Linköping Studies in Science and Technology
Licentiate Thesis No. 111111

Methods and Tools for Efficient Model-Based Development of Cyber-Physical Systems with Emphasis on Model and Tool Integration

Alachew Mengist

LINKÖPING UNIVERSITY

Linköping University
Department of Computer and Information Science
Division for Software and Systems (SaS)
SE-581 83 Linköping, Sweden

Linköping 2018

This is a Swedish Licentiate's Thesis

Swedish postgraduate education leads to a doctor's degree and/or a licentiate's degree.
A doctor's degree comprises 240 ECTS credits (4 years of full-time studies).
A licentiate's degree comprises 120 ECTS credits.

Edition 1:1

© Alachew Mengist, 2018
ISBN 123456
ISSN 0280-7971
URL http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-456789

Published articles have been reprinted with permission from the respective copyright holder.
Typeset using XƎTEX

Printed by LiU-Tryck, Linköping 2018

ABSTRACT

Model-based tools and methods are playing important roles in the design and analysis of cyber-physical systems before building and testing physical prototypes. The development of increasingly complex CPSs requires the use of multiple tools for different phases of the development lifecycle, which in turn depends on the ability of the supporting tools to interoperate. However, currently no vendor provides comprehensive end-to-end systems engineering tool support across the entire product lifecycle, and no mature solution currently exists for integrating different system modeling and simulation languages, tools and algorithms in the CPSs design process. Thus, modeling and simulation tools are still used separately in industry.

The unique challenges in integration of CPSs are a result of the increasing heterogeneity of components and their interactions, increasing size of systems, and essential design requirements from various stakeholders. The corresponding system development involves several specialists in different domains, often using different modeling languages and tools. In order to address the challenges of CPSs and facilitate design of system architecture and design integration of different models, significant progress needs to be made towards model-based integration of multiple design tools, languages, and algorithms into a single integrated modeling and simulation environment.

In this thesis we present the need for methods and tools with the aim of developing techniques for numerically stable co-simulation, advanced simulation model analysis, simulation-based optimization, and traceability capability, and making them more accessible to the model-based cyber physical product development process, leading to more efficient simulation. In particular, the contributions of this thesis are as follows: 1) development of a model-based dynamic optimization approach by integrating optimization into the model development process; 2) development of a graphical co-modeling editor and co-simulation framework for modeling, connecting, and unified system simulation of several different modeling tools using the TLM technique; 3) development of a tool-supported method for multidisciplinary collaborative modeling and traceability support throughout the development process for CPSs; 4) development of an advanced simulation modeling analysis tool for more efficient simulation.

Acknowledgments

Acknowledgments.tex

Contents

List of Figures

List of Tables

Acronyms

BEAST BEAring Simulation Toolbox. 14

CPSs Cyber-Physical Systems. 6–9, 12, 13, 16, 18, 19

DAE Differential-Algebraic Equation. 26

DAEs Differential-Algebraic Equations. 24

EOO Equation-based Object-Oriented. 5, 7, 10, 12, 13

FFT Fast Fourier Transform. 22

FMI Functional Mock-up Interface. 15, 18–20

FMU Functional Mock-up Unit. 19, 20, 49, 52, 56, 58

JSON JavaScript Object Notation. 55, 58

MSL Modelica Standard Library. 14

OCP Optimal Control Problems. 7, 21

OMC OpenModelica Compiler. 50, 73

OSLC Open Services for Lifecycle Collaboration. 50, 51, 54–58

SysML Systems Modeling Language. 16, 19, 54, 58

TLM Transmission Lines Modeling. 8, 10, 12, 15, 20, 36, 37, 39, 40, 42, 44, 46

UML Unified Modeling Language. 16, 17, 19

XML eXtensible Markup Language. 36, 37, 42, 50–52, 54, 56, 57

Model-based tools and methods are playing important roles in the design and analysis of cyber-physical systems before building and testing physical prototypes. The development of increasingly complex CPSs requires the use of multiple tools for different phases of the development lifecycle, which in turn depends on the ability of the supporting tools to interoperate. However, currently no vendor provides comprehensive end-to-end systems engineering tool support across the entire product lifecycle, and no mature solution currently exists for integrating different system modeling and simulation languages, tools and algorithms in the CPSs design process. Thus, modeling and simulation tools are still used separately in industry.

The unique challenges in integration of CPSs are a result of the increasing heterogeneity of components and their interactions, increasing size of systems, and essential design requirements from various stakeholders. The corresponding system development involves several specialists in different domains, often using different modeling languages and tools. In order to address the challenges of CPSs and facilitate design of system architecture and design integration of different models, significant progress needs to be made towards model-based integration of multiple design tools, languages, and algorithms into a single integrated modeling and simulation environment.

In this thesis we present the need for methods and tools with the aim of developing techniques for numerically stable co-simulation, advanced simulation model analysis, simulation-based optimization, and traceability capability, and making them more accessible to the model-based cyber physical product development process, leading to more efficient simulation. In particular, the contributions of this thesis are as follows: 1) development of a model-based dynamic optimization approach by integrating optimization into the model development process; 2) development of a graphical co-modeling editor and co-simulation framework for modeling, connecting, and unified system simulation of several different modeling tools using the TLM technique; 3) development of a tool-supported method for multidisciplinary collaborative modeling and traceability support throughout the development process for CPSs; 4) development of an advanced simulation modeling analysis tool for more efficient simulation.

1 Introduction

1.1 Motivation

Modeling and simulation of Cyber-Physical Systems (CPSs) are becoming increasingly important in many engineering applications. The ability to build mathematical models and simulate their behavior enables engineers to virtually analyze a system without conducting experiments on the real system, which would otherwise be too expensive, risky, or time-consuming. As a result, the design can be thoroughly tested, evaluated and optimized with different concepts before building physical prototypes. This, in principle, enables systems engineers to identify a wider set of early system-level mistakes and increase the quality of products.

The interest in Equation-based Object-Oriented (EOO) languages and tools has dramatically increased in industry during recent years because of their increasing importance in modeling, simulation, and specification of complex systems. There are several different EOO modeling languages and tools available today for mathematical modeling and simulation of large and heterogeneous physical systems. Some examples of such systems modeling and simulation languages and tools are Modelica [9], MSC.ADAMS [112], SKF's BEAST(BEAring Simulation Toolbox) [113], VHDL-AMS [28, 52], Simulink/Simscape [74], SysML [54, 99], and ModelicaML [100, 121], etc. Such languages open up the possibility of modeling multi-disciplinary dynamic complex systems through reusable model components, and generating simulation code for a number of different platforms. However, in spite of the similarities between modeling and simulation tools and languages, it is often difficult to use such EOO languages and tools in combination.

While modeling and simulation remain important tools for engineers in many disciplines, the landscape is shifting towards a more flexible and diverse use of model-based design methodologies. Many system engineers depend heavily on model-based design and control of dynamic complex CPSs involving different science and engineering disciplines. Of paramount importance is the ability to capture all central aspects of such systems in the models as an executable specification that reflects the evolution of a system, including the physical behavior of the system components and the architectural description of its software and hardware.

This trend raises new demands on associated tools. In particular, a model-based design and control process includes several activities, such as model checking, simulation, parameter sensitivity analysis, impact analysis, formal verification, design optimization, control analysis and synthesis, and state estimation and control system development and deployment, etc., to capture different aspects of dynamic systems modeling.

However, currently no vendor offers comprehensive whole-life-cycle systems engineering tool support for CPSs, and no mature solution exists that can integrate different tools together, thus modeling and simulation tools are still used separately in industry. Accordingly, flexibility, interoperability, and traceability are key success factors for algorithms and modeling tools in order to meet future challenges. Thus, it becomes very important to integrate modeling and simulation of several different tools including dynamic optimization, co-modeling, co-simulation, simulation analysis, and traceability.

1.2 Problem Formulation

There is currently an increasing need to integrate different system modeling and simulation languages, tools, and algorithms into the engineering design process. The unique challenges in CPSs integration emerge both from the increasing heterogeneity of components and interactions, and the increasing size of systems. To address increasing complexity and challenges of CPSs, significant progress needs to be made towards model-based integration of multiple design tools into a single modeling and simulation environment, where co-simulation, analysis, optimization, and traceability can be performed, among other things, for more efficient simulation.

Mathematical models are extensively used in different advanced engineering application areas (e.g., multi-body system dynamics, electronic circuit simulation, chemical process engineering), as a standard design methodology. Traditionally, EOO Languages and Tools (EOOLT) mainly focus on model simulation. However, model simulation is not the only objective of mathematical modeling. In recent decades there has been a strong trend towards using models to solve optimization problems, such as optimal control problems (OCPs). This allows for more efficient and automatic solution of dynamic optimization problems, as simulation models are reused for optimization. While there are several tools restricted to a particular modeling domain that support dynamic optimization, very limited toolboxes are available for optimizing heterogeneous physical systems. Thus, the first problem we tackled in this thesis is that,

Problem 1: Need to support dynamic optimization algorithms in the model-based development process of heterogeneous CPSs.

While simulation of dynamic systems remains an important tool for engineers, further analysis and post-processing for simulation results is required for most applications of models, e.g., OCP and linear models. Although the EOO modeling language Modelica is an industry-standard high-end modeling language and exchange format, the lack of tools for advanced analysis of models has been a weakness of the language compared to scripting languages. Scripting languages such as MATLAB [1] and Python [2] provide most of the desirable analysis capabilities such as control of simulation, linear model analysis, model sensitivity, optimization capability, and advanced post-processing of simulation results etc. In order to exploit the full potential of Modelica, either a Modelica supporting tool should be integrated with a scripting language for model analysis, or Modelica should be extended to a scripting language. Thus, the second problem we tackled in this thesis is that,

Problem 2: Need to support advanced simulation modeling analysis for more efficient simulation.

In the area of modeling and simulation of heterogeneous CPSs, domain experts use specialized modeling and simulation environments for sub-models of the system. These tools are also favored, as they usually offer larger libraries of sub-models of the system and more domain-specific features than a multi-domain tool. In reality, the different parts of the system represented by these sub-models are often physically tightly coupled and interdependent. However, problems may arise when trying to combine the different sub-models of the system model into a single, coupled system model simulation from various tools, as the simulation sub-models are likely to use different differential equation solvers with variable time step. Hence, numerical stability, which is not an issue for discrete time simulations, becomes an important consideration since there is a certain physical communication delay between different simulation sub-models of the system. Thus, the problem is still how to connect the sub-models into a single simulation model and perform the necessary

[1] https://se.mathworks.com/products/matlab.html
[2] https://www.python.org/

evaluations to compute the solution for a more complete and exact system analysis.

One earlier method that was used to de-couple such simulation sub-models and allow them to be independently simulated and coupled in a numerically stable way via co-simulation techniques is Transmission Lines Modeling (TLM) [85, 84, 29, 61]. TLM uses physically motivated time delays to separate the sub-models in time and enable efficient co-simulation. The technique has proven to be numerically stable and was implemented for coupling of hydraulic and mechanical sub-systems [85, 84]. However, no attempt has been made to design a general and convenient open-source co-modeling and simulation framework based on the TLM technique.

Problem 3: Need to integrate TLM based co-simulation which is an efficient and numerically stable simulation, and yet no general open-source tool exists that facilitates the task for modeling tool-specific simulation sub-models, connecting via TLM and co-simulating the complete system model.

A common situation in industry is that a system model is composed of several sub-models that may have been developed using different tools. The quality and effectiveness of large-scale system modeling depends heavily on the underlying tools that are used for different phases of the development lifecycle. Available modeling and simulation environments support specialized modeling of parts of the system model, and also support different operations on models, such as requirements modeling, model simulation, model checking, validation, and verification. Thus, seamless exchange of models in the context of different modeling tools is becoming increasingly important. However, due to the lack of interoperability between tools it is often difficult to use such tools in combination. Also, without support for tracing of the requirements and associating them with the models and the simulation results, the impact analysis, verification, and validation would be difficult. Hence, the problem to be dealt with is:

Problem 4: Need collaborative modeling of CPSs among different modeling and simulation tools and traceability of artifacts created throughout the whole system development process

1.3 Research Questions

Our overall goal is to design and implement tools and methods up to the point of a proof of concept and prototype demonstration, which would increase the efficiency and quality of the process of model-based development of CPSs. We formulate the following research question:

How can the features and functionality of computer-aided CPSs modeling and simulation tools be improved in order to make the process of model-based development of CPSs more efficient and usable in the product development process?

To answer the research question, we pursue five specific objectives:

- to ensure creating automatic links and traceability between requirements, simulation models, FMUs, and simulation results artifacts
- to ensure integration of product design tools with modeling and simulation tools and keep track of changes
- to ensure automatic solution of dynamic optimization problems by reusing simulation models for optimization
- to ensure reusing and combining existing simulation models formalized by different experts in different modeling languages by means of numerically stable co-simulation modeling
- to support advanced simulation modeling analysis

With our solution, we address the following properties: openness, generality, model portability, tool interoperability, model integration, model reusability, model evolution, and model maintenance.

1.4 Contributions

The work presented in this thesis makes the following primary contributions towards our ultimate goal of an integrated framework, ensuring high quality of simulation models, tool interoperability, and traceability of artifacts for efficient model-based development of CPSs:

- **We have contributed to the development of a model-based dynamic optimization approach by integrating optimization into the model development process.** Models and optimization algorithms are combined into an integrated model. Specifically, the systems to be controlled are formulated in Modelica, and the corresponding optimization problems are expressed in Optimica [3]. This allows for efficient optimization of heterogeneous physical systems as simulation models are reused for optimization.
- **We have contributed to the development of a versatile graphical co-modeling editor and co-simulation framework based on the TLM method.** This enables modeling, connecting, and simulation of several different modeling tools using the TLM co-simulation technique, which is a numerically stable and efficient simulation. **We have**

also developed a schema for standardizing the structure and validation constraints of a composite model.

