCF) (Atchley et al. 2023).

Most of the work on data center cooling has been focused on air-cooled systems—see, for example, (Lee and Chen 2013; Ham and Jeong 2016; Fu, Wetter, and Zuo 2018)—especially on cooling efficiency. Zhang et al. (Zhang et al. 2022) built a digital twin for air-cooled data centers using a combination of computational fluid dynamics (CFD) using 6SigmaDC (*DataCenter Design Software* n.d.) and an AI-based XGBoost model. They used AI to optimize the control parameters of the air conditioning system so as to optimize the power usage effectiveness (PUE) of a data center. Heydari et al. (Heydari et al. 2022) performed extensive analysis of secondary flow loops to deploy liquid-cooled systems in air-cooled data centers by a combination of numerical modeling and experimental testing of four different cooling loops. The numerical modeling for different cold plate designs was performed using a commercial CFD solver 6SigmaET (*Data-*

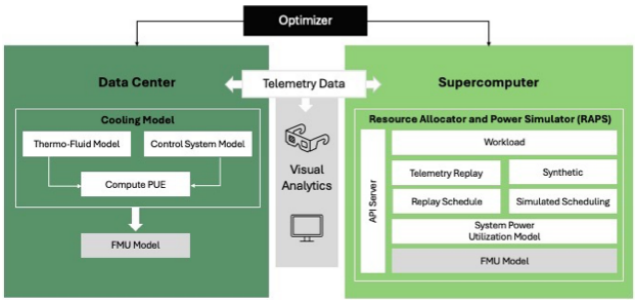


Figure 1. ExaDigiT architecture overview (Brewer et al. 2024)

Center Design Software n.d.), and flow network modeling of a liquid-cooled rack was performed with a custom system model and the commercial CFD solver 6SigmaRoom (*DataCenter Design Software* n.d.). Modi et al. (Modi et al. 2023) performed transient CFD simulations to optimize different flow configurations for rack-level models using the commercial CFD solver 6SigmaET. Modelica has been previously used by Fu et al. (Fu, Wetter, and Zuo 2018) to model air-cooled data center systems, which also used the Modelica buildings library (MBL) (Wetter et al. 2014). Leva et al. (Lee and Chen 2013) developed an open-source Modelica library using an object-oriented modeling (OOM) framework to model both air-cooled and liquid-cooled supercomputing clusters, and it can couple with 3D-ICE for chip simulations.

The current study primarily uses the Transient Simulation Framework of Reconfigurable Models (TRANSFORM) library, which is a Modelica-based open-source library developed at ORNL to enable rapid development of dynamic, advanced energy systems with an extensible system modeling tool (M. S. Greenwood 2017). TRANSFORM is organized as a series of packages, each of which has a general application; the library sub-categorizes models within each package, which helps users easily locate a component (M Scott Greenwood et al. 2020). Additional details on the TRANSFORM library, including model templates and the supervisory control system, can be found in previous work by the authors (M. S. Greenwood 2017; Michael Scott Greenwood et al. 2017). TRANSFORM was developed using the commercial integrated development environment (IDE) Dymola by Dassault Systèmes (Systèmes 2022) but should be compatible with other IDEs that are compatible with the Modelica specification 3.4+ (M. S. Greenwood 2017). In the current study, Dymola was used as the IDE. The current library has an additional dependency on the Buildings library (Wetter et al. 2014) for the variable speed cooling tower model.

This study is divided into two parts. The objectives of this work that form part one of this study are (1) to demonstrate a use case of the templating structure that is being laid out for modeling liquid-cooled supercomputing clusters in part 2—documented in a companion paper—and (2) to perform a validation exercise of the overall model using telemetry data. The supercomputing cluster chosen for validation is that of Frontier at ORNL. The validation exercise is divided into two parts: a component validation effort and the overall validation effort. The validation for the overall model serves to demonstrate how this library can be used to create very simple system models upon which additional complexity can be layered.

2 Frontier Model Description

2.1 Physical Facility Description

Frontier consists of 74 liquid-cooled HPE Cray EX supercomputing cabinets, which hold a total of 9472 compute nodes (Atchley et al. 2023; Choi 2022). Each Frontier node, designated as Cray EX 235a, contains one AMD 7A53 EPYC^{TM} 64-core "Trento" processor and four AMD Instinct MI250x GPUs. Each cabinet of Frontier consists of four shelves, each shelf has two chassis, and each chassis contains four active rectifiers and eight compute blades—a total of 64 blades and 32 rectifiers per cabinet (Brewer et al. 2024). Each cabinet is directly supplied with three-phase power from the distribution transformer switchboard, which is converted from 480 V AC to 380 DC voltage using AC-DC rectifiers and is subsequently stepped down to 48 V DC using super intermediate voltage converters (SIVOCs). The 48V DC is what supplies power to the node. Since each blade contains two nodes, there are two SIVOC converters per blade. Each HPE Cray EX cabinet can support up to 400 kW of power. Further details regarding the Frontier compute architecture can be found in (Atchley et al. 2023), whereas details regarding the Frontier system power conversion, including modeling conversion losses, are covered in forthcoming papers (Brewer et al. 2024; Wojda et al. 2024).

A simplified schematic of the overall cooling system layout for Frontier is shown in Figure 2; locations are marked in the figure to indicate the current cooling model outputs. The cooling system can be divided into three cooling loops, referred to in this paper as the cooling distribution unit (CDU) loop, the intermediate or high-temperature water (HTW) loop, and the cooling tower water (CTW) loop. The CDUs are used to remove heat from the compute nodes via forced convective liquid cooling. The CDUs deployed for the Frontier system can remove approximately 1.6 MW of heat, which translates to four cabinets. In practice, each CDU serves three cabinets at most, and some CDUs serve two cabinets. In total, there are 27 CDUs installed in the data center to serve the compute cabinets; of these, 25 are in operation, serving the 74 cabinets. The data center room contains piping underneath the raised floor which distribute primary flow to the CDUs. Additionally, the room also contains air handling systems and other auxiliary systems; these systems are not covered here, as they were not modeled. These systems serve a number of functions, including maintaining the dew point temperature in the data center room within an adjustable setpoint. Therefore, they will be considered in a more detailed modeling effort in the future.

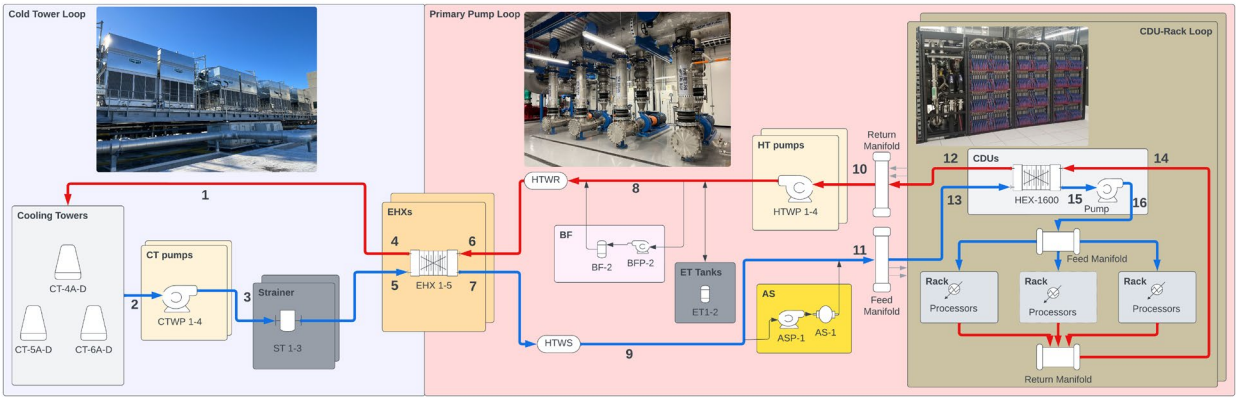


Figure 2. Simplified schematic of Frontier cooling system. Locations at which the cooling model predicts pressures, temperatures, and flow rates are numbered

From the current modeling perspective, the primary flow pumped from the HTW loop is used to remove the CDUs heat load generated primarily by the compute nodes in the CDU heat exchanger (shown as HEX-1600 in Figure 2). The secondary side of the CDU supplies pressurized liquid coolant via two pumps in parallel with a flow rate of $\sim$ 250–300 gpm at a system (gauge) pressure of $\sim$ 3.5 bar. This supply coolant is distributed in parallel to the cabinets/racks. In each rack, the flow passes through the 64 compute blades and the 32 rectifiers, with both actively cooled. Each compute blade consists of CPU and GPU cold plates, as well as cooling channels for peripheral components from a heat load perspective, such as memory, network interface cards (NICs), and SIVOCs. The hot water from the return side of the secondary CDU loop is cooled in the heat exchanger, which completes the loop. There is a tank on the secondary side to maintain the system pressure, as well as instrumentation that consists principally of temperature sensors, pressure gauges, and flowmeters. The primary side has a control valve that regulates the flow into the CDU based on the supply-side temperature, which is discussed further in Section 2.3.

The intermediate loop (or the HTW loop) provides up to 40 MW of process cooling at around 12–32 °C. The intermediate loop mainly consists of four variable-speed HTW pumps (HTWPs), expandable to 8 pumps; five economizer heat exchangers (EHXs) (four are considered in the model), expandable to 8 heat exchangers; and associated piping, which connects to the data center room for supply and return. The intermediate loop also consists of air separator pumps (denoted as ASP in Figure 2), HTW water bag filter (denoted as BF in Figure 2), and other systems that are peripheral to the current modeling perspective. The HTWPs supply water to the data center room at approximately 5000–6000 gpm at a gauge (system) pressure of $\sim$ 6.2 bar. The majority of this flow is directed toward the CDUs. About 10–15% of the flow is diverted through the bypass flow valve when system flows are low, and is mixed with the hot return, which results in a slightly decreased bulk return temperature. Some of the HTW flow also provides cooling for the Orion file system ($\sim$ 10%) when operating with supply temperatures less than 18 °C. This aspect was not modeled but would be considered in the future. The piping network in the central energy plant and the data center room are vast and complex. Here, considerable simplifications were made to capture the essence of the piping network; these simplifications are discussed in Section 2.3.

The CTW loop principally consists of five counter flow cooling towers, expandable to ten cooling towers, and four variable-speed CTW pumps (CTWPs), expandable to ten pumps. Each cooling tower (CT) has four independent cells with individual control valves and four corresponding variable-speed CTs fans. The CTWPs supply water at approximately 6000–10000 gpm at a gauge (system) pressure of approximately 1.5 bar. The CTW loop also consists of strainers for cooling water blowdown as well as other systems associated with chemical treatment of the cooling tower water, which are not shown in Figure 2. These systems are currently ignored in the model. The main flow path in the CTW loop (see Figure 2) starts from the hot water from the primary return side of the EHX, which flows to the cooling towers located on the roof of the central energy plant; the cold return from the cooling towers is pumped to the primary supply side of the EHXs. In the current model, only four cooling towers were modeled (i.e., 16 independent cells). This is a reasonable approximation, as generally 9–15 cells were in operation for the range of data that was analyzed. However, the current model could be easily expanded to include the additional CT or any of the other components with the templating system that is being put into place to easily generate Modelica models for liquid-cooled supercomputing clusters (S. e. a. Greenwood 2024).