- **We have contributed to a tool-supported method for multi-domain collaborative modeling and traceability support throughout the developments in CPSs.** This enables recording and establishing the traceability links of model elements (e.g., requirements, activities, artifacts, modeling tools, simulation results, validation, verification) through a standardized interface and format using Open Services for Lifecycle Collaboration (OSLC). The artifacts processed and generated by a toolchain are stored in a global data repository that supports version control and enables traceability at all stages of development. **We have also developed a schema for standardizing the structure and validation constraints of traceability data that can be used by several simulation and requirements modeling tools.**

- **Increased EOO simulation modeling analysis,** helping the user for controlling simulation models and automatically analyzing simulation results using various packages in Python, like a Fast Fourier Transform (FFT) analysis to improve simulation models for more efficient simulation. We have extended the list of simulator plugins for the automatic simulation results analysis tool, PySimulator [1], by Wolfram System-Modeler [3] simulator. We have also extended OMPython [46], which enables better integration with Python for simulation and analysis of EOO Modelica models.

1.5 List of Publication

The work presented in this thesis is based on the following publications:

[103] **Alachew Shitahun**, Vitalij Ruge, Mahder Gebremedhin, Bernhard Bachmann, Lars Eriksson, Joel Andersson, Moritz Diehl, and Peter Fritzson. **Model-Based Optimization with OpenModelica and CasADi.** In Proceedings of IFAC Conference in Tokyo, September 2013.

[104] **Alachew Shitahun**, Vitalij Ruge, Mahder Gebremedhin, Bernhard Bachmann, Lars Eriksson, Joel Andersson, Moritz Diehl, Peter Fritzson. **Tool Demonstration Abstract: OpenModelica and CasADi for Model-Based Dynamic Optimization.** In Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, Nottingham, UK, April 19, 2013.

[3] http://wolfram.com/system-modeler/

[68] Bernt Lie, Sudeep Bajracharya, **Alachew Mengist**, Lena Buffoni, Arun Kumar, Martin Sjölund, Adeel Asghar, Adrian Pop, Peter Fritzson. **API for Accessing OpenModelica Models From Python.** In Proceedings of 9th EUROSIM Congress on Modeling and Simulation, September 12-16, 2016, Oulu, Finland.

[8] Adeel Asghar, Andreas Pfeiffer, Arunkumar Palanisamy, **Alachew Mengist**, Martin Sjölund, Adrian Pop and Peter Fritzson. **Automatic Regression Testing of Simulation Models and Concept for Simulation of Connected FMUs in PySimulator.** In Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015.

[76] **Alachew Mengist**, Adeel Asghar, Adrian Pop, Peter Fritzson, Willi Braun, Alexander Siemers and Dag Fritzson. **An Open-Source Graphical Composite Modeling Editor and Simulation Tool Based on FMI and TLM Co-Simulation.** In Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015.

[77] **Alachew Mengist**, Adrian Pop, Adeel Asghar, Peter Fritzson. **Traceability Support in OpenModelica Using Open Services for Lifecycle Collaboration (OSLC).** In Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017.

[27] Lena Buffoni, Adrian Pop, **Alachew Mengist**. **Traceability and impact analysis in requirement verification.** In Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, Munich, Germany, December 1, 2017.

The publications listed above correspond to the chapters of this thesis as follows. The work on model-based dynamic optimization presented in [103] and [104] are covered in **Chapter 3**. The TLM-based approach for co-modeling and simulation framework developed in [76] is presented in **Chapter 4**. The approach to collaborative modeling and traceability support introduced in [77] is elaborated in **Chapter 5**. The tools for control of simulation models and advanced analysis of simulation results developed in [68] and [8] are described in **Chapter 6**. The work in progress for seamless tracing of Modelica-based requirements modeling and verification given in [27] is discussed in **Chapter 7**, as a future work.

1.6 Thesis Outline

The thesis is organized as follows:

Chapter 2 – Background and Related Work provides the basic concepts relevant to understanding the rest of this thesis. In particular, we present an overview of the state-of-the-art EOO modeling languages and simulation tools, and co-simulation technologies for CPSs.

Chapter 3 – Integration of Optimization toolchain into the Model-Based Development Process introduces the complete toolchain for model-based dynamic optimization together with industrially-relevant applications that are solved by the toolchain.

Chapter 4 – TLM-based Co-modeling Editor and Co-simulation Framework introduces our general approach for modeling of a composite model containing several tool-specific simulation models which can be integrated, connected and simulated based on the TLM co-simulation technique.

Chapter 5 – Collaborative Modeling and Traceability in the Model-Based Design of CPSs presents our tool-supported method for collaborative modeling of CPSs from requirements to models and simulation results, and tracing artifacts created during the whole system development process.

Chapter 6 – Advanced Simulation Modeling Analysis presents modeling and simulation tool interoperability and communication with Python for controlling simulation models and advanced analysis of simulation results for more efficient simulation.

Chapter 7 – Conclusions and Future Work concludes the work presented in this thesis and discusses possible directions for future work.

The thesis also contains an appendix in which we provide parts of contributions for standardizing the structure and validation constraints of composite modeling and traceability information.

2 Background and Related Work

Cyber-Physical Systems (CPSs), systems composed of closely-coupled computing and physical elements, are characterized by a complex architecture and a design process involving different science and engineering disciplines. Model-Based Design (MBD), emphasizes mathematical modeling to design, analyze, verify, and validate dynamic systems, and has been identified as a powerful design technique for CPSs [16, 25, 63]. However, due to the intrinsic heterogeneity and complexity of CPSs, a large number of modeling languages and tools have been utilized to address different aspects of the system such as physical processes and requirements management.

In the previous decade, several EOO languages used for mathematical modeling of heterogeneous dynamics of complex CPSs (e.g., automobiles, aircraft, and powerplants), have gained considerable attention from both industry and academia. Today the state of the art within multi-domain physical modeling (e.g., containing mechanical, electrical, hydraulic, thermal, fluid, and control components) is Modelica [9], which is an EOO modeling language for declarative mathematical modeling of large physical systems [36, 43, 42, 116].

Other examples of languages with similar modeling and simulation capabilities are gPROMS [37, 17, 82, 18] for chemical engineering, VHDL-AMS [28, 52], which is a hardware description language (HDL) for modeling of modern analog and mixed-signal designs, ADAMS [112] in the domain of mechanical modeling, SKF's BEAST [113] for simulation of the dynamics of rolling bearing models with detailed contact definitions, and control systems in Simulink

[74]. High level UML-based languages such as SysML [54, 99] and ModelicaML [121, 100, 122, 101] have also been utilized for modeling CPSs for design phases such as simulation and verification.

Recently, co-simulation technologies such as Functional Mock-up Interface (FMI) [78, 21] and Transmission Line Modeling (TLM)-based co-simulation techniques [85, 29, 61, 84] have also been utilized for modeling and simulating individual components of complex distributed CPSs, using different simulation tools simultaneously and collaboratively. Thus, system engineers can use different simulation tools together to create virtual prototypes of entire Cyber-Physical Systems.

2.1 Modelica

Modelica [9, 42, 43] is a freely available, object-oriented, declarative and equation-based language for component-oriented modeling of large, complex, and heterogeneous systems. It is suited for component-oriented multi-domain modeling of physical systems, for example, systems containing electrical, electric power, mechanical, hydraulic, thermal, control, state machine subsystems, or process-oriented subcomponents. The open standard Modelica language specification is developed by a non-profit organization, the Modelica Association [10]. The Modelica Association also develops the open-source Modelica Standard Library (MSL) [11] with a large set of models. MSL version 3.2.2 [9, 11] contains about 1600 model components and 1350 functions from different application domains. Libraries of model components are described by object diagrams, which consist of connected components. Components are connected by ports and are defined by subcomponents or a textual description in the Modelica language based on standardized interface definitions. Models in Modelica are built on acausal modeling and object-oriented constructs with mathematical equations described by differential, algebraic, and discrete equations to facilitate exchange and re-use of models. Thus, no particular variable needs to be solved for manually. A Modelica tool will have enough information to automatically determine the computational solution order and generate efficient simulation code.

Modelica, which is suited for hardware-in-the-loop simulations and for embedded control systems, is increasingly used for model-based development in industry. In particular, many automotive companies, such as Audi, BMW, Daimler, Ford, Toyota, and VW use Modelica to design energy efficient vehicles and/or improved air conditioning systems. Also, power plant providers, such as ABB, EDF, and Siemens use Modelica, as do many other companies.

A number of commercial and open-source simulation environments are available that support modeling with Modelica, such as Wolfram SystemModeler [73, 75, 44], Dymola [114, 26], SimulationX [49], MapleSim [72], OpenModelica [31, 41] JModelica.org [2], and more.

2.2 OpenModelica

OpenModelica [31, 41] is an extensible Modelica-based open-source framework for modeling, simulation, and analysis of dynamic systems intended for research, teaching, and industrial usage. It is a result of research at the Programming Environments Laboratory (PELAB), Linköping University, and its long-term development is supported by a non-profit organization, the open-source Modelica Consortium (OSMC) [30]. The main objective of the OpenModelica effort is to create a flexible and comprehensive Modelica modeling, compilation, simulation and systems engineering environment for research, teaching, and industrial usage.

The OpenModelica environment consists of several subsystems. An advanced interactive OpenModelica Compiler (OMC) performs the translation of Modelica models to C code, which is compiled and executed to simulate the model. Textual and graphical model editing, including browsing of the Modelica standard library, simulating, analyzing simulations, and presenting documentation is performed using the OpenModelica connection editor (OMEdit). The OpenModelica Notebook (OMNotebook) provides a tutorial for Modelica, and Modelica models, together with documentation and pictures, can be written and simulated in it. Debuggers for equation-based modeling and for algorithmic subsets of Modelica are supported in OpenModelica. OpenModelica currently supports FMI 1.0 and FMI 2.0 for model exchange and most of FMI 2.0 for co-simulation. Other tools can access this functionality by dynamically linking OMC or by invoking it using a message-passing interface. For more information, see the openmodelica.org[31] home page.

Recent research developments of the OpenModelica environment reported in this thesis include co-modeling and simulation based on TLM, model-based dynamic optimization integration with CasADi [7] for solving large-scale optimization problems, seamless tracing of the requirements and associating them with the models and the simulation results, and better interoperability with Python for advanced modeling simulation analysis.

2.3 Model-Based Systems Engineering (MBSE)

Model-Based Systems Engineering (MBSE) is defined in [58] as “the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities, beginning in the conceptual design phase and continuing throughout development and later life cycle phases.” An overview of existing methodologies used in industry is given in [38]. Some of these methodologies use standardized languages, such as UML [53], or SysML [54] for system modeling. More recently, there has been increased focus on using a system model as an executable specification to numerically evaluate the dynamic behavior of complex systems throughout the development process. For example, ModelicaML [121, 100], an extended SysML-based profile for system modeling, can be combined with Modelica to enable the modeling and simulation of complex heterogeneous CPSs. An example of model-based design methodology for CPSs is given in [60].

2.4 UML-Based System Modeling Languages

The Unified Modeling Language (UML) [53] is a general-purpose visual modeling language for architectural modeling, design, and implementation of complex software systems, both structurally and behaviorally. The static structure of a software system is captured in UML through a combination of class diagrams, and/or composite structure diagrams. The dynamic behavior of a software system is captured in UML-based specialized languages (e.g., SysML[54, 99] and ModelicaML[121, 100] through a combination of sequence diagrams, activity diagrams, and/or state machine diagrams and their own specific extensions, using a UML profile to adapt the language to a particular domain or purpose.

2.4.1 SysML

The Systems Modeling Language (SysML) [54] is a graphical modeling language that is based on UML for Systems Engineering.It is defined as an extension of a subset of the UML, using UMLś profile mechanism. Through these extensions, SysML supports the specification, analysis, design, verification, and validation of complex systems that may include hardware and software. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametrics, which are used to integrate with other engineering analysis models. However, SysML models are not executable in a manner that allows them to interoperate with other model simulation and analysis tools. An extended version of SysML to support modeling of complex heterogeneous CPSs is given in [60].

2.4.2 ModelicaML

Modelica Modeling Language (ModelicaML) [121, 100] is a graphical modeling language for the description of continuous-time and discrete-time/event-based system dynamics. ModelicaML is defined as an extended subset of UML and a language extension for Modelica. This subset enables the generation of executable Modelica code from graphical models.

ModelicaML extends the graphical modeling capabilities of Modelica by providing more diagrams (UML diagrams for presenting the composition, connection, inheritance, or behavior of classes) for graphical model definition or documentation. Parts of the system model are entered as text (i.e., Modelica equations or algorithmic code, modification and declaration expressions).

Moreover, ModelicaML supports a method for formalizing and verifying system requirements using simulations via the vVDR[1] (Virtual Verification of Designs against Requirements) method.The vVDR method (See Chapter 3 in [101, 122]) enables model-based design verification against system requirements.

2.5 Modeling and Simulation Tools

2.5.1 Wolfram SystemModeler

Wolfram SystemModeler [73, 75, 44] is a commercial platform, developed by Wolfram MathCore [Wolfram MathCore Engineering AB, 2007], for modeling and simulation of Modelica models based on an older version of the OpenModelica compiler front-end. It provides an interactive graphical modeling editor, simulation, and plotting environment and a customizable set of component libraries. In addition, SystemModeler has a link to Mathematica [125], which enables further analysis and transformation of Modelica models.

2.5.2 MATLAB/Simulink

Simulink [74] is an extension to MATLAB for graphical modeling, simulation, and model-based design of multi-domain dynamic systems. Mathematical models representing physical systems are represented graphically in Simulink as block diagrams. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. Simulink is tightly integrated with the MATLAB environment, enabling users to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis. It is widely used in automatic control and digital signal processing for multi-domain simulation and Model-Based Design [23, 95]. Compared to Modelica, Simulink only supports causal modeling, and a Simulink model for a specific data flow context cannot be reused for another data flow context.

2.5.3 Dymola

Dymola [114, 26], developed by Dassault Systemes[2], is a commercial modeling and simulation tool based on the Modelica modeling language for model-based design of complex systems. In addition, Dymola supports FMI 1.0 and 2.0 for model exchange and co-simulation, model calibration, parameter optimization, and real-time simulation for a wide range of hardware-in-the-loop platforms.

[1] https://github.com/lenaRB/VVDRlib
[2] https://www.3ds.com/products-services/catia/products/dymola

2.5.4 20-Sim

20-sim[3] is a commercial tool for modeling and simulation of mechatronic systems. It is widely used to simulate and analyze the behavior of multi-domain dynamic systems and the development of control systems [32, 33, 24]. The 20-sim tool can represent continuous time models using equations, block diagrams, physical components, and bond graphs. Bond graphs [47], a domain-independent description of a physical system's dynamics, implement such connected blocks.

2.5.5 Overture

Overture[4] is an open-source integrated development environment for modeling and analyzing VDM (The Vienna Development Method)[5] models. Typically, it supports [118] the design of discrete systems using VDM-RT (VDM-Real Time)[6] dialect, including both time and distribution of functionality on different computational nodes for CPSs.

2.5.6 ADAMS

ADAMS (Automated Dynamic Analysis of Mechanical Systems) [112] is a multibody dynamics simulation environment for building, simulating, and refining models of mechanical systems. It is equipped with Fortran and C++-based numerical solvers.

2.5.7 Modelio

Modelio [111] is an open-source modeling tool supporting industry standards like UML and its extensions SysML for high-level system architecture modeling. Modelio extends the SysML language [54, 99] and proposes extensions for CPSs modeling, enabling the specification of several aspects of the system from requirements to the hardware/software architecture through use case specification, and system functional design. In particular, requirement, FMI interface, FMU connections, automatic documentation generation, and impact analysis are supported.

2.5.8 RT-Tester

RT-Tester [50, 88] is a test automation tool for automatic test generation, test execution, and real-time test evaluation. The RT-Tester Model-Based Test

[3] http://www.20sim.com/

[4] http://overturetool.org/

[5] http://overturetool.org/method/ - Model-oriented formal methods for the development of computer-based systems and software

[6] http://overturetool.org/download/examples/VDMRT/ - A real-time dialect of the VDM formal modeling language

Case and Test Data Generator(RTT-MBT) [50, 87] supports model-based testing: automated generation of test cases, test data, and test procedures from UML/SysML models. Additionally, it generates tests as FMUs that are executed against the system under test, and also generates traceability data that relates requirements, test cases, test procedures, and results.

2.5.9 BEAST

BEAST (BEAring Simulation Tool) [113], developed by SKF[7], is a simulation program that enables SKF engineers to perform simulations of bearing dynamics on any major bearing types. This enables studies of internal motions and forces in a bearing under virtually any load condition. The model is fully three-dimensional, solving the general differential equations of motion for all components; all components have six degrees of freedom. External forces and moments can be applied to all bearing components except the rolling elements. Most bearing types can be modeled.

2.5.10 PySimulator

PySimulator [1] is a simulation and analysis environment in Python with plugin infrastructure. The environment provides a graphical user interface for simulating different model types (currently Functional Mockup Units, Modelica Models and SimulationX Models), plotting result variables and applying simulation result analysis tools like Fast Fourier Transform. The modular concept of the software enables easy development of further plugins for both simulation and analysis.

2.6 Co-simulation Technologies

2.6.1 Functional Mock-up Interface (FMI)

The Functional Mock-up Interface (FMI) [78, 21] is an open and tool-independent standardized interface for exchange between tools and co-simulation of dynamic models. FMI defines a C interface that is implemented by an executable called a Functional Mock-up Unit (FMU). The idea is to allow tools to export pre-compiled system models containing the model description eXtensible Markup Language (XML) file and model equations in C-code or binary code, and to exchange models that comply with the FMI specification.

The FMI standard currently specifies two types of protocols: FMI for Model Exchange (import and export), and FMI for co-simulation. The main difference between these two protocols is that in Model Exchange the FMU is simulated using the importing tool's solver, while in co-simulation the FMU

[7] http://www.skf.com/se/index.html

is shipped with its own solver to couple two or more simulation tools in a co-simulation environment. The FMI standard is currently supported by over 106 modeling and simulation tools[8], for example Dymola[9], Wolfram System-Modeler[10], OpenModelica[11], and Jmodelica.org[12].

2.6.2 Transmission Line Modeling (TLM)

The transmission line element method (TLM) [85, 84, 29, 61] is a one-dimensional simulation technique for power transmitting systems. This means systems where the physical behavior can be modeled with intensity and flow variables, for example hydraulics, electrics, mechanics, and acoustics. The method, also known as bi-lateral delay line modeling [12], was used in transmission line modeling as described by [61]. The difference between TLM and other simulation methods based on centralized integration is that it uses time delays in the model to simulate how the information propagates through the system. Information propagation is thus simulated more accurately than with other methods, because physically motivated time delays are taken into account. This is especially useful when accurate wave propagation results are important. The use of time delays also means that all components are separated by some distance. There is no immediate communication taking place between components separated in time. This makes TLM ideal for parallel or distributed processing.

2.7 Dynamic Optimization

2.7.1 CasADi

CasADi [7] is an open-source framework for C++ and Python for numerical optimization in general and optimal control in particular. The main idea of the tool is to provide users with the ability to easily and efficiently implement optimal control algorithms with a wide range of methods, including multiple shooting and collocation, rather than providing users with a “black-box” OCP solver.

2.8 Related Work

Due to the influence of high-level equation-based modeling languages in the industrial community, there have been several attempts to integrate tools for such languages with numerical algorithms for optimization. However, most

[8] http://fmi-standard.org/tools/
[9] https://www.3ds.com/products-services/catia/products/dymola
[10] http://wolfram.com/system-modeler/
[11] https://openmodelica.org/
[12] http://www.jmodelica.org/

available tools usually only support a particular optimization algorithm. For example, Dymola supports parameter and design optimization of models written in Modelica, whereas JModelica.org, presented in [4], and OpenModelica, presented in [14], have native support for optimal control. Several other applications of dynamic optimization of Modelica models have been reported, e.g., [5, 40, 65, 91, 94].

gPROMS [37] supports dynamic optimization, both a single shooting and a multiple shooting algorithm, intended for chemical engineering applications. In comparison with Modelica, which partially supports Partial Differential Equations (PDEs) [98, 97], gPROMS supports PDEs more extensively using the method of lines. ACADO [67, 57] is a numerical package based on Ipopt [119, 120], for automatic control and dynamic optimization of direct optimal control including model predictive control, state and parameter estimation, and robust optimization. While ACADO offers state of the art algorithms, formulating the model and optimization descriptions are not supported by a graphical user interface.

During recent decades, different co-simulation technologies and frameworks have emerged. For example, Cosimate [117], Ptolemy-II [35], MILAN [15], and the integrated co-simulation environment for heterogeneous systems prototyping [64]. Most of them are focused on co-simulation of control systems and mechanical components, hardware-software co-simulation, or embedded system simulation.

The HOPSAN [56] software is one of the first general TLM-based co-simulation implementations with its own graphical modeling language. A newer version, HOPSAN-NG [13], has recently been developed. Moreover, a TLM implementation for the Modelica language has recently been developed as part of OpenModelica (See Chapter 7 in [110]).

A meta-modeling approach for TLM-based co-simulation has been also developed in [108] and extended later in [106, 105] to better support composite modeling for mechanical system simulations. More recently, the work presented in [107] supports a fully functional composite model coupled simulation environment that supports integration of many different simulation tool-specific models into a co-simulation (Note: the term ”meta-model” is used in some of these papers with a meaning defined as ”composite model”). The most important consideration in such coupled simulation is numerical stability for solvers using variable time steps, as well as to make co-simulation modeling applicable for a wider range of tools. However, it is limited to in-house usage at SKF and also lacks advanced graphical modeling and validation to assist users in the early phases of co-simulation modeling. Therefore, an additional goal in the development of TLM-based co-simulation tools is to provide an open-source tool for advanced graphical and textual co-modeling simulation, including validation.

A tool for accessing Modelica from Python, OMPython [45], has been developed. It communicates with OpenModelica via CORBA. OMPython is a Python package which makes it possible to pass OpenModelica Shell commands as strings to a Python function, and then receive the results back into Python. However, this requires good knowledge of OpenModelica Shell commands and syntax.

A simulation and analysis tool, PySimulator, has been developed to ease the use of Modelica from Python [1]. Essentially, PySimulator provides a GUI based on Python, which allows running of Modelica models and presentation of the results. It is also possible to analyze the results using various packages in Python, e.g., FFT analysis. However, PySimulator does not currently give the user full freedom to integrate Modelica models with Python and use the full available set of packages in Python, or to freely develop analysis routines in Python.

The free JModelica.org tool includes a Python package for converting Modelica models to FMUs, and then for importing the FMU as a Python object. This way, Modelica models can essentially be simulated from Python — Optimica is also supported. It is possible to do more advanced analysis with JModelica.org via CasADi, see, e.g., [90] and [89]. However, the functionalities in [90] use an old version of JModelica.org. It would be more useful if these functionalities were supported by the tool developer.

3 Integration of Optimization tool-chain into the Model-Based Development Process

This chapter is based on the following paper:

- **Alachew Shitahun**, Vitalij Ruge, Mahder Gebremedhin, Bernhard Bachmann, Lars Eriksson, Joel Andersson, Moritz Diehl, and Peter Fritzson. **Model-Based Optimization with OpenModelica and CasADi.** In Proceedings of IFAC Conference in Tokyo, September 2013.

- **Alachew Shitahun**, Vitalij Ruge, Mahder Gebremedhin, Bernhard Bachmann, Lars Eriksson, Joel Andersson, Moritz Diehl, Peter Fritzson. **Tool Demonstration Abstract: OpenModelica and CasADi for Model-Based Dynamic Optimization.** In Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, Nottingham, UK, April 19, 2013.

3.1 Introduction

During the last decade, nonlinear model predictive control (NMPC) and non-linear optimal control problems (NOCP) based on Differential-Algebraic Equations (DAEs) have had a significant impact in the industrial community, particularly in the control engineering area [20, 115]. State-of-the-art methods use numerical algorithms for dynamic optimization based on direct multiple shooting [22] or collocation algorithms [20].

Use of equation-based, object-oriented modeling languages such as Modelica for industrial applications has increased. These languages enable users to conveniently model large-scale physical systems described by differential, algebraic, and discrete equations, primarily with the goal of performing virtual experiments (simulation) on these systems, but recently also optimization.

Due to the influence of such equation-based, object-oriented modeling languages in the industrial community, there have been several attempts to integrate tools for such languages with numerical algorithms for optimization. For example, Dymola (DassaultSystemes, 2010) supports parameter and design optimization of models written in Modelica, whereas JModelica.org [4] and OpenModelica [14] have native support for optimal control.

This chapter presents results of an effort in which OpenModelica and CasADi [7] have been integrated to perform model-based dynamic optimization. The problem formulation and modeling is done in Modelica (Modelica Association, 2010) including the optimization [3] language extension. The integration is based on standardized XML format presented in [86] for exchange of DAEs models. OpenModelica supports export of models written in Modelica and the optimization language extension using this XML format, while CasADi supports importing of models represented in this format. This allows users to define optimal control problems (OCP) using Modelica and optimization language specification, and solve the underlying model formulation using a range of optimization methods, including direct collocation and direct multiple shooting. The proposed solution has been tested on several industrially relevant optimal control problems, including a diesel-electric power train.

3.2 CasADi

CasADi [7] is an open-source framework for C++ and Python that provides numerical optimization in general and optimal control in particular. The main idea of the tool is to provide users with the ability to easily and efficiently implement optimal control algorithms with a wide range of methods, including multiple shooting and collocation, rather than providing users with a "black-box" OCP solver. The tool supports symbolic import of OCPs via an extended version of the functional mockup interface (FMI) format as explained in [6]. This OCP can then be transcribed into a nonlinear programming problem (NLP) using the approach outlined in Section 2.2, and solved with one of CasADi's interfaced NLP solvers.

3.3 Modelica and the Optimization Language Extension

Modelica is a mature and powerful language with regard to modeling of complex hybrid dynamical systems. However, it lacks important features for describing or modeling optimization problems. This is not a surprise since Modelica was not originally designed to help with dynamic optimization problems.