2.2 Library Structure for the Frontier Model

The Modelica model library for the Frontier system is currently hosted in an internal ORNL Git repo: `https:`

//code.ornl.gov/exadigit/coolingModel. It is expected to be open-sourced within the next few months. The library relies primarily on components from the TRANSFORM model, as previously discussed. An additional dependency is the Buildings library from Lawrence Berkeley National Laboratory (Fu, Zuo, et al. 2019) for the variable-speed CT model. The library is being developed in Dymola (Systèmes 2022) as the TRANSFORM library was developed in Dymola. In the future, the library will be extended to work with Open Modelica. The library structure for the Frontier cooling model follows from the subsystem templating approach used in TRANSFORM library. This structure allows for ease of modeling integrated systems, and it is extensively discussed in previous work by Greenwood (M. S. Greenwood 2017). A sample of the subsystem templating is shown in Figure 3 for the CDU subsystem.

In the figure, the left-hand side shows the structure of the library for the CDU package, which opens to a directory structure with the following packages: 'Examples', 'Components', 'Data', 'ControlSystems', and 'BaseClasses'. The physical models that are located in the top-level directory extend from the 'Partial_SubSystem model' within the 'BaseClasses' package. The 'BaseClasses' directory also defines signal buses and actuator buses for the input and output signals, respectively, from the 'Partial_SubSystem model' to the 'Partial_ControlSystem model'. The models within the ControlSystems package are inherited from the 'Partial_ControlSystem' model. Similarly, the data records within the 'Data' package extend from the 'Record_Data' record within the 'BaseClasses' package. The 'Examples' directory contains example tests for the physical models. This type of layout, which exploits the inheritance and replaceability features of Modelica, allows for easily layering complexity. As an example, the compute cabinet model shown in Figure 3 is inherited from 'PartialCabinetModel' and is a simple model. This simple model can easily be replaced with a better-resolved model which also inherits from 'PartialCabinetModel', without requiring the modification of any higher-level models. More details on the layout can be found in (M. S. Greenwood 2017). A user of this library could easily copy an existing model, the CDU model and adapt it to their supercomputing facility as long as the input structure to their model remain unchanged. As noted earlier, the AutoCSM model is specifically being developed for this purpose.

The three main loops—that is, the CDU loop, the HTW loop, and the CTW loop—follow a similar structure to those of the CDU model and are part of the systems package. The reason why the systems package does not contain subsystem-specific models for blades and cabinets is that the control systems operate only at the level of the CDU. Beyond the 'System' package, there are four other top-level packages in the library: 'SubComponents', 'Examples', 'Icons', and 'FMUs'. The 'Examples' package consists of integrated subsystem model tests. The 'Icons' package defines the icons used for the various subsystems, as well as some of the components. The 'Controls' package hold the sub-models used in the control system models in the systems package, as well as unit tests. The 'SubComponents' package mainly contains packages for the 'Media' used throughout the library and a 'Fluid' package that holds the major components used in the different loops—CTs, heat exchangers, and pumps—along with unit tests for each. A cold plate model is currently under development within the 'Fluid' package.

Two fluid media have been used with the models in the library, specified in the 'Media' package within the 'SubComponents' package. The first, taken from the Modelica Standard Library, is water with constant properties, and the second is water with linear properties, which was taken from the TRANSFORM library. The principal disadvantage of the constant properties medium is that a tank must be modeled that fixes the pressure at the location of the tank. Therefore, the linear properties model was used in the current study as a balance between the constant properties model and the more comprehensive water model from the Modelica Standard library, which often presents convergence issues. In the actual system, the coolant has additives that inhibit bacterial growth. Here, it is assumed that the thermal properties of the medium are largely unchanged with the addition of a small concentration of additives.

The system model that was built for the current study was simplified by replacing pipes with a combination of fluid volumes and hydraulic resistances to model fluid mass and pressure drop, respectively. The hydraulic resistances were tuned with telemetry data. The heat exchangers were also approximated as a combination of fluid volumes and hydraulic resistances in series for each stream, with thermal data obtained from the manufacturer either as performance curves of overall heat transfer coefficient as a function of flow rates or constant values of overall heat transfer coefficient. For the pump models, pump curves were obtained from publicly available manufacturer data for all the pumps at their corresponding nominal pump speeds. The system controls were incorporated into the simple model, as the focus was to accurately capture the system dynamics. It must be noted that in the absence of pipes, the model lacks accuracy in capturing fluid flow dynamics, especially during sharp transitions. However, the compromise was a model with a shorter run time that can reasonably capture system dynamics for the range of data tested. Future extensions to this model could easily be made to incorporate pipe flow models with the templating structure in place.

The present model only simulates to the level of the CDU, and the cabinets are represented as a combination of simple volume components and hydraulic resistances from the TRANSFORM library (shown as 'Compute Cabinets' in Figure 3). Similar approximations have been used to model the heat exchangers, whereby the fluid flow in the primary and secondary streams is replaced by volumes

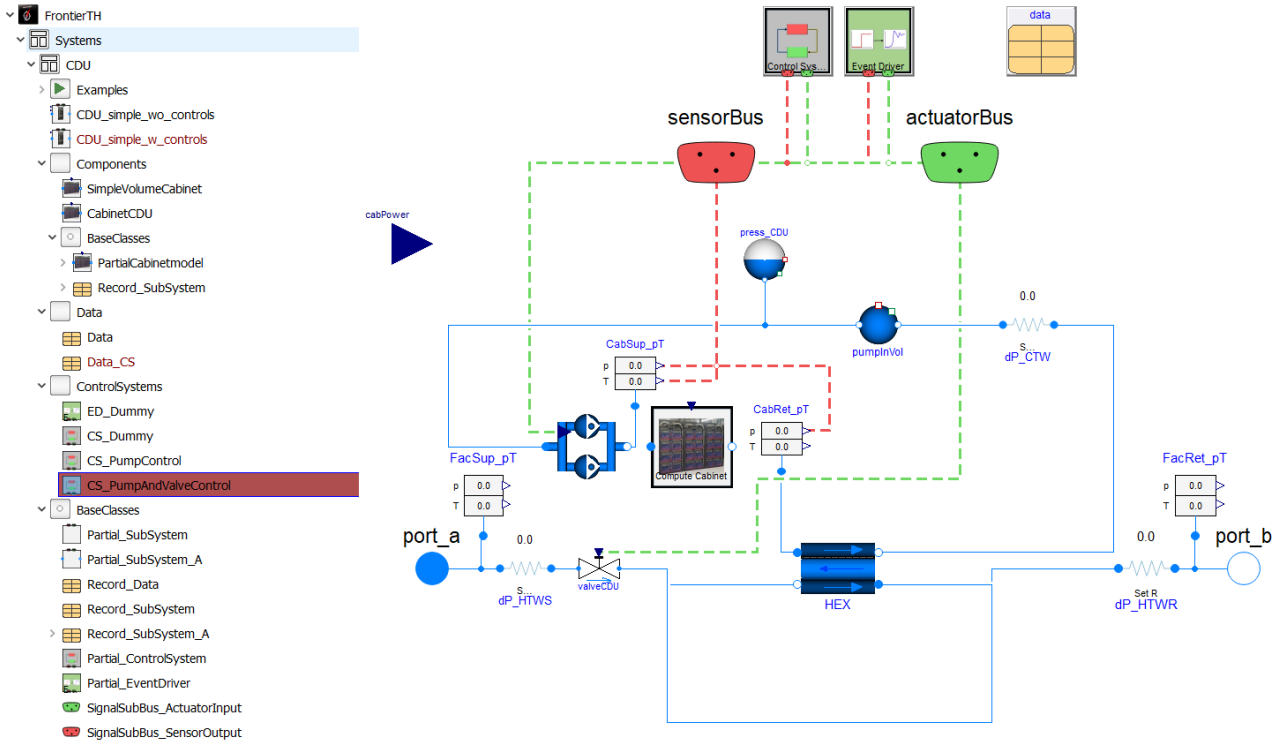


Figure 3. CDU model in Dymola based on the subsystem specific TRANSFORM library template (M. S. Greenwood 2017)

and resistances, and the overall heat transfer coefficients are taken from data. Lastly, publicly available data were used for all the pump curves.

2.3 Controls System Modeling

Figure 4 depicts the simplified control system logic currently implemented for the cooling model. The control system logic is divided into the central energy plant and the data center. A detailed overview of the control system logic is beyond the scope of this paper. The Modelica model captures the essentials of the control logic, which activates once the auto-operation of the physical cooling system has commenced after the start-up sequence has been completed. At this stage, the fail-safe configuration has not been implemented into the control logic, but the existing model could easily be extended to include it. The control logic for the system can be described as follows from Figure 4. Any disturbance in the CDU loop in terms of changes in the load, the HTW inlet pressure, or the HTW supply temperature would trigger the control system to bring the system back to an operational set point. Any given CDU can regulate its primary valve as its compute demand changes, and, consequently, the demand for more or less coolant flow in the primary side is regulated by the speed of the HTWPs via the differential pressure setpoint in the HTW loop. HTWPs stage up or down at a given moment depending on the % relative speed of the pumps currently in operation. A change in the primary supply temperature, on the other hand, is regulated by the CT loop by staging the number of CTs up or down. CTs stage up when the CTW return (CTWR) header pressure is at its maximum boundary and the HTW supply (HTWS) temperature is increasing, and, consequently, they stage down when the CTWR header pressure is at its minimum set point and the HTWS temperature is decreasing. Additionally, EHXs are staged up or down depending on the number of CTs in operation. Therefore, the criteria to achieve HTWS temperature stability inform both the staging of the CTs directly and the EHXs indirectly. Finally, the CTWPs maintain the header pressure in the CT loop within certain bounds by regulating its speed and staging the number of pumps in operation. Once the HTWS temperature has stabilized within certain bounds and the differential pressure setpoint in the HTW loop is met, the compute CDU is satisfied, as shown in Figure 4.