The optimization language extension [3] complements Modelica by providing features that enable formulation of dynamic optimization problems based on Modelica models. The optimization extension to Modelica consists of the following elements:

- *objective* and *objectiveIntegerand*, which maps the Mayer and the Lagrange term in the objective function respectively.
- *startTime*, which defines the start of the optimization interval.
- *finalTime*, which defines the end of the optimization interval.
- A new section: *constraint*, which defines inequality constraints.

The requirement and motivation for introducing these specific features is covered in more detail in [3].

3.4 Modeling NOCP and XML Export in OpenModelica Compiler

The OpenModelica compiler front-end has been extended to support the optimization language extension described in Section 3.3. This enables users to use the OpenModelica graphical editor (OMEdit) (see Figure 3.1)to formulate and use model-based NOCP that can be solved by CasADi.

In addition, the OpenModelica compiler has recently been extended with XML export of models [102] based on the XML format defined in [86]. This schema is an extended version of the XML schema defined by the Functional Mock-up Interface (FMI) [78], and is the most recent in a series of Modelica-related XML schemas starting with ModelicaXML [92].The XML export also includes the optimization language extension, and OpenModelica is integrated with CasADi for the type of model-based dynamic optimization reported in this thesis.

3.5 Complete Model-Based Dynamic Optimization tool-chain

Before exporting a Modelica and optimization language model to XML, the model should be symbolically instantiated by the compiler in order to get a single flat system of equations. The model variables should also be scalarized. The compiler front end performs this, including syntax checking, semantics and type checking, simplification and constant evaluation, etc. Then the complete flattened model is exported to XML code. The exported XML document

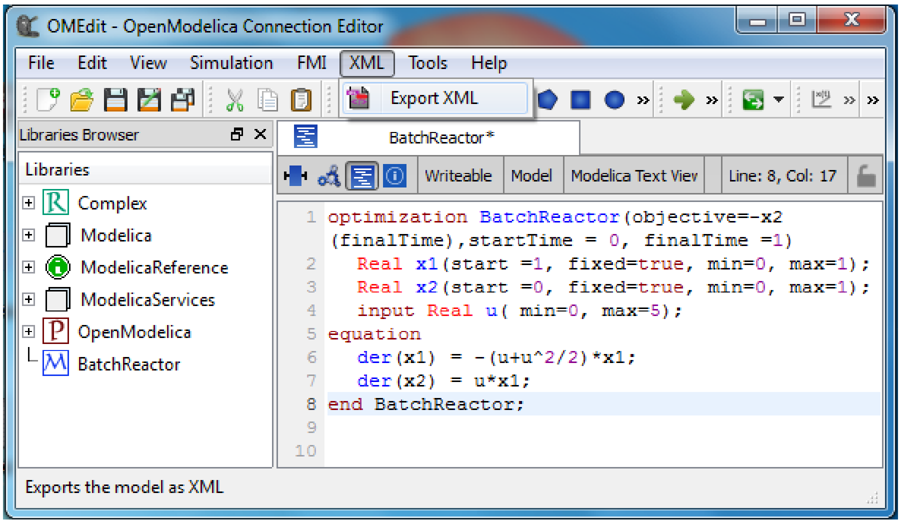

Figure 3.1: Modeling NOCP using the OpenModelica Graphical and Textual Editor.

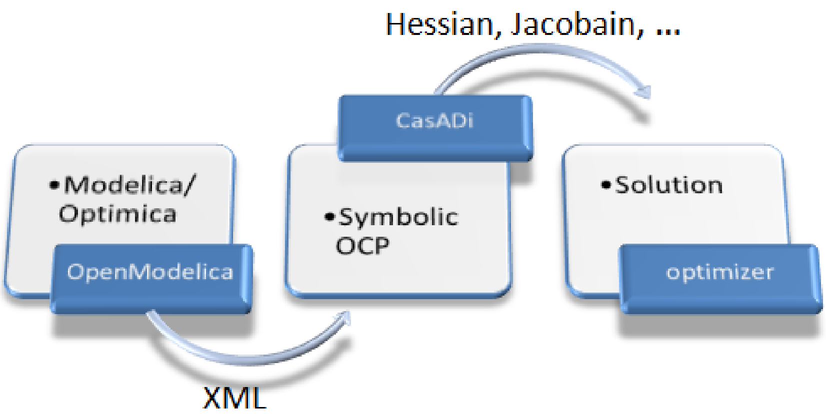

Figure 3.2: Model-Based Dynamic Optimization tool-chain for OpenModelica and CasADi.

can then be imported to CasADi for model-based dynamic optimization. The complete tool chain is visualized in Figure 3.2.

The XML will be imported and symbolically pre-processed in CasADi. In particular, the fully-implicit DAE from Modelica is reformulated in a semi-explicit form. With the NOCP now available in CasADi's native data struc-

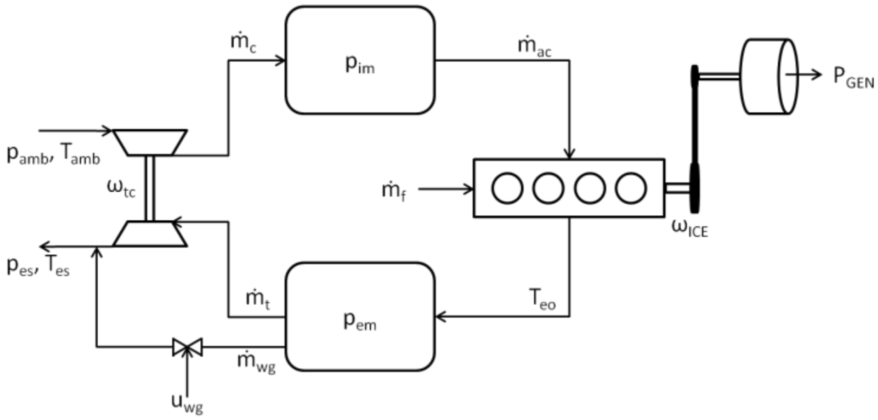

Figure 3.3: Diagram of the diesel-electric powertrain model.

tures, the NOCP can be reformulated to a NLP as outlined in [103] (See Section 2.2).

At the time of writing, the efficient symbolic pre-processing model evaluation from OpenModelica is not yet completely implemented to import into CasADi. So the symbolic preprocessing in CasADi can be used. The symbolic work in CasADi makes it easy to create the goal function and constraints.

The NLP is solved by one of the NLP solvers interfaced to CasADi, e.g., IPOPT [119]. First and second order derivative information will be generated by CasADi using automatic differentiation and passed to the solver.

3.6 Testing the Tool-chain Implementation

In this section, we describe the solution of an industrially-relevant optimal control problem for a diesel-electric powertrain. The formulation of the underlying optimization problem and the corresponding optimization results are presented in the following subsections.

3.6.1 Fuel optimal control of a diesel electric powertrain

The diesel-electric powertrain model presented in [109, 14] is a nonlinear mean value engine model (MVEM) containing four states and three control inputs, while the generator model is simplified by considering constant efficiency and maximum power over the entire speed range, see Figure 3.3 for the schematic diagram of the model.

In a diesel-electric powertrain the operating point of the diesel engine can be freely chosen, which would potentially decrease fuel consumption. Moreover, the electric machine has better torque characteristics. These are the main reasons that the diesel-electric power-train concept is interesting for further studies.

To investigate the fuel-optimal transients of the powertrain from idling condition to a certain power level while the accelerator pedal position is interpreted as a power level request, the following optimal control problem is solved:

$$\text{states} \quad x = 0, \begin{pmatrix} w_{ice} \\ p_{im} \\ p_{em} \\ w_{tc} \end{pmatrix} = \text{controls}, \quad u = \begin{pmatrix} u_f \\ u_{wg} \\ p_{gen} \\ w_{tc} \end{pmatrix}$$

$$\min \int_0^T \dot{m}_f d_t$$

subject to

$$\begin{aligned}
\dot{x}_1 &= f_2(x_2, x_3, u_1, u_3) \\
\dot{x}_2 &= f_3(x_1, x_2, x_4) \\
\dot{x}_3 &= f_4(x_1, x_2, x_3, u_1, u_2) \\
\dot{x}_4 &= f_5(x_2, x_3, x_4, u_2) \\
0 &= f_6(x_2, x_4) - f_7(x_1, x_2) \\
0 &= f_7(x_1, x_2) + f_8(x_1, u_1) - f_9(x_3) - f_{10}(x_3, u_3) \\
0 &= \frac{f_{11}(x_3) - f_{12}(x_1)}{f_{13}(x_4)} - f_{14}(x_4)
\end{aligned}$$

$$\begin{aligned}
54rps \;\leq\; & x_1 \;\leq\; 220rps \\
0.8p_{amp} \;\leq\; & x_2 \;\leq\; 2P_{amb} \\
P_{amb} \;\leq\; & x_3 \;\leq\; 3P_{amb} \\
300rps \;\leq\; & x_4 \;\leq\; 10000rps \\
0 \;\leq\; & u_1, u_2 \;\leq\; 1
\end{aligned}$$

and boundary conditions are:

$$\text{at} \quad t = 0, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \text{idle operating values,}$$

$$\text{at} \quad t = T, \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = 0, \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \end{pmatrix} = \text{desired values}$$

$$\text{and} \quad v_3 = P_{required}.$$

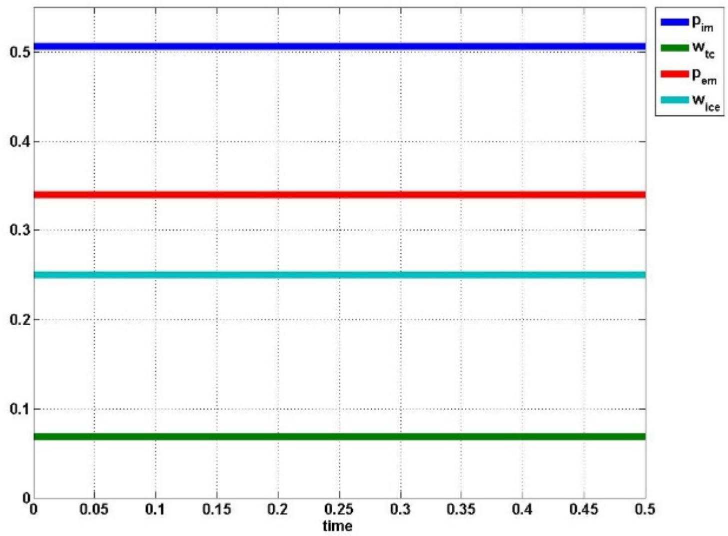

Figure 3.4: Initial guess for diesel model - state variables.

The constraints are originated from components' limitations and the functions f_i are described in [109].

3.6.2 Model import into CasADi and NLP transcription

We used OpenModelica to translate Modelica/ Optimization language extension code into an OCP in DAE and Lagrange cost function. This OCP is then exported into an XML-based symbolic expression format and imported into CasADi via OpenModelica. The OCP can then be transcribed into a nonlinear programming problem (NLP) using the approach outlined in [14] of Section 5, and solved with one of CasADi's interfaced NLP solvers.

3.6.3 Solution of the NLP

The NLP was solved using IPOPT [119] running by default with the MUMPS linear solver. The right scaling is important for a solution without oscillations. On the other hand, if the scaling does not work in all steps then changing of the solver tolerance is helpful. By itself, the diesel model scales well in the time interval[0.32,0.5], which is the critical interval here.

In order to cover the optimal solution of the diesel-electric powertrain model, 140 NLP iterations for 128 sub-intervals with the total collocation (Lobatto6 and Radau5) are required by IPOPT.

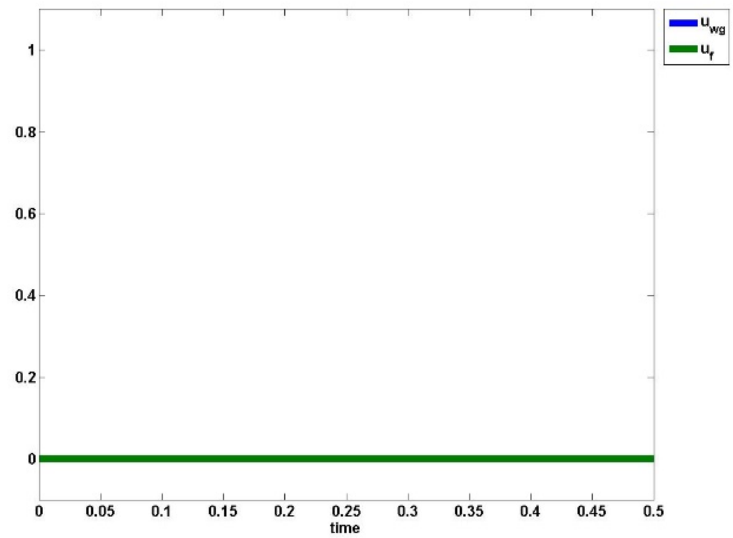

Figure 3.5: Initial guess for diesel model - control variables.

Table 3.1: Execution times for the diesel-electric powertrain model.

Step	Time
IPOPT (without function evaluation)	2.140s
NLP function evaluations	1.158s

A better initial guess will change the NLP iterations. Table 3.1 shows the total CPU time for the optimization. The calculations have been done on a Dell Latitude E6410 laptop with an Intel Core i7 processor of 2.8 GHz, 8 GB of RAM, 4M Cache, running Windows.

The control and state trajectories of the optimal solutions are shown in Figure 3.4 and Figure 3.5, respectively. The problem solved here is a minimum fuel problem for a transient from idle to 170 kW, for an end time of 0.5 s. For simplicity, only diesel operating condition is assumed which means ($u_3 = P_{gen} = 0$). As expected, the fuel optimal results happen when the engine is accelerated only near the end of the time interval ($t \approx 0.32s$), to meet the end constraints while minimizing the fuel consumption.

With amendments to the initial values, the process is robust. For this purpose, the initial values are changed slightly.

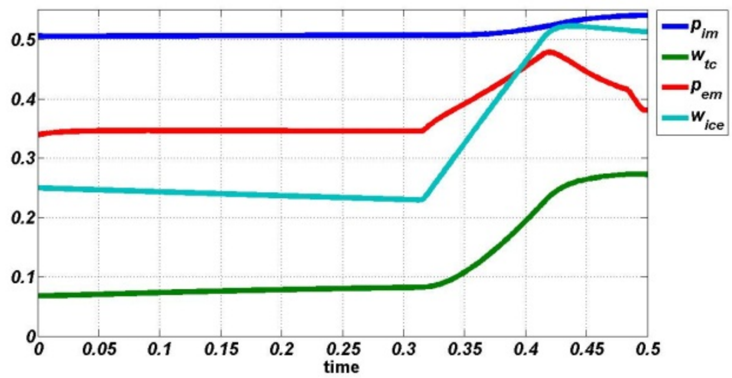

Figure 3.6: Optimization result for diesel model - state variables.

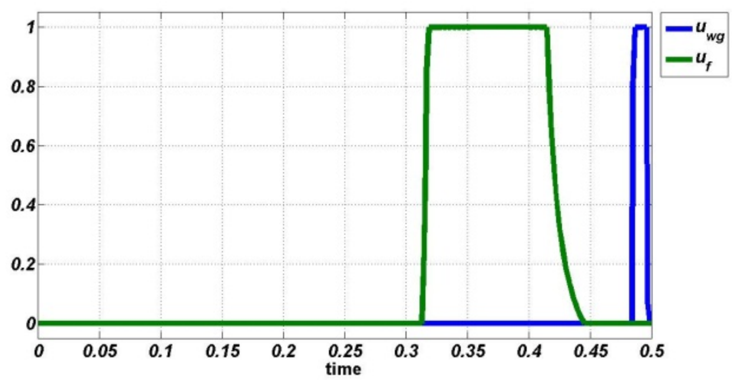

Figure 3.7: Optimization result for diesel model - control variables.

3.7 Summary

This chapter demonstrates simulation-based optimization through the coupling of two open-source tools: OpenModelica, which is a Modelica-based modeling and simulation platform, and CasADi, a framework for numerical optimization. The coupling uses a standardized XML format for exchange of differential-algebraic equations (DAE) models. OpenModelica supports export of models written in Modelica and the optimization language extension using this XML format, while CasADi supports import of models represented in this format. This allows users to define optimal control problems (OCP) using Modelica and optimization language specification, and to solve the underlying model formulation using a range of optimization methods, including direct collocation and direct multiple shooting. The proposed solution has

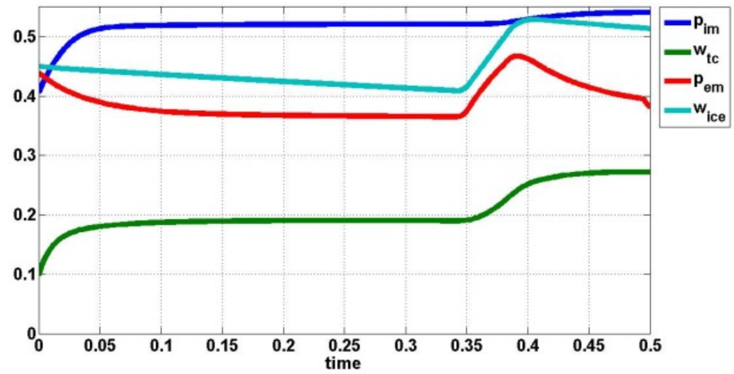

Figure 3.8: Optimization result for diesel model with changed initial values - state variables.

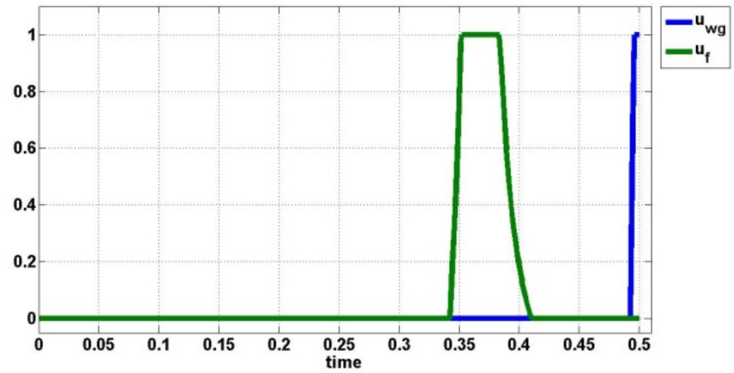

Figure 3.9: Optimization result for diesel model with changed initial values - control variables.

been tested on several industrially-relevant optimal control problems, including a diesel-electric power train.

Acknowledgements

This work has been partially supported by Serc, by SSF in the EDOp project and by Vinnova as well as the German Ministry BMBF (BMBF Förderkennzeichen: 01IS09029C) in the ITEA2 OPENPROD project and in the ITEA2 MODRIO project. The open-source Modelica Consortium supports the OpenModelica work. JA and MD acknowledge support by PFV/10/002 OPTEC, GOA/10/09 and GOA/10/11, FWO G.0320.08, G.0377.09, SBO LeCoPro; Belspo IUAP P7 DYSCO, FP7-EMBOCON (ICT-248940), SADCO (MC

ITN-264735), ERC ST HIGHWIND (259 166), Eurostars SMART, vicerp, ACCM.

4 TLM-Based Co-Modeling Editor and Co-Simulation Framework

This chapter is based on the following paper:

- **Alachew Mengist**, Adeel Asghar, Adrian Pop, Peter Fritzson, Willi Braun, Alexander Siemers, and Dag Fritzson. **An Open-Source Graphical Composite Modeling Editor and Simulation Tool Based on FMI and TLM Co-Simulation.** In Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015.

4.1 Introduction

Industrial products often consist of many components that have been developed by different suppliers using different modeling and simulation tools. Integrated modeling and simulation support is needed in order to integrate all the parts of a complex product model. TLM-based modeling and co-simulation is an important technique for modeling, connecting, and simulation of mechanical systems. It is simple, numerically stable, and efficient. A number of tool-specific simulation models, such as Modelica models, SimuLink models, Adams models, BEAST models, etc., have been successfully connected and simulated using TLM-based co-simulation.

This has been successfully demonstrated by integrating and connecting several different simulation models, especially for mechanical applications. Such an integrated model, consisting of several model parts, is referred to here as a composite model, since it is composed of several sub-models. Another name that is used for such a model is meta-model, since it is a model of models. In earlier work [108, 106], Modelica, with its object-oriented modeling capabilities and its standardized graphical notations, has been used to demonstrate the potential benefits of meta-modeling/composite modeling of mechanical systems using TLM.

The availability of a general XML-based composite modeling language [108] is an important element of our TLM-based modeling and co-simulation framework. However, modelers developing composite models are likely to take advantage of the additional availability of tools that assist them with respect to the composite modeling process (i.e., the process of creating and/or editing a composite model, here represented and stored as XML).

We introduce a graphical composite model editor that is an extension and specialization of the OpenModelica connection editor, OMEdit. In the context of this work, a composite model is composed of several sub-models, including the interconnections between these sub-models. The editor supports creating, viewing, and editing a composite model both in textual and graphical representation. The system supports simulation of composite models consisting of sub-models created using different tools. It is also integrated with the SKF TLM-based co-simulation framework.

4.2 TLM-Based Co-simulation Framework

As mentioned, a general framework for composite model-based co-simulation has previously been designed and implemented in [108]. The design goals for the simulation part of that framework were portability, computational efficiency, and simplicity of incorporation for additional simulation tools. It is also the framework that is used for the TLM-based composite model co-simulation described in this chapter. TLM composite model co-simulation is primarily handled by the central simulation engine of the framework called the TLM simulation manager. It is a stand-alone program that reads an XML definition of the coupled simulation as defined in [108]. It then starts external model simulations and provides a communication bridge between the running simulations using the TLM [80] method. The external models only communicate with the TLM simulation manager, which acts as a broker and performs communication and marshalling of information between the external models. The simulation manager sees every external model as a black box with one or more external interfaces. The information is then communicated between the external interfaces belonging to the different external models. Additionally, the simulation manager opens a network port for monitoring all communicated data. TLM simulation monitor is another stand-alone program that connects to the TLM simulation manager via the network port. The TLM simulation manager sends the co-simulation status and progress to the TLM

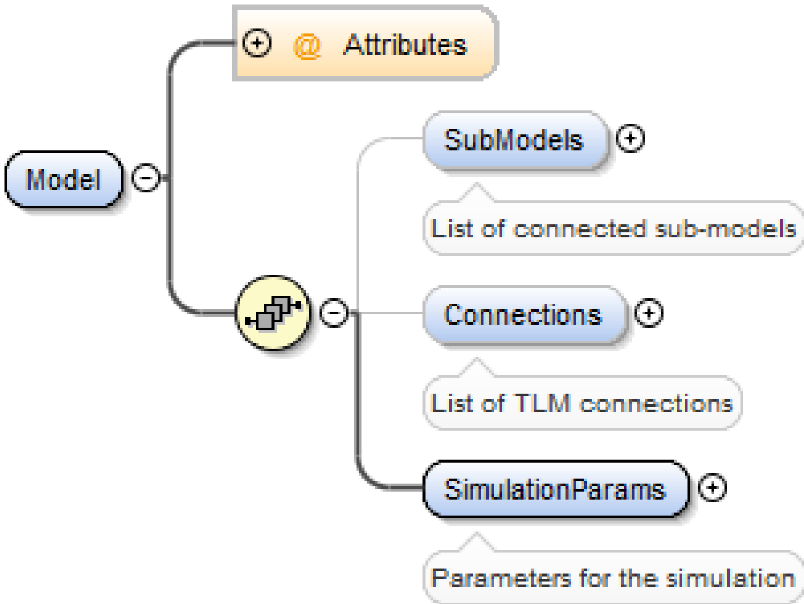

Figure 4.1: The Model (root) element of the Composite Model Schema.

simulation monitor via TCP/IP. The simulation monitor receives the data and writes it to an XML file.

4.3 Composite Model XML Schema

The composite model XML-Schema for validating the co-simulation composite model is designed according to its specification described in [108]. The following is a sample composite model XML representation:

In order to use graphical notations in the composite model editor, the composite model XML file needs to describe annotations for each sub-model and connections between them. We propose to extend the composite model specification by including the *Annotation* element in the *SubModel* and *Connection* elements.

<Annotation Origin="-50,54" Extent="-10,-10,10,10" Rotation="0" Visible="true"/>

The contents of our composite model XML root element, namely *Model* is depicted in Figure 4.1. The root element can contain a list of connected *SubModels* and TLM *Connections*. *SimulationParams* element is also inside the root element. It has an attribute *Name* representing the name of the composite model.

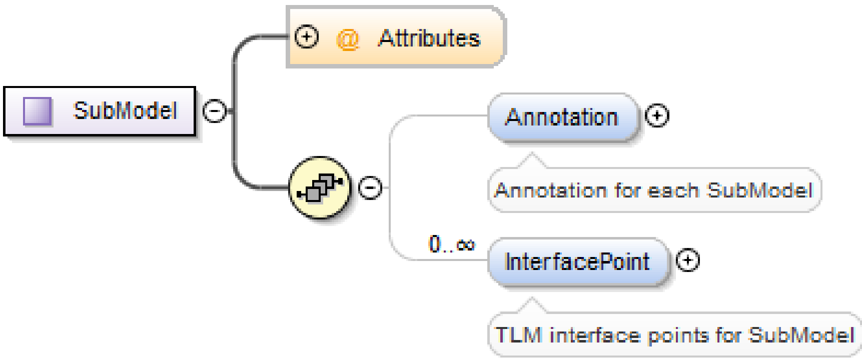

Figure 4.2: The SubModel element from the Composite Model Schema.

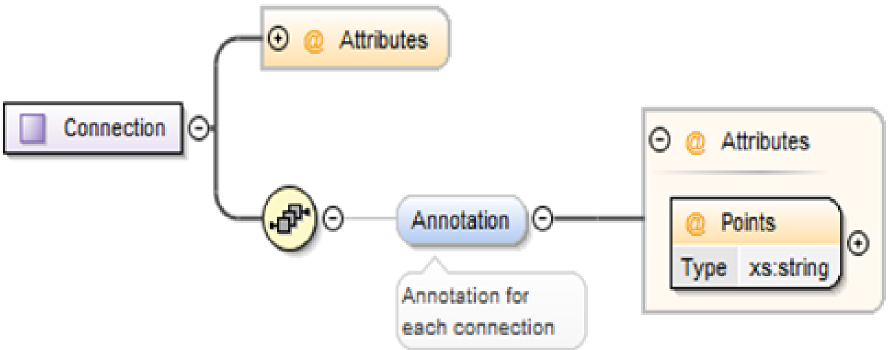

Figure 4.3: The Connection element from the Composite Model Schema.

The *SimulationParams* element specifies the start time and end time for the co-simulation.

The *SubModel* element, presented in Figure 4.2, represents the simulation model component that participates in the co-simulation. The required attributes for a *SubModel* are *Name* of the sub-model, *ModelFile* (file name of the submodel), and *StartCommand* (the start method command to participate in the co-simulation). Each *SubModel* also contains a list of interface points. *InterfacePoint* elements are used to specify the TLM interfaces of each simulation component (sub-model).