A proportional-integral-derivative (PID) controller is used to regulate the CDU relative % pump speeds based on the CDU loop differential pressure in the current model. Both the CDU pumps are assumed to be in operation at all times with the same speeds. This is a reasonable approximation based on telemetry data. A control value on the primary side, as just discussed, is used to regulate the primary coolant flow based on the secondary supply temperature setpoint. A snapshot of the controls used for the CDU loop is shown in Figure 5. At the start of the simulation, the CDU pumps and the control valve are fixed for numerical considerations and a 1.0 second delay clock is used to switch to the PID mode for both the pumps and the valve. Additional low-pass filters are employed to filter the output from the PIDs to prevent very high oscillatory

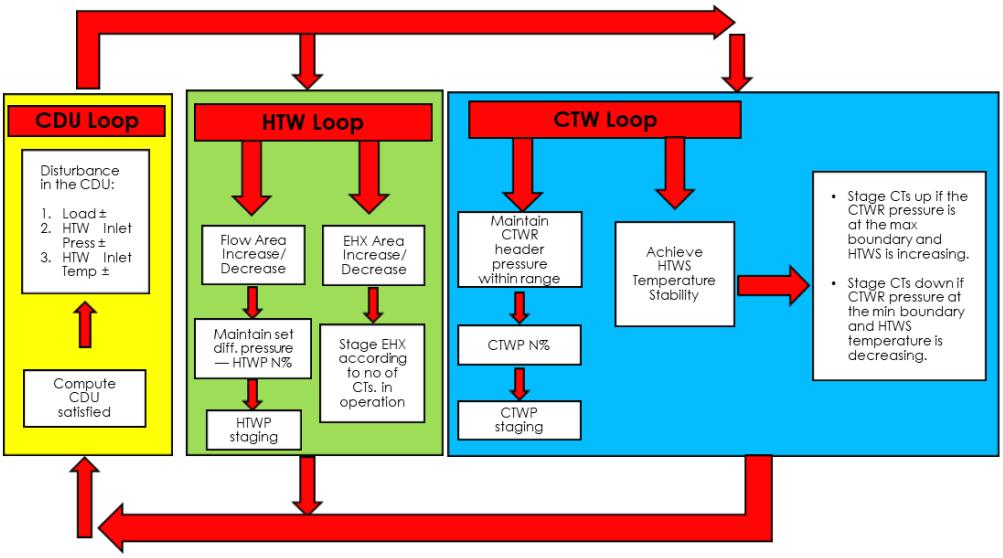


Figure 4. Frontier cooling model controls summary

behavior and to ensure numerical convergence.

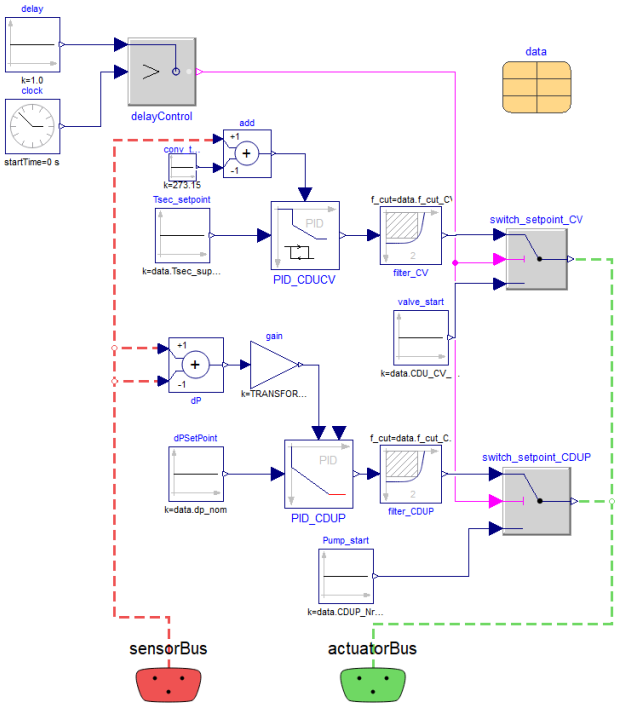


Figure 5. CDU controls model snapshot in Dymola

The intermediate loop similarly has a PID controller for the four HTWPs. The HTWPs are staged up or down depending on the relative % pump speeds of the running pumps, as previously discussed. The controls logic has been templated where possible, such as for the staging of the pumps. This should allow for easier extensions to other systems by users of the library. It must be noted that the HTW loop relies on the number of CTs in operation for staging the EHXs, and the staging of the CTs, in turn, relies on the HTWS temperature. This cross-flow of information exchange among subsystems is handled in the model by using a delay transfer function between the intermediate loop and the CT loop. A future scope would be to optimize the control parameters to achieve better system stability as well as respond quickly to a surge in the load.

3 Validation of the cooling model for Frontier

3.1 Component validation

Component validation was conducted as a first step before undertaking the full system validation. Component Verification & Validation (V&V) tests are important to ensure that model performs as expected—firstly, to validate the range of the expected telemetry data, and secondly to validate the components individually using telemetry data as boundary conditions before integration into the larger system. These tests can help to identify input errors in component parameters such as pump curves and heat exchanger data. Verification tests were not explicitly performed because the majority of the components used in the library were taken primarily from the TRANSFORM library and one component from the Buildings library, with modifications made in the margins which do not warrant a thorough verification process.

An example of component validation for the counterflow CDU heat exchanger is shown in Figure 6. This validation was performed using 22000 seconds of telemetry data for the primary flow rate, primary return pressure, primary supply temperature, secondary flow rate, secondary supply temperature, and secondary return pressure. As shown in the figure, the model can predict the primary (Facility) and the secondary (Cabinet) return temperatures

with reasonable accuracy. The over-prediction in the facility return temperature is most likely because of the secondary flowmeter instrument uncertainty. The lower end of the prediction uncertainty shown in Figure 6, matches the prediction data well for the majority of the $\sim$ 6 hour snapshot.

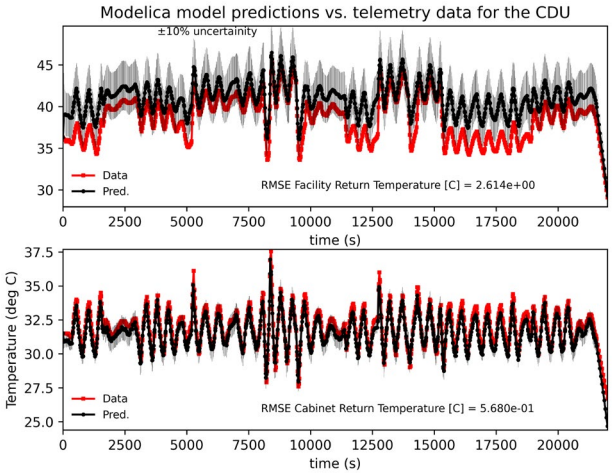


Figure 6. Modelica model predictions vs. telemetry data for the CDU return temperature for a $\sim$ 6 hour snapshot

Another validation test, shown in Figure 7, was used to test the control logic for the CTWP speed control, which is regulated by the CTWR header pressure setpoint. Similar to the CDU heat exchanger validation, telemetry data of the CTWR header pressure was used to test the model. The model can respond to the sharp changes in the pump speed after approximately 17 hours for this particular dataset but essentially filters the header pressure for smaller variation. Further improvements are required in the model to match the CTWP speed variation for changes of smaller magnitude the CTWR header pressure.

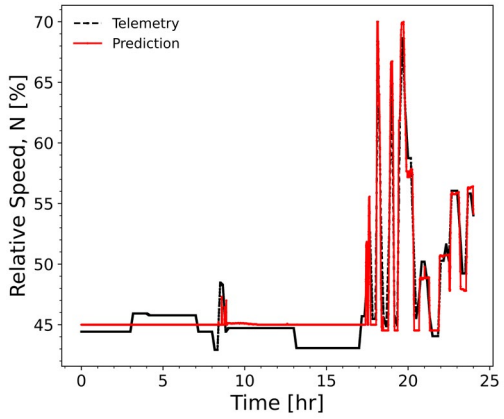


Figure 7. Modelica model predictions vs. telemetry data for the CTWP relative % pump speed for a 1 day snapshot

3.2 Complete system validation

A validation study of the entire *cooling model* (shown in Fig. 8) was conducted using $\sim$15 hours of telemetry data for a given day from the central energy plant and the data center down to the level of the CDU. The only inputs to the model were the power to the 25 CDUs and the wet bulb temperature as a function of time. The run time of the model exported as an an Functional Mock-up Unit (FMU) to simulate $\sim$15 hours of telemetry data on a standard windows workstation was about $\sim$20 minutes. Significant improvements have been made to the model to reduce the number of nonlinearities, and the current model runs 2-3x faster than the model discussed here. The translated model discussed here has 912 states with $\sim$14000 time varying variables and $\sim$13000 alias variables. The model outputs for the model exported as an FMU, have been listed in Table 1. The power to the CDUs for the validation exercise was calculated from the heat removed by the cooling water using telemetry data. This was found to closely match the power sensor data, which were obtained after the validation study was performed. After the Resource Allocator and Power Simulator (RAPS) module being developed as part of the ExaDigiT digital twin framework has been thoroughly validated with power sensor data, it will be utilized as an input to the cooling model.

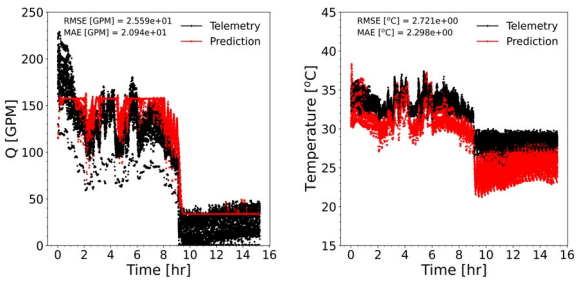
Parameter	Description
Pressure	at locations shown in Figure 2
Temperature	at locations shown in Figure 2
Flow rate	at locations shown in Figure 2
Pump speed	% speed of HTWPs & CTWPs
Pump staging	operational HTWPs & CTWPs
EHX staging	operational EHXs
CT staging	operational CTs
Power consumption	HTWPs, CTWPs & CT fans
PUE	for the entire facility

Table 1. Outputs for the cooling model exported as an FMU.

The first second of predictions was removed from the analysis of $\sim$15 hours because the model predictions are biased by initial conditions at the very start of the transient. The resolution of the telemetry data varied from 15 seconds at the level of the CDU to 10 minutes for some of the facility telemetry data. For consistency, all measurements and predictions were interpolated to 15 seconds intervals for comparison. The annotations on the plots in Fig. 8 (a–e) correspond to stations in Fig. 2. A few observations can be made when comparing model predictions with telemetry data. The Frontier system was idle for about half a day because of system upgrades, which is why the cooling system load is at a minimum beyond $\sim$ 10 hours. This coincidentally proved to be a good transient test to see how the model performs in transition from a loaded system state to an idle state. For most of the predicted parameters, some of which are shown in Fig.