The *Connection* element of the composite model XML schema is shown in Figure 4.3.

The *Connection* element defines connections between two connected interface points, that is, a connection between two TLM interfaces. Its attributes

From and *To* define which interfaces of which submodels are connected. Other attributes of the *Connection* element specify the delay and maximum step size.

4.4 Composite Model Graphical Editor

One of the primary contributions of this effort is our focus on interoperability in modeling and simulation. Our effort leverages OpenModelica for graphical composite model editing as well as SKF's co-simulation framework for TLM-based co-simulation. The implementation of this graphical composite model editor is an extension of OMEdit, which is implemented in C++ using the Qt graphical user interface library.

The full graphical functionality of the composite modeling process can be expressed in the following steps:

1. Import and add the external models to the composite model editor,
2. Specify startup methods and interfaces of the external model,
3. Build the composite models by connecting the external models,
4. Set the co-simulation and TLM parameters in the composite model.

An overview of the different components that the graphical composite model editor relies on is shown in Figure 4.4.

The graphical composite model editor communicates with the OpenModelica compiler to retrieve the interface points for the external model and SKF's co-simulation framework to run the TLM simulation manager and simulation monitor. Each tool component is described in the following subsections.

In the graphic composite model editor, the modeling page area is used for visual composite modeling or text composite modeling. This allows users to create, modify, and delete sub-models in a user-friendly manner.

4.4.1 Visual Modeling

Each composite model has two views: a Text view and a Diagram view. In the diagram view, each simulation model component (sub-model) of the TLM co-simulation can be dragged and dropped from the library browser to this view, causing the sub-model to be automatically translated into a textual form by fetching the interface name for the TLM based co-simulation. The user can complete the composite model (see Figure 4.5) by graphically connecting components (sub-models).

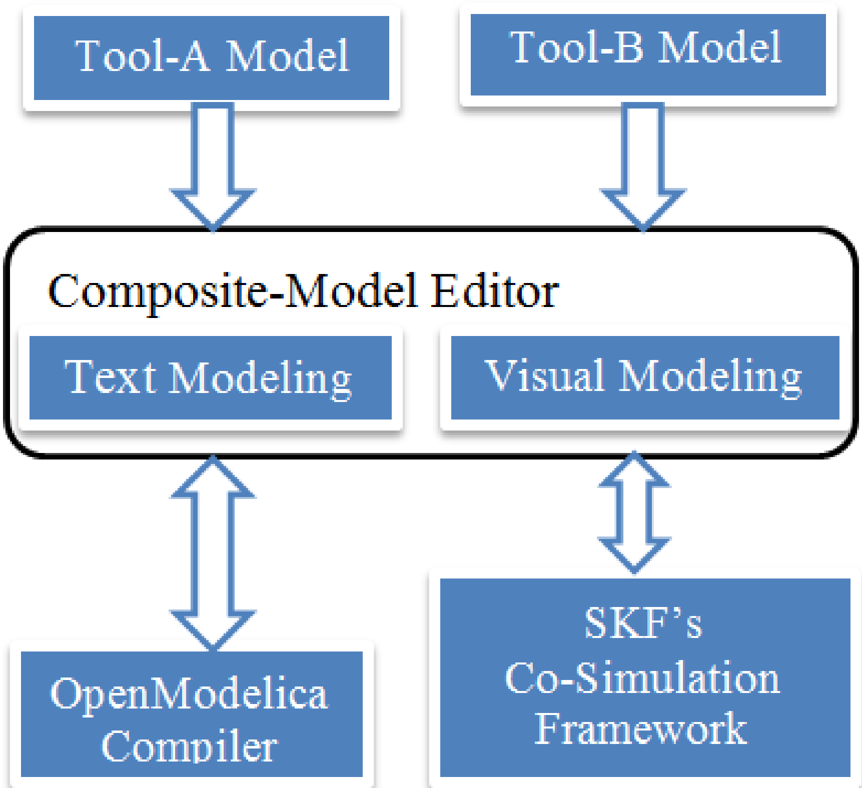

Figure 4.4: An overview of the interaction between the composite model (meta-model) graphic editor and the other components.

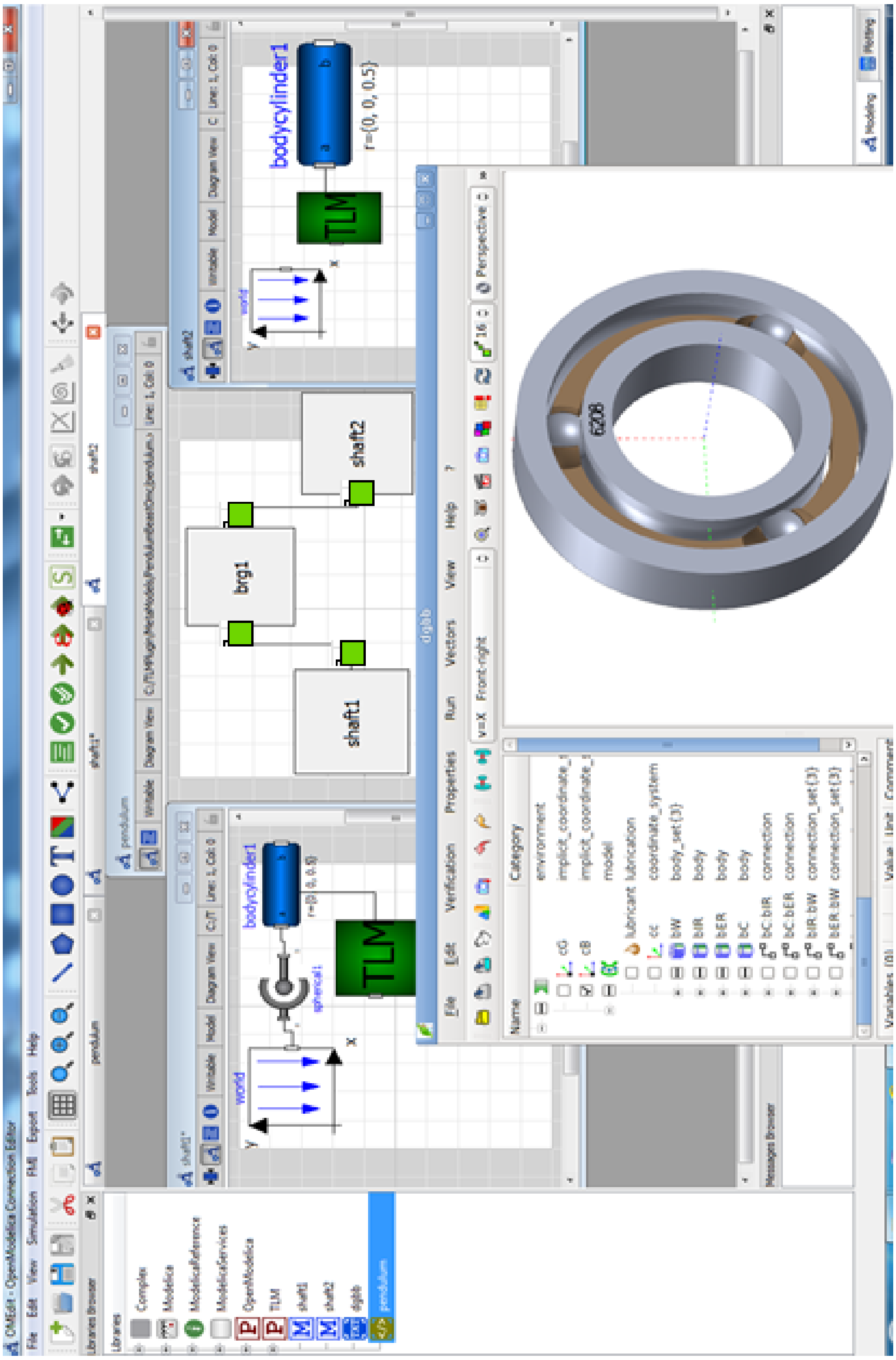

Figure 4.5: A screenshot of visual composite modeling of a double pendulum.

The test model (see Figure 4.5) is a multibody system that consists of three sub-models: Two OpenModelica *Shaft* sub-models (*Shaft1* and *Shaft2*) and one SKF/BEAST bearing sub-model that are assembled to build a *double pendulum*. The SKF/BEAST bearing submodel is a simplified model with only three balls to speed up the simulation.

Shaft1 is connected with a spherical joint to the world coordinate system. The end of *Shaft1* is connected via a TLM interface to the outer ring of the BEAST bearing model. The inner ring of the bearing model is connected via another TLM interface to *Shaft2*. Together they build the double pendulum with two shafts, one spherical OpenModelica joint, and one BEAST bearing.

4.4.2 Textual Modeling and Viewing

The text view (see Figure 8 4.6) allows users to view the contents (sub-models, connections, and simulation parameters) of any loaded composite model. It also enables users to edit a composite model textually as part of the composite modeling construction process. To facilitate the process of textual composite modeling and to provide users with a starting point, the text view (see Figure 4.6) includes the composite model XML schema elements and the default simulation parameters.

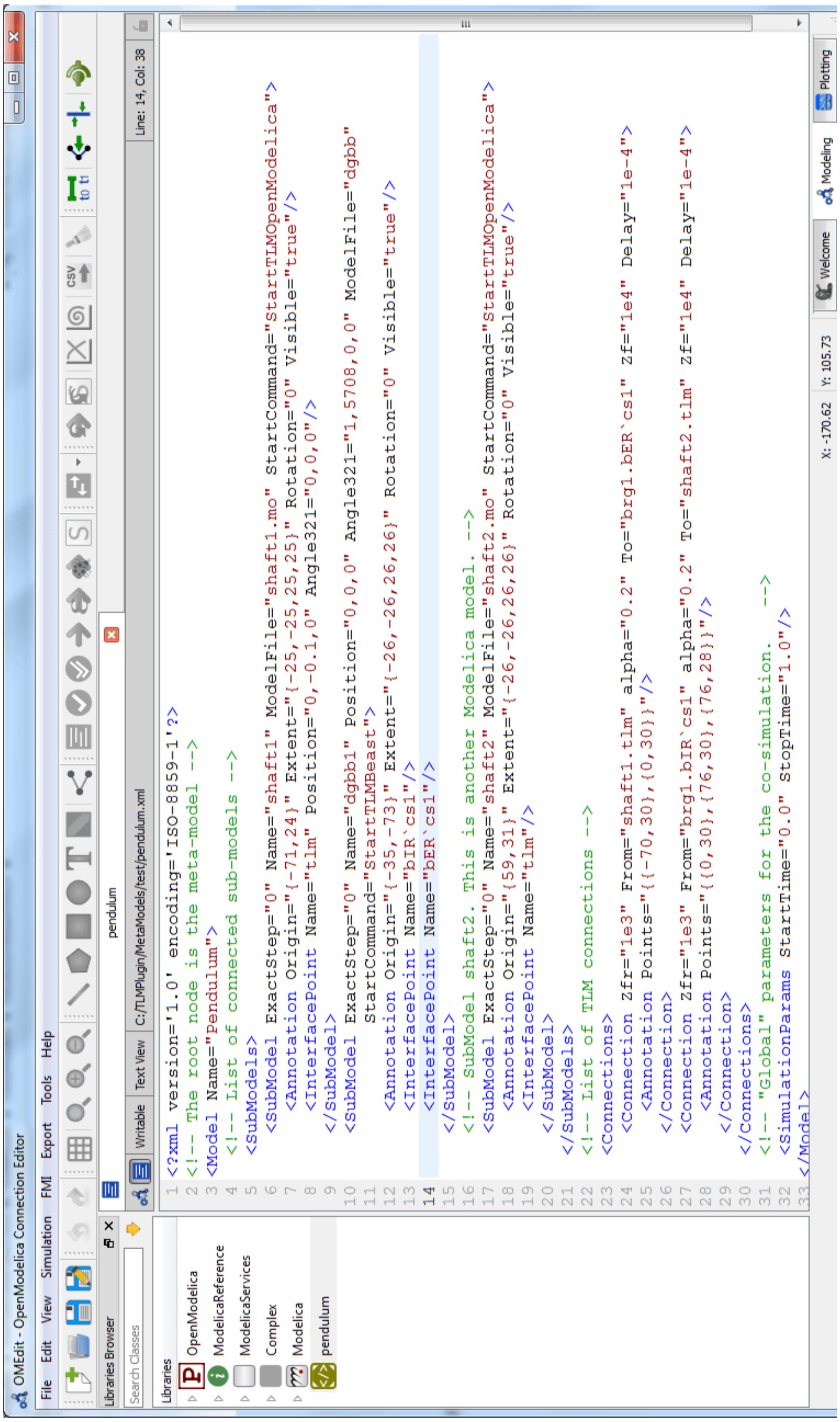

Figure 4.6: A screenshot of textual composite modeling.

4.4.3 Composite Model Validation

Since model validation is part of the composite modeling process, the composite model editor supports users by validating the composite model to ensure that it follows the structure and content rules specified in the composite model schema described in Section 4. In general, the composite model editor validation mechanism permits users to verify that:

- The basic structure of the elements and attributes in the composite model matches the composite model schema.
- All information required by the composite model schema is present in the composite model.
- The data conforms to the rules of the composite model schema.

4.4.4 OpenModelica Runtime Enhancement

To support TLM-based co-simulation, the OpenModelica runtime has been enhanced. The added functionality supports single solver step simulation so that the executed simulation model can work together with the TLM manager. New flags to enable this functionality in the simulation executable are now available:

- -noEquidistantOutputFrequency
- -noEquidistantOutputTime

The new flags control the output, e.g., the frequency of steps and the time increment.

4.4.5 Communication with the SKF TLM-Based Co-Simulation Framework

The graphic composite model editor in OpenModelica provides a graphical user interface for co-simulation of composite models. It can be launched by clicking the TLM co-simulation icon from the toolbar, see Figure 4.7.

The editor runs the TLM simulation manager and simulation monitor. The simulation manager reads the composite model from the editor, starts the co-simulation, and provides the communication bridge between the running simulations. Figure 4.8 shows the running status of the TLM co-simulation.

The simulation monitor communicates with the simulation manager and writes the status and progress of the co-simulation to a file. This file is read by the editor, which uses the data to show the co-simulation progress bar to the user. The editor also provides the means of reading the log files generated by the simulation manager and monitor.

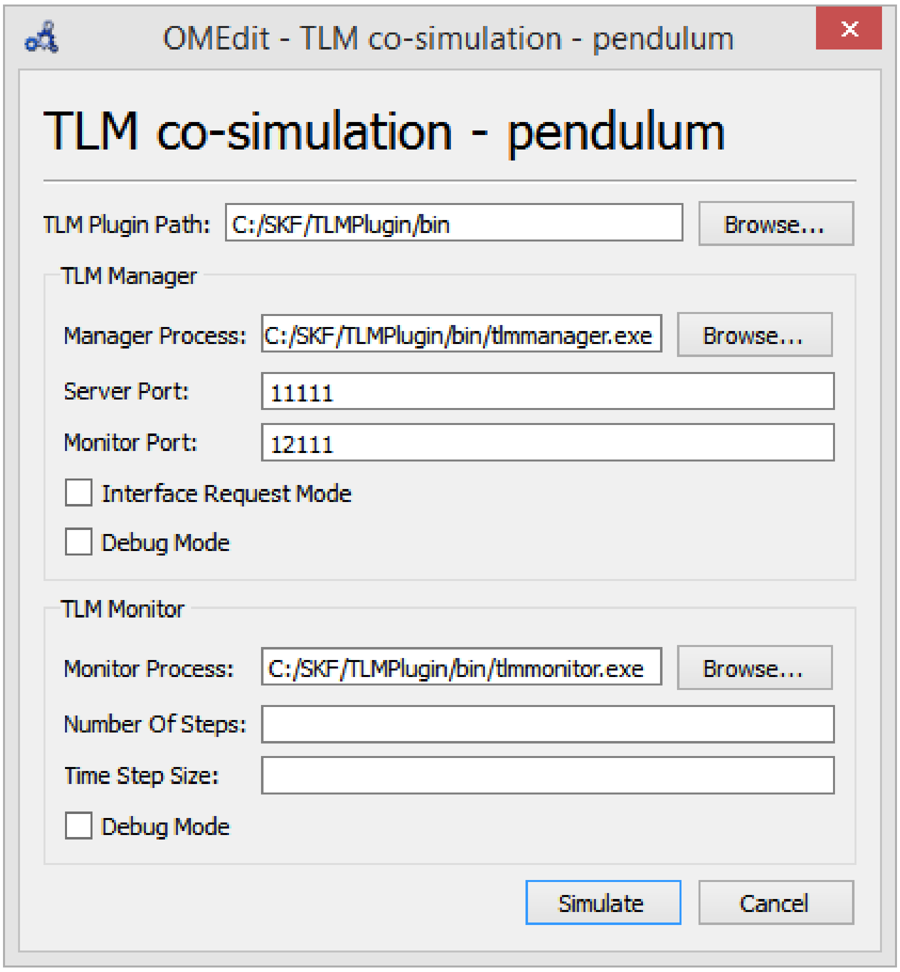

Figure 4.7: TLM co-simulation setup.

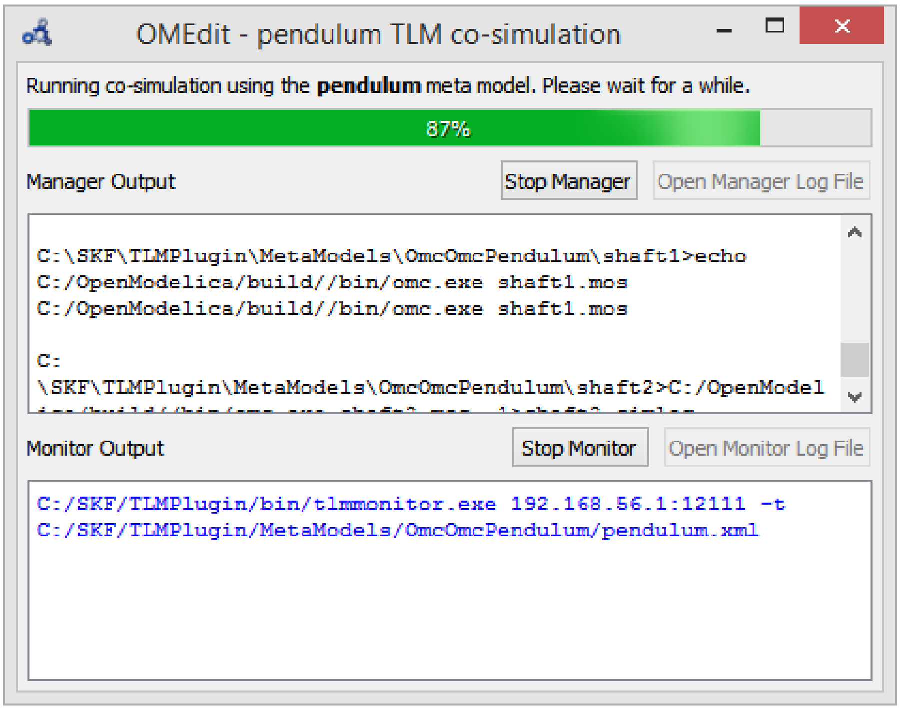

Figure 4.8: TLM co-simulation.

During the post-processing stage, simulation results are collected and visualized in the OMEdit plotting perspective, as shown in Figure 4.9.

4.4.6 Industrial Application of Composite Modeling with TLM Co-Simulation

SKF has successfully used the TLM co-simulation framework to simulate composite models. For example, Figure 4.10 shows one such application with an MSC.ADAMS [112] car model containing an integrated SKF BEAST[113] hub-unit sub-model connected via TLM-connections.

4.5 Summary

In this chapter, we introduced a general open-source graphical editor and simulation tool for composite modeling and co-simulation as well as its integration with SKF's TLM-based co-simulation framework for TLM based co-simulation. The graphical editor combines a number of features to support end-users with respect to the creation of composite models and co-simulation. These include adding, removing, and connecting components (submodels)

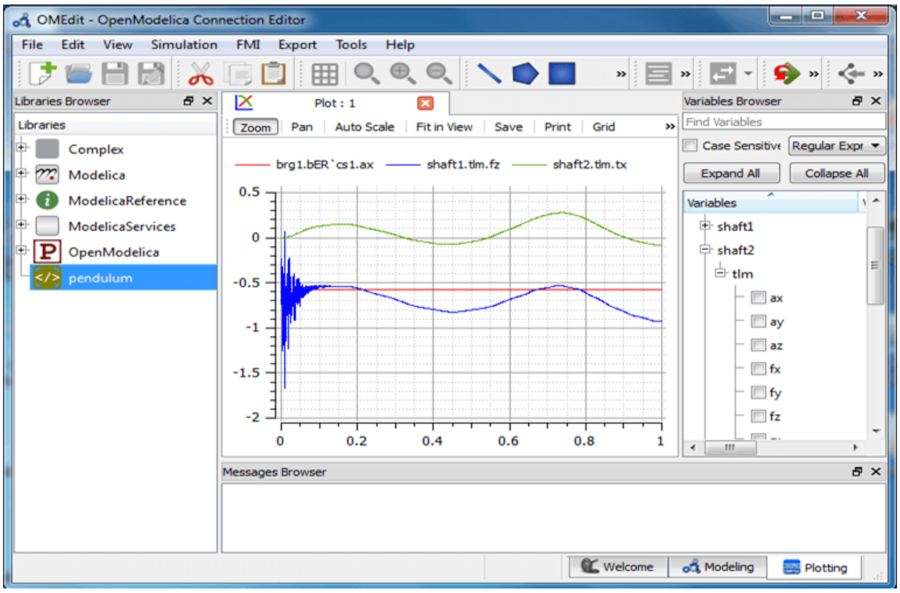

Figure 4.9: Results of TLM co-simulation.

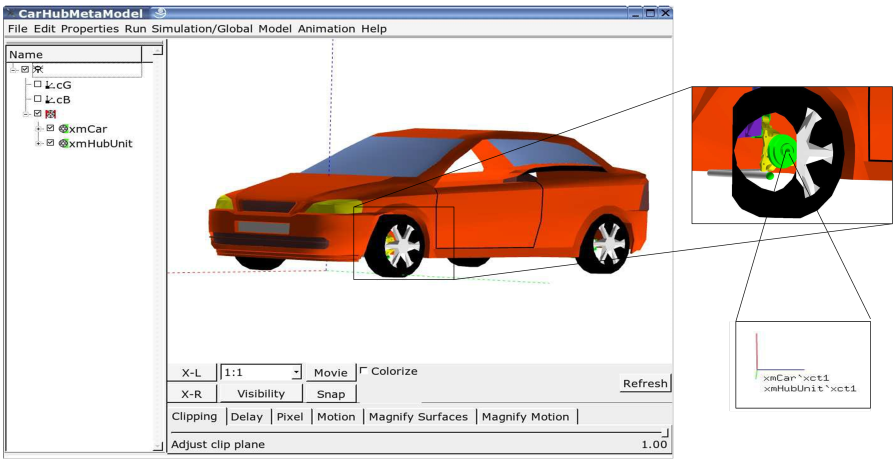

Figure 4.10: A composite model of an MSC.ADAMS car model with an integrated SKF BEAST hub-unit sub-model (green), connected via TLM connections for co-simulation.

both textually and graphically, as well as integrated co-simulation and visualization of simulation results. The composite model editor supports external non-Modelica models represented in XML form (essentially black boxes with interfaces) inside the component tree, which can be used for composite model composition. A number of tool-specific simulation models, such as Modelica models, SimuLink models, Adams models, BEAST models, etc., have been successfully connected and simulated using TLM-based co-simulation. A schema for validation of composite modeling has been developed as a part of this work.

Acknowledgements

The work has been supported by Vinnova in the ITEA2 MODRIO project, by EU in the INTO-CPS project, and by the Swedish Government in the Swedish Government in the ELLIIT project. The open-source Modelica Consortium supports the OpenModelica work. The TLM-based co-simulation framework is provided by SKF.

5 Collaborative Modeling and Traceability in the Model-Based Design of CPSs

This chapter is based on the following paper:

- **Alachew Mengist**, Adrian Pop, Adeel Asghar, Peter Fritzson. **Traceability Support in OpenModelica Using Open Services for Lifecycle Collaboration (OSLC).** In Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017.
- Lena Buffoni, Adrian Pop, **Alachew Mengist**. **Traceability and impact analysis in requirement verification.** In Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, Munich, Germany, December 1, 2017.

5.1 Introduction

The use of modeling and simulation tools for industrial applications has increased. Such tools support different activities in the modeling and simulation lifecycle, like specifying requirements, model creation, model simulation, FMU export, model checking, and code generation. However, the heterogeneity and complexity of modern industrial products often requires special-purpose modeling and simulation tools for different phases of the development life cycle. Seamless exchange of models between different modeling tools is needed in order to integrate all the parts of a complex product model throughout the development life cycle.

During the past decade, the OSLC specifications [83] have emerged for integrating development lifecycle tools using Linked Data [55, 69, 19]. For traceability purposes, the OSLC Change Management specification is particularly relevant. In earlier work [70] OSLC has been successfully demonstrated for integration of modeling tools in general, and traceability in particular.

In [93] the OMC supports traceability in terms of tracing generated C code back to the originating Modelica source code, but not in the OSLC sense, and is mostly used for debugging.

In this chapter we present new traceability support in OpenModelica, where the traceability information is exchanged with other lifecycle tools through a standardized interface and format using OSLC. In particular, it supports automatic recording and tracing of modeling activities such as creation, modification, and destruction of models, import of model description XML, export of FMUs, and creation of simulation results to link models from various tools. OpenModelica supports simple queries (traces to and traces from) to present traceability information to the user.

5.2 Open Services for Lifecycle Collaboration (OSLC)

Open Services for Lifecycle Collaboration (OSLC) [83] is an open-source initiative for creating a set of specifications that enables integration of development life cycle tools (e.g., modeling tools, change management tools, requirements management tools, quality management tools, configuration management tools). The goal of OSLC is to make it easier for tools to work together by specifying a minimal protocol without standardizing the behavior of a specific tool.

The OSLC specifications use the Linked Data model to enable integration at the data level via links between tool artifacts that are defined as Resource Description Framework (RDF) [71] resources (beside other possible representations such as XML, JavaScript Object Notation (JSON) [62], Atom, and Turtle). The resources are identified by HTTP URIs. A common protocol to perform creation (HTTP POST) and retrieval (HTTP GET), update (HTTP PUT) and delete (HTTP DELETE) operations on resources is also specified.