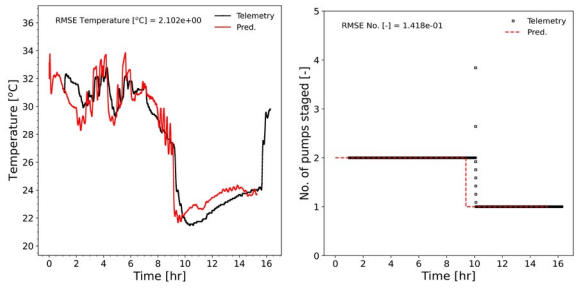
8, the trend-wise predictions are good up to about $\sim$ 10 hours, after which some deviation occurred in the predicted facility parameters when the power to the CDUs is at a minimum. This can be confirmed by the good agreement in the primary flow rates (shown in Fig. 8(a)) up to $\sim$ 10 hours. However, in the physical system, the primary flow in the intermediate loop is maintained at a minimum of $\sim$3000 gpm, whereas the model predicted a minimum flow of $\sim$2300 gpm (not shown). This flow difference is attributed to an additional bypass flow of $\sim$700 gpm in the actual system. Improving this aspect of the model should improve the prediction when the system transitions to an idle state. The model predictions for the CDUs secondary supply temperatures (not shown) show greater fluctuation than the physical system, which does a good job maintaining the temperature at the setpoint. This result suggests that the controls for the primary valve in the CDU must be further investigated. The staging of the HTWPs is predicted to occur earlier than it does in the actual system. The staging of the CTWPs and the CTs (not shown) must also be improved. It must be noted that there are manual overrides within the system, such as those for manually staging the CTs, which were deployed during this particular transient. This is a feature that could be introduced into the model in the future. Overall, the root mean square error (RMSE) of the parameters shown in Fig. 8 are within reasonable bounds, and a future study would focus on model uncertainty.

Finally, the comparison between the PUE predicted by the model and that calculated from telemetry data is shown in Fig. 8(f). The predicted PUE is within four percent of the calculated PUE, within the range of data tested ($\sim$8.3 hours). It must be noted that in both the calculations, the auxiliary systems considered for power consumption are the following: CDU Pumps (CDUPs), HTWPs, CTWPs, and CT fans. Other auxiliary systems such as the air-handling system are not considered in the calculation, as they were not modeled. Therefore, it is expected that the actual PUE would be higher.



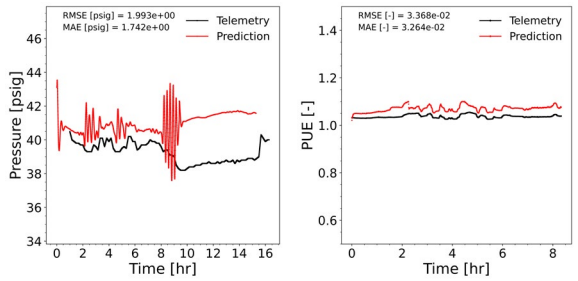
(a) Primary CDUs flow rate predictions (Station 12 in Fig. 2)

(b) Primary CDUs return temp. predictions (Station 12 in Fig. 2)



(c) HTW return temperature prediction (Station 10 in Fig. 2)

(d) HTWP staging (Station 10 in Fig. 2)



(e) HTW return pressure predictions (Station 10 in Fig. 2)

(f) PUE Modelica model predictions

Figure 8. Modelica model predictions (exported as an FMU) vs. telemetry data for the CDU and the central energy plant.

4 Conclusions

This paper presents the cooling model that is being developed in Modelica using Dymola as part of the ExaDigiT project to develop digital twins for liquid-cooled exascale supercomputers. The cooling model is being developed using primarily the open-source TRANSFORM library developed at ORNL, with the cooling tower model from the Buildings library. The overall goal is to develop a templating structure, Auto-CSM, for creating physics-based thermo-fluid cooling system models. Auto-CSM seeks to streamline the creation of cooling system model (CSM) for integration into the ExaDigiT framework, and that work is covered in Part 2 of the study and is documented in a companion paper. While Auto-CSM is being developed, the current study (Part 1) focuses on the cooling system library that is being developed and demonstrated on the cooling system of the 2 exaflop Frontier supercomputer at ORNL. The library follows the templating architecture developed within the TRANSFORM library for modeling subsystems and integrating them to quickly model complex systems. Although the library is currently hosted in an internal Git repo, it is expected to be open-sourced within the next few months.

The subsytems used for the cooling system library to model Frontier are the following: the CDU loop, the HTW loop, and the CT loop. It remains to be seen how generalizable these subsystems are to other supercomputing clusters. The simplified model, which makes use of fluid volumes and hydraulic resistances in place of pipes, extends only to the level of the CDU and is integrated with system controls. Extending the model to the level of the compute blade would result in a more accurate thermal response prediction with the downside of an increased run

time. A component validation was conducted before performing the full model validation with telemetry data for an approximately 15 hour snapshot that was provided for a given day. The model performed reasonably well, especially when the system was loaded, and significant improvements have been made to improve model performance in terms of runtime and robustness. Future use cases for such a model could be both in the design phase when designing new systems or optimizing the operation of existing systems.

Acknowledgments

This research was sponsored by and used resources of the Oak Ridge Leadership Computing Facility (OLCF), which is a DOE Office of Science User Facility at the Oak Ridge National Laboratory (ORNL) supported by the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

Atchley, Scott et al. (2023). "Frontier: Exploring Exascale". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16.

Brewer, Wesley et al. (2024-11). "A Digital Twin Framework for Liquid-cooled Supercomputers as Demonstrated at Exascale". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

Choi, Charles Q. (2022). *Beneath the Hood of the First Exascale Computer*. https://spectrum.ieee.org/frontier-exascale-supercomputer. [Online; accessed 4-April-2024].

DataCenter Design Software (n.d.). Available at https://www.cadence.com/en_US/home/resources/datasheets/datacenter-design-software-ds.html. Accessed: 2024-03-12.

Fu, Yangyang, Michael Wetter, and Wangda Zuo (2018). *Modelica models for data center cooling systems*. Tech. rep. Univ. of Colorado, Boulder, CO (United States).

Fu, Yangyang, Wangda Zuo, et al. (2019). "Equation-based object-oriented modeling and simulation of data center cooling systems". In: *Energy and Buildings* 198, pp. 503–519.

Greenwood, M Scott et al. (2020). "Demonstration of the advanced dynamic system modeling tool TRANSFORM in a molten salt reactor application via a model of the molten salt demonstration reactor". In: *Nuclear Technology* 206.3, pp. 478–504.

Greenwood, M. S. (2017-09). *TRANSFORM - TRANsient Simulation Framework of Reconfigurable Models*. Computer Software. Accessed on August 30, 2023. DOI: 10.11578/dc.20171025.2022. URL: https://github.com/ORNL-Modelica/TRANSFORM-Library.

Greenwood, Michael Scott et al. (2017-08). "A Templated Approach for Multi-Physics Modeling of Hybrid Energy Systems in Modelica". In: DOI: 10.2172/1427611. URL: https://www.osti.gov/biblio/1427611.

Greenwood, Scott et al. (2024-10). "Thermo-fluid Modeling Framework for Exascale Supercomputing Digital Twins: Part 2, Automated Cooling Models". In: *Proceedings of the American Modelica Conference 2024*. Under review.

Ham, Sang-Woo and Jae-Weon Jeong (2016). "Impact of aisle containment on energy performance of a data center when using an integrated water-side economizer". In: *Applied Thermal Engineering* 105, pp. 372–384.

Heydari, Ali et al. (2022). "Liquid to Liquid Cooling for High Heat Density Liquid Cooled Data Centers". In: *International Electronic Packaging Technical Conference and Exhibition*. Vol. 86557. American Society of Mechanical Engineers, V001T01A007.

IEA (2024). *Electricity 2024*. Tech. rep. INTERNATIONAL ENERGY AGENCY.

Lee, Kuei-Peng and Hsiang-Lun Chen (2013). "Analysis of energy saving potential of air-side free cooling for data centers in worldwide climate zones". In: *Energy and Buildings* 64, pp. 103–112.

Modi, Himanshu et al. (2023). "A Transient CFD Study on Implementation of Dynamic Liquid Cooling for Series and Parallel Arrangement of Components in a Server at Rack Level". In: *2023 39th Semiconductor Thermal Measurement, Modeling & Management Symposium (SEMI-THERM)*. IEEE, pp. 1–6.

Systèmes, Dassault (2022). *Dymola*. URL: https://www.3ds.com/products-services/catia/products/dymola/ (visited on 2023-02-10).

Todd, Austin et al. (2021). *Artificial Intelligence for Data Center Operations (AIOps)*. Tech. rep. National Renewable Energy Lab.(NREL), Golden, CO (United States).

Wetter, Michael et al. (2014). "Modelica buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Wojda, R et al. (2024-10). "Dynamic modeling of power conversion stages for an exascale supercomputer". In: *Proceedings of the IEEE Energy Conversion Congress & Exposition (ECCE)*. Under review.

Zhang, Ziting et al. (2022). "Smart DC: an AI and digital twin-based energy-saving solution for data centers". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, pp. 1–6.

Zhu, Hongyu et al. (2023). "Future data center energy-conservation and emission-reduction technologies in the context of smart and low-carbon city construction". In: *Sustainable Cities and Society* 89, p. 104322.

Thermo-Fluid Modeling Framework for Supercomputing Digital Twins: Part 2, Automated Cooling Models

Scott Greenwood[1] Vineet Kumar[1] Wesley Brewer[2]

[1]Fusion and Fission Energy and Science Directorate, Oak Ridge National Laboratory, USA, {kumarv, greenwoodms}@ornl.gov

[2]Computing & Computational Sciences Directorate, Oak Ridge National Laboratory, USA, brewerwh@ornl.gov

Abstract

The development of digital twins for the purpose of improving the energy efficiency of supercomputing facilities is a non-trivial endeavor that is complicated by the difficulty of creating physics-based thermo-fluid cooling system models (CSMs). Within ExaDigit—an open-source framework for liquid-cooled supercomputing digital twins—a thermo-fluid modeling framework is being developed. This effort has been segmented into two with two companion papers describing each portion of the overall effort. Part 1 focuses on the development of a cooling system library in Dymola for the Frontier supercomputer at Oak Ridge National Laboratory (Kumar et al. 2024). Part 2, this paper, describes an effort to create a template-based auto-generation methodology for CSMs, called *AutoCSM*. In this paper, an overview of the initial AutoCSM architecture and workflow is provided, along with a practical example using the Oak Ridge Leadership Computing Facility's (OLCF) Frontier supercomputer CSM. AutoCSM will (1) improve ExaDigiT's user accessibility by providing a flexible workflow for modularizing the creation of the CSM system and control logic, (2) decrease the development time of CSMs, and (3) standardize the method for incorporating CSMs into the ExaDigiT framework.

Keywords: digital twin, automation, cooling system, supercomputer, architecture, framework

1 Introduction

Across myriad disciplines, high-performance supercomputing facilities have long been a key resource for exploring complex scientific and engineering challenges, spurring technological innovation and opening new avenues of discovery (National Research Council 2005). As the problems being explored increase in complexity, and therefore computational cost, it will become increasingly difficult to make significant advances in energy efficiency.

1.1 Motivation

Fully operational in 2022, the Frontier supercomputer at Oak Ridge National Laboratory became the world's first exascale supercomputer (Atchley et al. 2023). It could be argued that the ability to achieve this milestone was feasible principally because of the enormous gains in hardware optimizations that have been made over the past decade. For example, if the technology of 2009 used in the Jaguar Supercomputer were used for the Frontier facility, the power requirements would have been multiple gigawatts. Instead, Frontier consumes only approximately 20 MW while achieving 1,000 times higher performance than Jaguar. Although enormous strides have been made in the past decade on computational hardware, it is postulated that hardware optimizations have approached their limits of being the primary means of obtaining efficiency improvements; instead, future efficiency gains will be dominated by software innovations such as operational performance (i.e., controls, job staging/prediction) (Brewer et al. 2024). The use of digital twins is one such software innovation that may be a primary means of achieving the necessary performance efficiency improvements for current and future supercomputing systems.

1.2 Digital Twins

A *digital twin* is defined as a virtual representation of a real-world system that synchronizes and exchanges information with the real-world system. In the context of exascale computing facilities, the digital twin is expected to require, at minimum, connections to- and models of- power consumption, the cooling system, and network behavior. A digital twin should also incorporate human–computer interfaces and optimization capabilities. The realization of a digital twin for such computing facilities is non-trivial given the complexity and scale of the facility, components, and data.

1.3 ExaDigiT

To help accelerate the development of exascale facility digital twins and their value in achieving efficiency

gains, Oak Ridge National Laboratory created ExaDigiT, an open-source framework for developing comprehensive digital twins for liquid-cooled supercomputers (Brewer et al. 2024). This framework is intended to streamline the creation and connection of the necessary cross-disciplinary systems and data as well as to provide methods of performing advanced analysis and predictive exploration to build toward sustainable, energy-efficient supercomputing.

1.4 Cooling System Model (CSM) Development

The broader development approach to ExaDigiT takes into account that an effective digital twin of an exascale facility will likely require physics-based system models of the liquid cooling system at various levels of fidelity throughout the system's life cycle—and that it must also be capable of capturing the transient operations of the system for optimization and scenario exploration. In support of ExaDigiT's development, a Modelica model of Frontier's cooling system was completed and is detailed in a companion paper (Kumar et al. 2024). Part of that modeling efforts purpose was to provide insights into the value of the cooling system model (CSM) toward delivering efficiency improvements and into how the CSM would be integrated into ExaDigiT .

1.5 AutoCSM

The Frontier CSM described in (Kumar et al. 2024) adopted a preliminary template architecture based on previous efforts (Greenwood et al. 2017a). However, as the ExaDigiT framework matured and more specific users of the framework were identified, it became apparent that supercomputing facilities—and datacenters more generally—could benefit from a CSM template approach more tailored to the way the systems are hierarchically designed, constructed, and operated. The creation of sufficiently accurate physics-based CSMs is a non-trivial exercise that requires domain-specific knowledge and good modeling experience and best practices. The additional complexities of iterative development and the integration of the model into a broader digital twin exacerbate the difficulty of achieving proper value from a CSM in the broader digital twin. Therefore, the creation of a template system-of-systems modeling approach for automating the development, deployment, and integration of CSMs for supercomputing facilities was proposed. This methodology is called *AutoCSM*.

The remainder of this paper describes the AutoCSM proof-of-concept methodology in the broader ExaDigiT context in which it is situated. A description of the current architecture and general workflow of the AutoCSM is then provided. An illustrative example of the adaption of the original Frontier model to the AutoCSM approach is then provided as well as the extension of that AutoCSM based model for exploration of a parallel datacenter study. Finally, because the users of ExaDigiT are expected to have limited familiarity with Modelica, and to help clearly address the role of AutoCSM in ExaDigiT explicitly, a section of exploring potential questions regarding AutoCSM is provided.

2 ExaDigiT & AutoCSM

The CSM within ExaDigiT consumes telemetry data (e.g., facility operation and job loading data, weather data), couples with the power simulator (RAPS), and ultimately produces operational predictions that can be leveraged for scenario exploration and facility health analysis. The CSM also provides data to reduced-order models for AI/ML facility studies, optimization, and visual analytics (Figure 1). Given the wide adoption of the open-source Functional Mock-up Interface (FMI) standard (*Functional Mock-up Interface* n.d.) by various tools and programming languages—and given its direct application to dynamic simulations at the levels of fidelity that are expected to be valuable for supercomputer digital twins—using Functional Mock-up Units (FMUs) was identified as the primary method of standardizing the incorporation of the CSM into the broader ExaDigiT framework. However, a question was posed about how the process for generating the FMU could be simplified for the user, making it less error prone and less time-intensive, as well as requiring less experience, while remaining agnostic to the underlying technology (e.g., commercial software, programming languages) used to create the CSM. This line of reasoning led to introducing AutoCSM as an optional interface layer for streamlining CSM model generation to FMU deployment. In the framework depicted in Figure 1, AutoCSM provides a means of automating the process for creating a simulation-ready CSM FMU, depicted on the left side.

3 AutoCSM

The motivating philosophy behind AutoCSM is to remove as many barriers as possible to incorporating CSMs into a facility's digital twin. Therefore, attention was given to identifying the broad architecture, identifying functional requirements, and determining how AutoCSM development could be compartmentalized and focused to achieve near-term impact. This section touches on each of these topics and provides a pseudo-code example of the AutoCSM workflow.

3.1 Architecture

AutoCSM is intended to be an optional interface that can be used to automate the generation of system models that have adopted a template architecture for rapid deployment to simulation-ready FMUs. The AutoCSM interface, or AutoCSM API (Figure 2), relies on the implementation of an API that has a language- or modeling-specific extension. That extension will be used to script the generation of the CSM in that language using a user template strategy that can be unique to their facility and language. Finally, the scripted model will be exported as an FMU using ex-

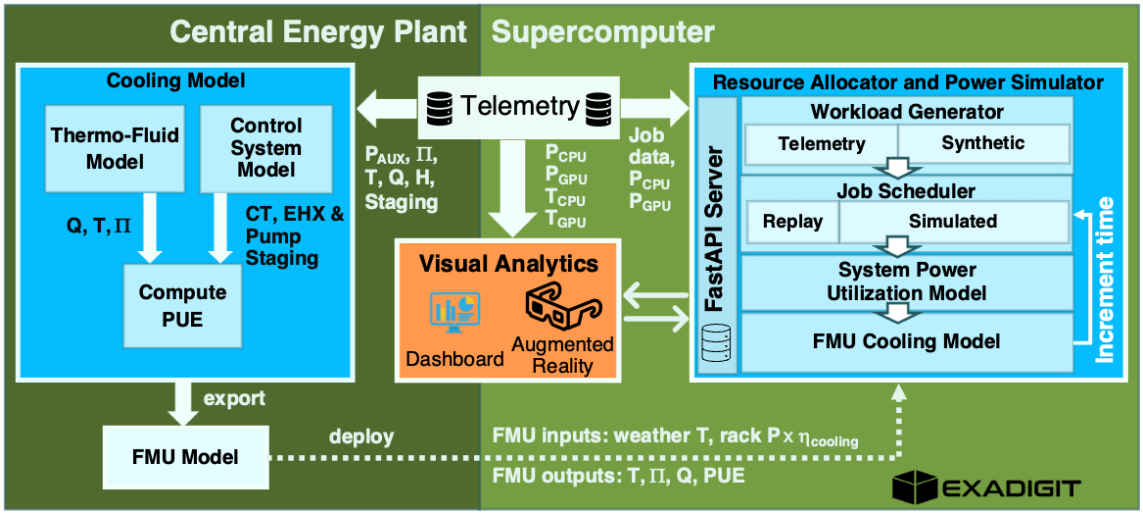


Figure 1. The base ExaDigiT framework, incorporating telemetry data, a power simulator (RAPS), visual analytics, and a CSM. (Brewer et al. 2024)

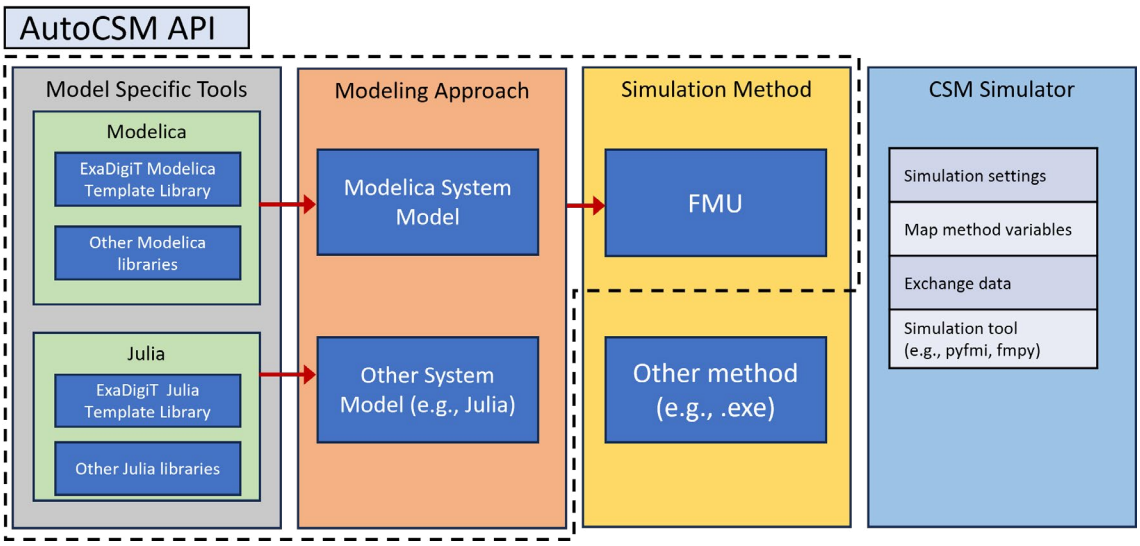


Figure 2. AutoCSM API in the broader ExaDigiT procedure. Dotted line indicates scope of AutoCSM.

isting tools. Again, the tool used will be user-definable and can be chosen according to the user's preferred modeling approach and tool (e.g., integrated development environment or language FMU export library). This FMU will then be consumed within the broader ExaDigiT CSM Simulator. At each step, AutoCSM is intended to remain as agnostic to specific facility requirements as possible by abstracting and reducing modeling requirements to the input specification (to be discussed).

Internal to the AutoCSM API is a specific procedure that is used to create an FMU export from user-input specification. Figure 3 illustrates the process of consuming input specifications. Additional details of the principal focus areas of the API are discussed in a subsequent section.

3.2 Functional Requirements

The functional requirements for the development of AutoCSM are identified below. Some have been mentioned previously but are repeated here for completeness. In general, the broad theme of the requirements is to provide the framework and avoid encroaching on a user's methods. AutoCSM will:

1. conform to the deployment requirements of ExaDigiT (e.g., open-source),
2. be language/tool agnostic (both across and within tools),
3. support custom specification extensions,
4. leverage existing solutions/methods where possible (e.g., third-party Python libraries),

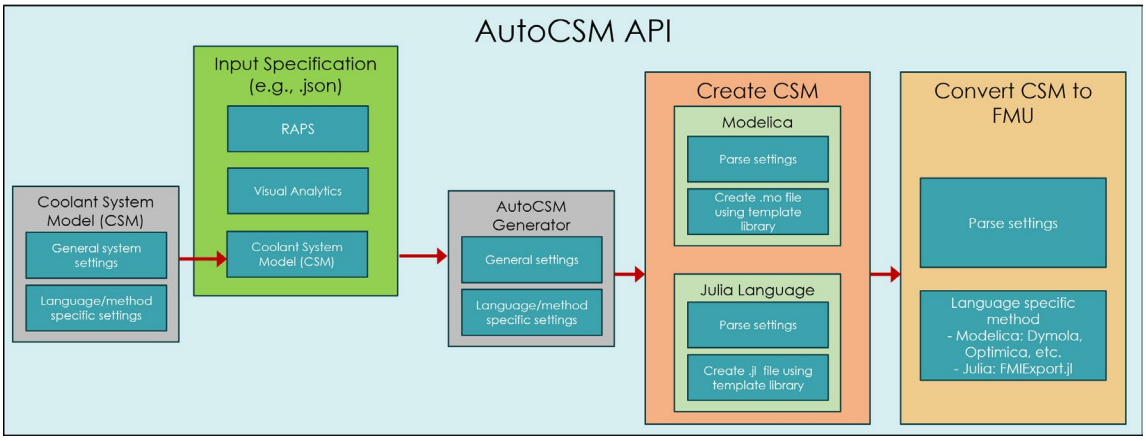


Figure 3. Internal to AutoCSM API procedure for CSM creation.

5. produce an FMU consistent with the requirements of ExaDigiT,

6. assemble pre-configured systems and subsystems (i.e., it will not generate the internals to a system model itself, such as connecting or creating pipes, volumes, etc.), and

7. not dictate the modeling template approach.

3.2.1 Modeling Approach Requirements

As shown in Figure 2, AutoCSM requires a modeling approach (e.g., Modelica, Julia, or other software/languages and libraries) that supports the creation of a formal modeling structure or template approach for the assembly of the FMU. To use AutoCSM for a selected modeling approach, three requirements have been identified.

1. Be templatable (i.e., able to organize models into system-subsystem architectures)

2. Be scriptable at the level of system assembly (i.e., not binary or otherwise inaccessible from outside tools)

3. Support FMU export

Satisfying these three requirements allows AutoCSM to support virtually any modeling approach a user may want to employ. If a requirement is not met with a selected tool, then the user will need to decide if that is an insurmountable issue with that tool (e.g., commercial software unable to be adapted) or if investment in an AutoCSM extension for that tool is feasible. As an open-source project, AutoCSM will be open to community contributions, and users may develop proprietary extensions that are not shared with the broader community.

3.3 AutoCSM Focus Areas

The development of the AutoCSM API consists of three distinct focus areas: input specification, the automated CSM generator, and an example template architecture for demonstration and development. Figure 3 provides an illustration of these areas within the internal AutoCSM methodology, from input specification to model creation to FMU generation. The following subsections elaborate on the role of each of these focus areas.

3.3.1 Input Specification

AutoCSM requires a standardized means of collecting the necessary information into a form that can be incorporated into the broader ExaDigiT input requirements. As referenced in Figure 3, the specification that proceeds to the CSM generation stage is an amalgam of the CSM-specific settings and additional information from other ExaDigiT pieces that a CSM may indirectly require. From a user's perspective, the CSM settings are what a modeler would define, and the CSM generator would use those CSM settings after they have undergone a settings compilation step. Potential examples of CSM input include the number of various systems used, modifications on parameters and input, and/or specification of the different versions of systems to be used.

3.3.2 AutoCSM Generator

The AutoCSM Generator provides the necessary logic to automatically translate the input specification into a complete system model that can be used within the ExaDigiT framework. The initial demonstration uses the Modelica language, but the API is extensible so that other languages or tools can be added as needed by users (e.g., Julia). To limit scope creep and edge cases, the initial development focused on the creation of co-simulation FMUs as the product of the automation process for use in the digital twin.

3.3.3 ExaDigiT Modelica Library

To enable automated generation of a CSM, a formalized architecture or template approach that compartmentalizes the facility into standardized systems, subsystems, and control logic is required. This is achievable because of how supercomputer facilities are constructed and orga-

nized and because of the level of fidelity necessary from the CSM for ExaDigiT's purposes. Therefore, work in this focus area involved creating a proof-of-concept template approach for assembling the facility.

As previously noted, a CSM for Frontier was developed (Kumar et al. 2024) to provide support to its operations and contribute to the development of ExaDigiT (**??**). The model outlined in the aforementioned work leveraged a template approach previously developed for advanced energy systems studies contained within the TRANSFORM Modelica library (Greenwood et al. 2017b). Although the adopted template approach has been adequate for its original purposes, the CSM would benefit from an adapted template approach that aligns better with a nested structure that could be leveraged by external programming languages and be better aligned with ExaDigiT. Therefore, an ExaDigiT specific Modelica template prototype was developed using other Modelica template approaches (Modelica Asscotiation 2024; Greenwood et al. 2017a) for inspiration and the Frontier CSM modeling efforts as a use case demonstration. Figure 4 shows the outline of the developed Modelica template system. This leverages identical underlying structures–*BaseClasses*–and replaceable components to create a common template–*TemplateSystem*–that can be duplicated by AutoCSM or a user and serve as the foundational structure to create an AutoCSM compliant model.

In addition to a common structure, at each system level the models are implemented as arrays such that all inputs/outputs to the simulation can be readily associated with ExaDigiT input/output requirements and the ExaDigiT user can easily customise the structure of their facility. For example, to access the summary output of a particular model a path of the following form can be utilized: *system[i].systemA[j].systemB[k].summary.VAR* where *i*, *j*, and *k* are the index of the instance reference. Finally, if desired, parallel flow logic may be implemented for situations where computational performance is more important than model fidelity. If employed, the parallel logic treats an array of model system as a single representative model (i.e., without parallel logic an array may have 10 instances but with it included and enabled an array will be reduced to a 1 instance). This feature is built into the input specification and Modelica template library structure.

Figure 5 shows an example of a system model using the ExaDigiT Modelica template library. One of the advantages of this approach is that no matter which level in the hierarchy a system exists it has the same foundational components shown in the figure–i.e., structure, summary, inputs, control system, data, and the sensor bus. Everything beside those components are definable by the user for that system–e.g., alternative fluid ports, or none at all. Below is a description for each of these components.

- *structure*: instances of the same name as systemmodels used at that level. This component contains information that requires recompilation of a model such as the number of instances and a flag to enable a parallel model.
- *summary*: user-defined variables or calculations of interest to users of the model that are not readily accessible or desired to be highlighted–e.g., some type of calculation of many variables.
- *inputs*: replaceable model containing time-dependent variables for external access.
- *control system*: replaceable control system model for that model.
- *data*: replaceable data for containerizing information for that model.
- *sensorBus*: An expandable model for collecting signals for communicating to/from inputs, control systems, or other levels in the hierarchy.

This template approach directly informs the input specification development and is used by the automation process. Although AutoCSM does not require the use of this specific template approach, this library can also serve to accelerate the development of the system models themselves, as well as modified template approaches that may be more appropriate for a specific supercomputing facility. Additional features of the ExaDigiT Modelica Library are templates for testing system models for verification purposes. Leveraging the overall template approach and the practices demonstrated therein should provide significant efficiency gains for supercomputer facility modelers and analysts. To help orient a new user to using AutoCSM, a simplified Modelica library—*GenericCSM*—demonstrating the use of the template library is included with the AutoCSM source code. It is expected that new adopters of AutoCSM using Modelica would take that model and then adapt it to their system.

3.4 Example AutoCSM Workflow

Figure 6 provides a step-by-step pseudo-code workflow of AutoCSM's use, along with a description of each step. The current version of AutoCSM is written in Python and adopts a RedFish-style (*REDFISH | DMTF* n.d.) JSON input specification. The FMU is generated using Dymola's Python API, and simulation of the generated FMU is performed via FMPy (CATIA-Systems 2024). To reiterate, the use of tool-specific choices (i.e., Dymola) is not dictated by AutoCSM.

4 Frontier AutoCSM Demonstration

The Frontier CSM detailed in (Kumar et al. 2024) was modified and updated to use the ExaDigiT Modelica template library as described above and AutoCSM API for model generation. This section will first briefly discuss key aspects of that conversion and FMU auto-generation process. The ability to then extend this approach to extension of that base model to a preliminary exploration of

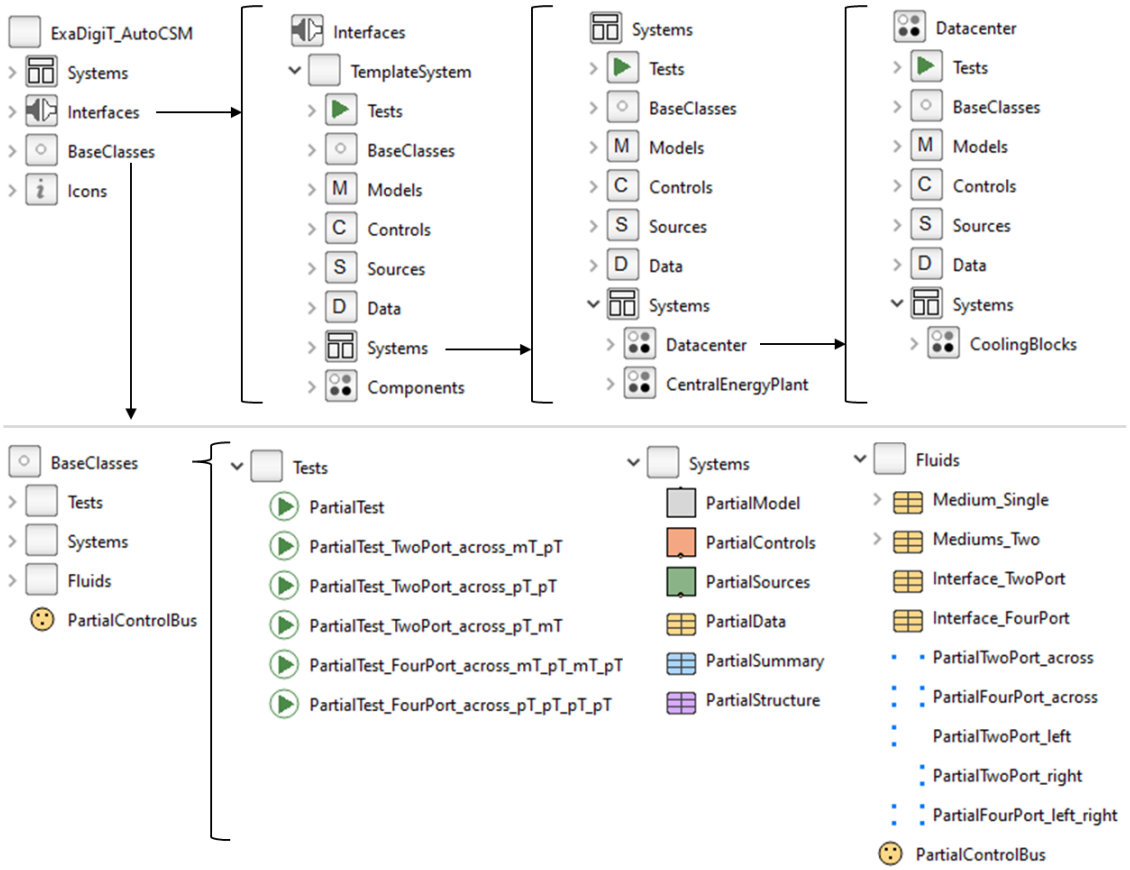


Figure 4. ExaDigiT Modelica template library. All systems use a common *TemplateSystem* that is then modified for the specific system. The *TemplateSystem* relies on replaceable components built on the (BaseClasses) models.

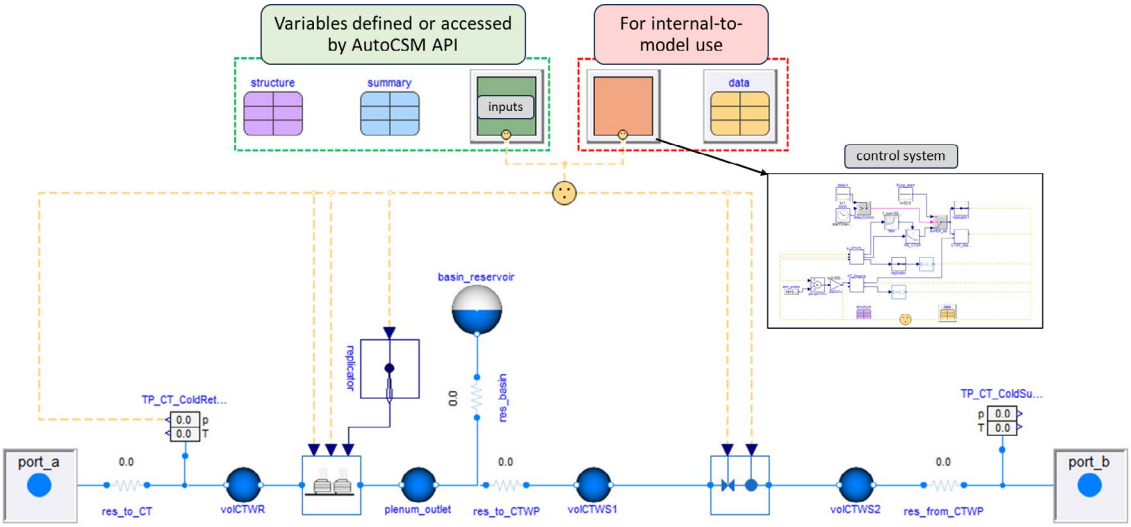


Figure 5. Example of a system model using the ExaDigiT Modelica template library. This is the Frontier AutoCSM model cooling tower loop. Every system model has a common set of components for use by the AutoCSM API or for internal-to-model usage.

two parallel datacenters with a single heat rejection system will be presented.

4.1 Conversion of Frontier to AutoCSM

The conversion process of taking the pre-AutoCSM Frontier model and converting it to the AutoCSM approach involved four general steps. In each of these steps, as-

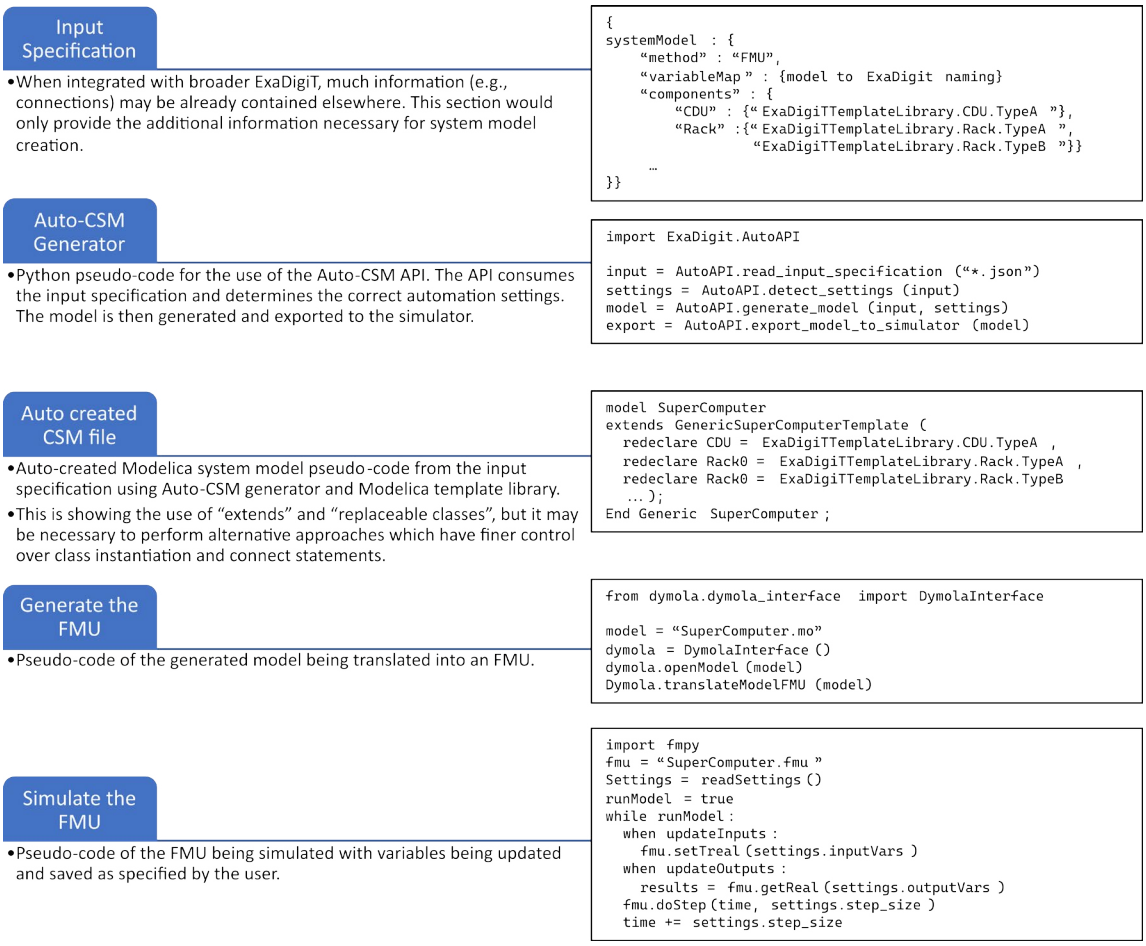


Figure 6. Pseudo-code workflow of the AutoCSM workflow with descriptions of each of the steps.

sociated variables for control, summary, or external input where also updated accordingly. Once all steps were complete, the Frontier model was properly formatted into a hierarchical structure Figure 8 and accessible by AutoCSM for the auto-generation of the FMU.

1. Identify and convert models to arrays and then create template-based *System* packages accordingly–e.g., the compute blocks that make up the datacenter Figure 7.

2. Break models into compartmentalized sections and convert, as with the first step, or turn into component-only models.

3. In parallel two the previous two steps, create test simulations for dynamic and steady-state to verify the models returns the expected results. Typically these tests only use a small number of instances of a particular system model to be tested sufficient to verify the behavior and performance–e.g., 2 compute blocks instead of 25. The input JSON file facilitates creation of the complete model.

4. Improve numerical or structural issues uncovered via the tests to reduce or eliminate numerical issues (e.g., numerical Jacobians and non-linearities).

4.2 Frontier to AutoCSM FMU

A nested hierarchical modeling approach for Modelica was implemented in AutoCSM API. Therefore, with the Modelica model updated to this approach the input JSON specification was created that reflects the desired overall model structure including instances of each model and parallel logic flags Figure 10. The JSON file is then processed and an FMU is generated using the AutoCSM Python API Figure 10. If needed, the intermediate *.mo* file generated in this process is accessible and can be loaded into a Modelica IDE for simulation, debugging, or manual FMU generation. After the Frontier model was converted to the AutoCSM approach, and various numerical issues were resolved, the simulation time was cut approximately by one-third while significantly improving the tractability and robustness of the model.

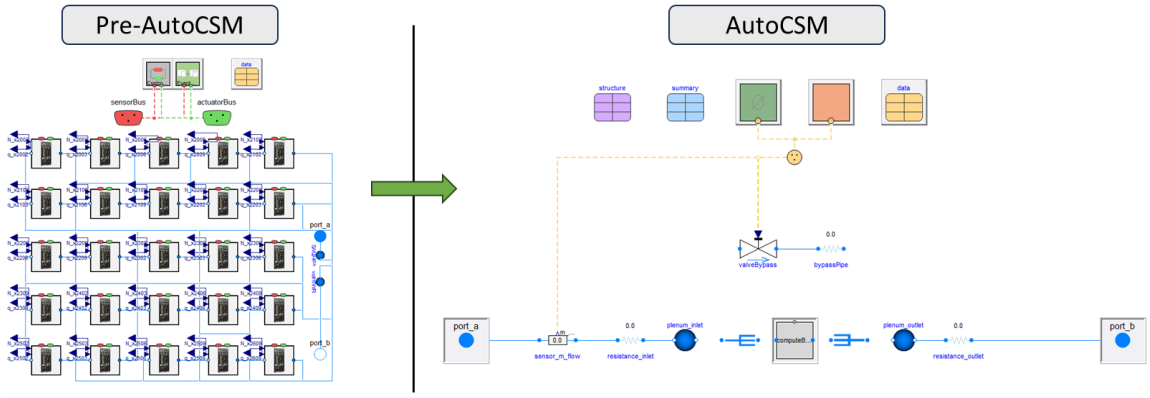


Figure 7. Example of the datacenter portion of the Frontier CSM before and after using the ExaDigiT Modelica AutoCSM template library where the compute blocks were able to be modified to an array.

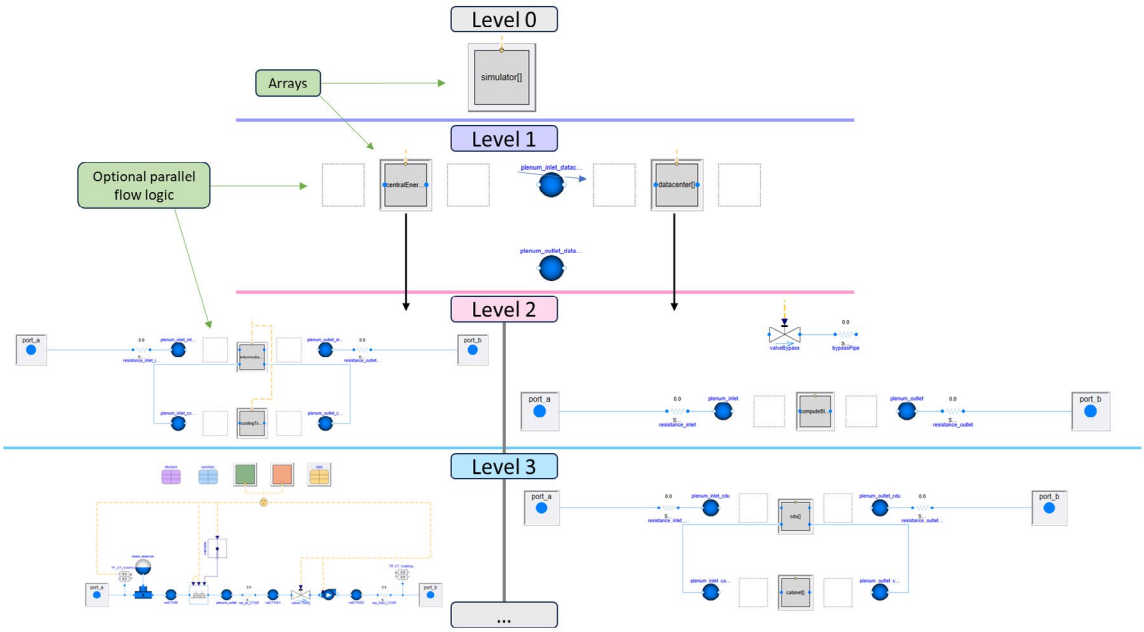


Figure 8. Subset of the Frontier CSM model implemented using the ExaDigiT Modelica AutoCSM template library demonstrating the nested hierarchy of the model structure. The dots at the bottom indicate that the levels of the hierarchy may continue as needed.

4.3 Extension to Alternative Studies

The migration to an AutoCSM approach enabled the exploration of a secondary test at Oak Ridge National Laboratory (ORNL). ORNL is beginning to assess whether its heat rejection system can support both Frontier and the next flagship supercomputer simultaneously. Although a detailed performance analysis of this system is beyond the scope of this work, it's worth discussing how AutoCSM could be adapted for that study.

A key requirement for this effort is that the new supercomputer will be structurally differ from Frontier. Since Modelica doesn't support arrays of replaceable models from different classes, a second datacenter was added in parallel to Frontier instead of simply increasing the instance number by one. The datacenter-level structure component was modified to include this addition. With the new model and associated sub-models created, the JSON file was updated to include a new *Datacenter* system under the *ORNLSupercomputing* node, named *datacenter_new*. The entries for the new datacenter were populated, and an FMU was generated without changes to the Python code.

This exercise demonstrated the flexibility of the AutoCSM approach to meet the needs of various datacenter studies with minimal effort. Originally designed for a supercomputer facility, AutoCSM is also likely applicable to most datacenter modeling activities and could likely serve as a starting point for any system modeling effort with a hierarchical architecture.

Figure 9. Part of the Frontier input JSON specification file for AutoCSM used to auto-generate the FMU.

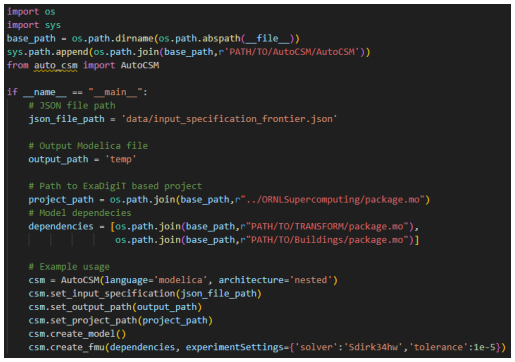


Figure 10. The python code required to process the input JSON file to create a Frontier FMU.

5 Potential Questions

During the exploration and development of AutoCSM, several common questions were identified, especially from the perspective of a potential user unfamiliar with Modelica. Below is a non-exhaustive list of questions that may be relevant to a potential AutoCSM user. While some of these questions or their answers are addressed in part within this paper, they are repeated here for completeness.

What is the value of an AutoCSM API in ExaDigiT? Creating CSMs is a time-consuming process that typically requires a deep understanding of the facility and expertise across various domains, such as thermal-hydraulics and controls. Additionally, integrating the model demands another specialized skill set, further complicating CSM development. AutoCSM accelerates this process by offering a step-by-step framework that allows for greater compartmentalization between the CSM developer and the user (in ExaDigiT). Another key time-saving benefit of AutoCSM lies in its support for exploratory studies, from analyzing subsystems at different levels of fidelity to exploring what-if scenarios to understand how design or operational changes might affect facility performance. Ultimately, time savings is the core value of AutoCSM.

What is the value of having a CSM in ExaDigiT? The benefit of having a CSM in ExaDigiT is to allow a user to understand how all aspects of their facility interact—for example, how job loading, facility cooling, and facility usage all respond to each other. This type of information can then be leveraged for exploration and optimization during all facility life cycles—across the processes of design, upgrades/downgrades, and operation. Thus, a CSM within ExaDigiT enables improvements and changes in facility design and operational efficiency that otherwise may not be possible.

If a user's digital twin needs do not require the use of the same or all subsystem levels as those of the template example library, how will the template system handle this scenario? The template system assumes top-down assembly of the CSM, allowing users to abstract lower-level facility components. The automation process adapts to the user's desired level of detail. For instance, if individual blades don't need modeling, the user can specify this in the input, omitting lower-level template subsystem models. The CSM modeler must only ensure that the models necessary for an ExaDigiT study are included.

As many Modelica integrated development environments (IDEs) are commercial software, is there concern that the template library will become tool specific? Although this work will use the commercial Modelica IDE Dymola to accelerate development, the proof-of-concept development will use pedantic mode to strictly enforce language standards such that the library should be usable with any Modelica-compliant IDE (e.g., Dymola, Modelon Impact, OpenModelica, ANSYS Twin Builder). Although this requirement will not be enforced upon users of ExaDigiT, the template library and any components used to create the example facility will be cross-tool compliant. To satisfy this requirement, components from existing libraries will be used, if possible; otherwise, modified components that satisfy the requirement will be created. However, this work prioritizes the input specification and API over the Modelica library, so the initial version will only include essential model development.

Is the limited availability of Modelica expertise a concern in AutoCSM value and adoption? While the methodology is language-agnostic, Modelica is chosen for initial demonstration due to its established capabilities. Alternatives like Julia's ModelingToolkit may be viable in the future, but uncertainties exist regarding problem size handling, solver availability, language stability, and domain-specific libraries for CSM-relevant dynamic system modeling. Lessons learned from Modelica are ex-

pected to be transferable to other approaches. More generaly, Modelica has a proven track record of signficantly decreasing model development time as compared to other approaches. Therefore, with AutoCSM and other open libraries available to users, it is expected that the creation of CSM subsystems for use in ExaDigiT, even by novice Modelica users, will not be a barrier to using ExaDigiT.

What is the advantage to using Modelica over alternative modeling languages? The advantages of using Modelica for this type of application derives from the maturity of the Modelica Language Standard and of the integrated development environments which implement it. The three primary relevant aspects is the importance of supporting *extends*, *replaceable*, and the language being acausal. The acausal nature assists in rapid and flexible model creation. The other two are foundational for creating architecture based implementations.

How will the AutoCSM process know how to model a user's facility? The template architecture provides the framework for connecting systems. This system-of-systems abstraction is therefore abstracted to a level where the method of defining the interfaces is the critical enabler for automating CSM creation. The architecture will not create the internal logic of individual subsystems. For example, the specific way to model the manner in which a blade or GPU is cooled will not be automated. The user must create the subsystem internal model by using Modelica components and then connect that internal model to the subsystem template.

6 Conclusions

The development of digital twins is a non-trivial endeavor. Methods that can help standardize and streamline the process for model development and incorporation into a framework used to operate a digital twin are critical. AutoCSM is one such methodology that aims to accelerate CSM development for integration into ExaDigiT's digital twin framework for supercomputing facilities. It strives to enhance speed, robustness, and the quality of results by enabling users to focus more on specific problems by abstracting out of the workflow, to the greatest extent possible, the infrastructure requirements for connecting models. This paper provides an overview of the AutoCSM methodology and workflow and provides an example overview of AutoCSM being used on the ORNL Frontier supercomputer facility. Future efforts in AutoCSM development will be driven by community adoption and feedback to the open source code base. Therefore, if AutoCSM is relevant to a potential user's needs, they are highly encouraged to provide feedback to the authors or directly via the code repository (`https://code.ornl.gov/exadigit/AutoCSM`).

Acknowledgments

This research was sponsored by and used resources of the Oak Ridge Leadership Computing Facility (OLCF), which is a DOE Office of Science User Facility at the Oak Ridge National Laboratory (ORNL) supported by the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

Atchley, Scott et al. (2023). "Frontier: Exploring Exascale". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: Association for Computing Machinery, pp. 1–16. DOI: 10.1145/3581784.3607089.

Brewer, Wesley et al. (2024-11). "A Digital Twin Framework for Liquid-cooled Supercomputers as Demonstrated at Exascale". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. To be published. Atlanta, GA.

CATIA-Systems (2024). *CATIA-Systems/FMPy*. https://github.com/CATIA-Systems/FMPy. Published 2017.

Functional Mock-up Interface (n.d.). https://fmi-standard.org/. Accessed April 19, 2024.

Greenwood, Scott et al. (2017a-08). *A Templated Approach for Multi-Physics Modeling of Hybrid Energy Systems in Modelica*. Technical Report 10.2172/1427611. DOI. URL: https://www.osti.gov/biblio/1427611.

Greenwood, Scott et al. (2017b-09). *TRANSFORM - TRANsient Simulation Framework of Reconfigurable Models*. DOI: 10.11578/dc.20171025.2022. URL: https://www.osti.gov/biblio/1503596.

Kumar, Vineet et al. (2024-10). "Thermo-Fluid Modeling Framework for Supercomputing Digital Twins: Part 1, Demonstration at Exascale". In: *Proceedings of the America Modelica Conference*. Storrs, CT.

Modelica Asscotiation (2024-05). *VehicleInterfaces Library*. https://github.com/modelica/VehicleInterfaces.

National Research Council (2005). *The Future of Supercomputing–Conclusions and Recommendations*. Washington, DC: The National Academies Press. DOI: 10.17226/11148.

REDFISH | DMTF (n.d.). https://www.dmtf.org/standards/redfish. Accessed April 19, 2024.


Modelica
Association