5.3 Traceability Design and Architecture

The traceability design and architecture is mainly being developed in the INTO-CPS project [51, 59], which contains a set of tasks. One of these tasks is the design of traceability and model management with the following goals **??**:

- Checking the realization of requirements in models

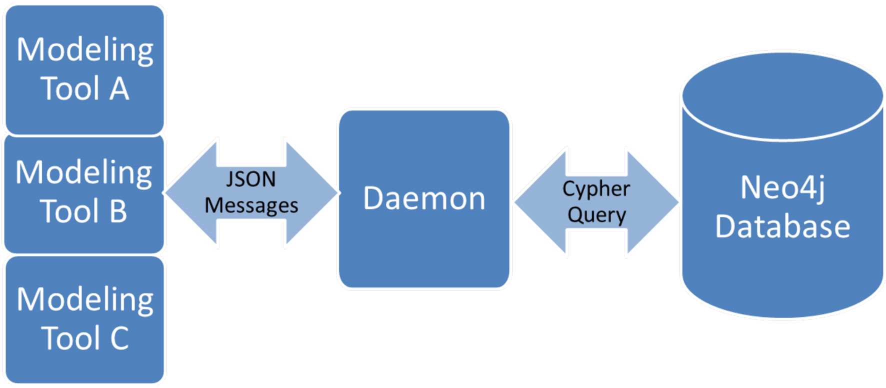

Figure 5.1: Schematic architecture of traceability-related tools.

- Enabling collaborative work by connecting artifacts and knowledge from different users
- Decreasing redundancy by connecting different tools to a single requirements source and allowing a system-wide view that is not solely limited to single tools

The Provenance (PROV) [79] and OSLC standards presented in [39]) are used to support traceability activities. PROV is a set of documents built on the notation and relation of entities, activities, and agents.

The design and architecture of traceability-related tools has recently been developed in [66] and is shown in Figure 5.1. Any modeling tool written in any programming language can use these traceability standards to support the traceability of activities performed within the tool and interact with other tools.

As depicted in Figure 5.1, the architecture is divided into three parts:

Modeling Tools The modeling tools send traceability information from activities that are performed within the tools (e.g., model creation, modification, import model description in XML) to the traceability Daemon.

Traceability Daemon The traceability Daemon provides an OSLC interface compliant with RESTful [96] to store the traceability information into the database and retrieve the traceability data from the database. It is launched and terminated by modeling tools.

Neo4j Graph Database The Neo4j database [81] is a graph database to store the OSLC triples that make up the traceability data.

5.4 An Example of Integrated Tools for Cyber-Physical Model Development

OpenModelica has been successfully integrated with the INTO-CPS tool chain to trace artifacts created during the system development process from high level requirements to simulation results. The tools involved are Overture, 20-sim, Modelio and RTTester. The tool chain as shown in Figure 5.2 is defined by the connections between the system architecture and the simulation via the model description XML file and the FMU.

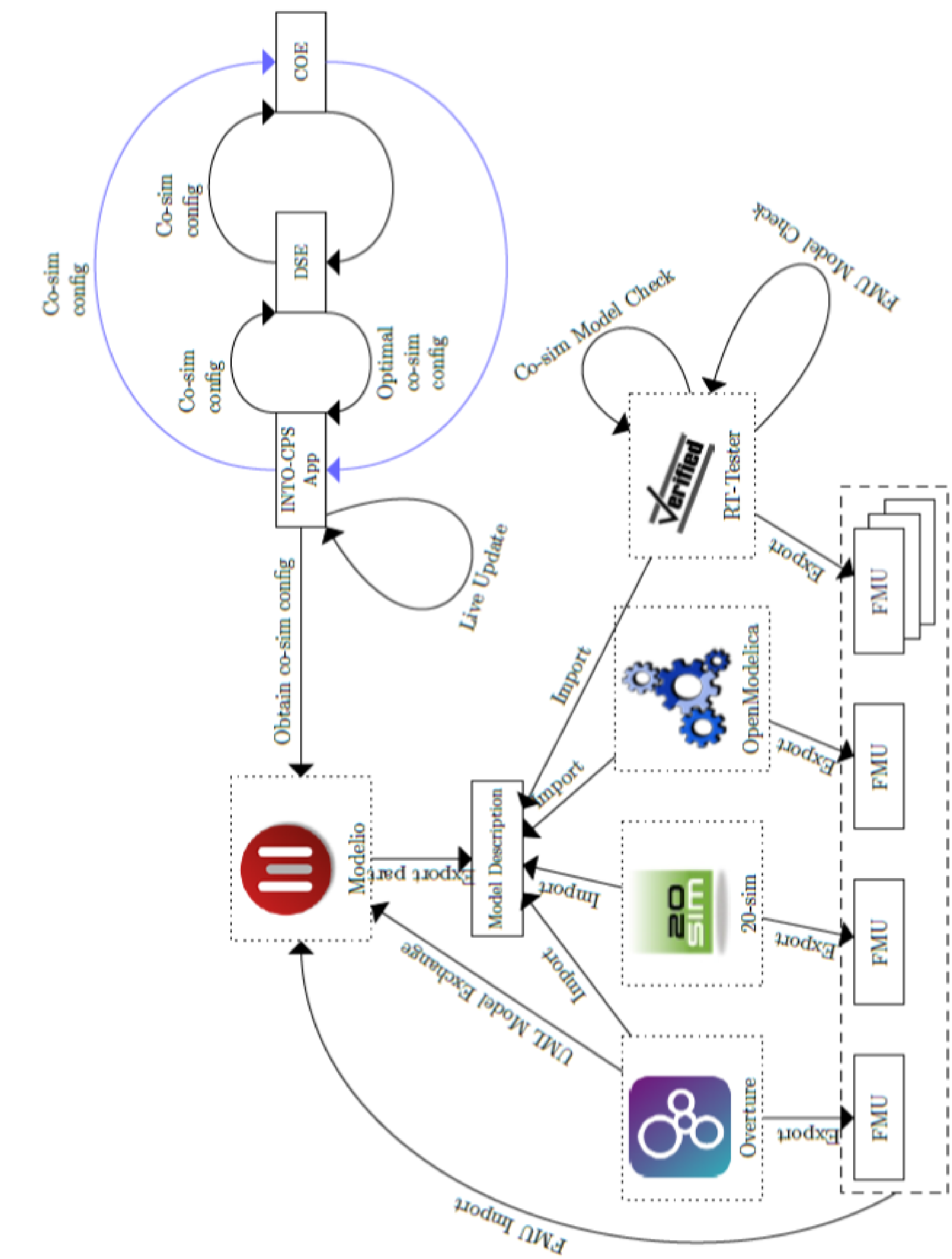

Figure 5.2: An example of integrated tools to trace artifacts created during the system development process (Bandur et al, 2016).

The SysML Connection diagram defines the components of the system and their connections. The internals of these block instances are created in the various modeling tools and are exported as FMUs. The modeling tools support importing the interface definition (ports) of the blocks in the Connection diagram by importing a modelDescription.xml file containing the block name and its interface definition linked with requirements. All tools are storing information in Git and sending information about existing and created artifacts to the global database.

5.5 Traceability and Model Management in OpenModelica

In the new work reported in this chapter, OpenModelica has been extended with support for traceability in the OSLC sense, where traceability information is exchanged with external tools through a standardized interface and format. The implementation is based on an architecture and a common interface, defined in [66], for exchanging traceability information.

The modeling activities that can be recorded automatically and traced within OpenModelica are:

- Model description XML import (linked with requirements)
- Model creation
- Model modification
- Model destruction
- FMU export
- Simulation result creation

The complete workflow for traceability artifacts within OpenModelica and the different components that it relies on are shown in Figure 5.3.

The following summarizes the main workflow that could be used to create and record traceability information in OpenModelica during the cyber-physical model development process.

1. Commit model file entity to the Git repository and record the git commit hash
2. Create URIs of the activity based on the git commit hash
3. OSLC triples describing the activity are generated using the URIs
4. OSLC triples are sent to the traceability Daemon

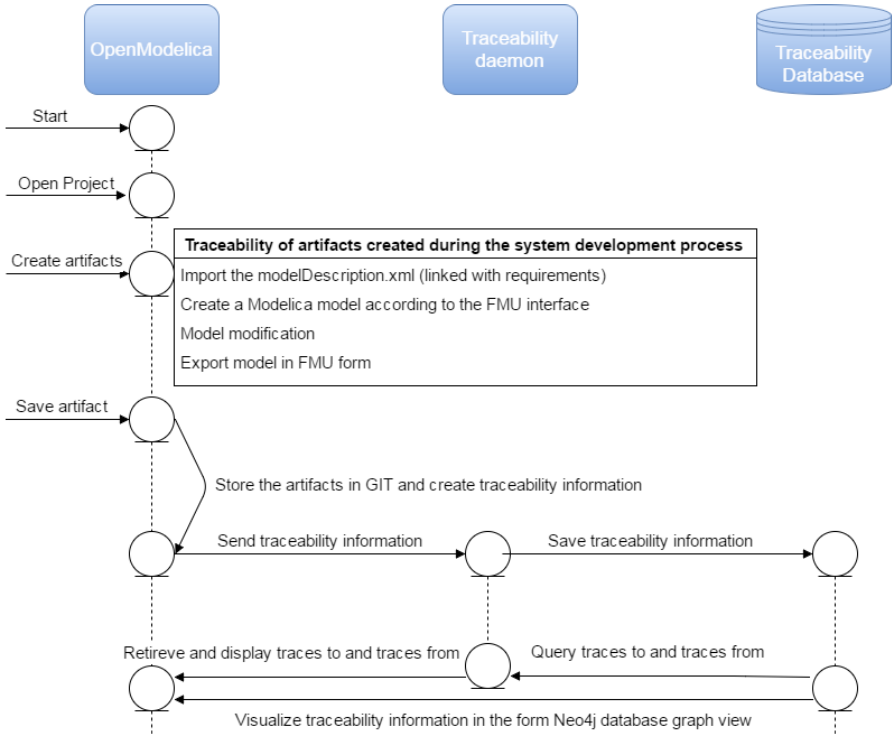

Figure 5.3: Workflow of traceability of artifacts during the system development process in OpenModelica.

5. Retrieve the traceability information (traces to and traces from)

The traceability information is represented in JSON format. The modeling activities described by OSLC triples represented in JSON format are sent from OpenModelica to the traceability Daemon. These traces are then sent through the traceability Daemon to the Neo4j database, where they are stored. In order to view and analyze traceability data, it is retrieved later (traces to and traces from) from OpenModelica, through the appropriate queries from the traceability Daemon to the database.

5.6 Prototype Implementation

We have implemented a prototype to demonstrate the idea of exchanging traceability information for integrating lifecycle modeling tools using OSLC. The prototype is implemented based upon the design and architecture presented in Section 5.1.

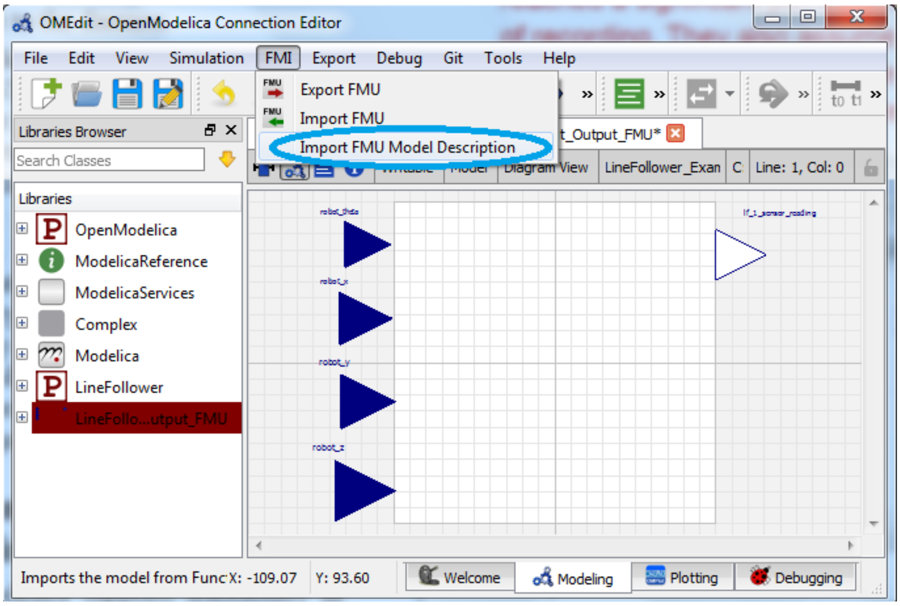

Figure 5.4: A screen shot of the model description XML import operation.

As mentioned, the implementation of this prototype is an extension of OMEdit which is implemented in C++ using the Qt Framework graphical user interface library. For the purpose of presentation, we have grouped the prototype functionality into three categories: importing model description XML, model management with Git integration, and traceability support using OSLC, each of which is described in the following subsections.

5.6.1 Import Model Description in XML

In preparation for the extension to support tracing for importing modelDescription.xml interface files, we extended OpenModelica to support importing modelDescription.xml (See Figure 5.4).

OpenModelica can import model description XML interface files (linked with requirements) created using other system architectural modeling tools, and can then create Modelica models from this information. The result is a generated file with a Modelica model stub containing the inputs and outputs specified in the model Description.xml file. Then the user can create a complete model using the GUI via drag and drop in the editor. Hence, the traceability chain within OpenModelica traces models linked with requirements through model description XML import, model creation, model modification, FMU export and simulation results.

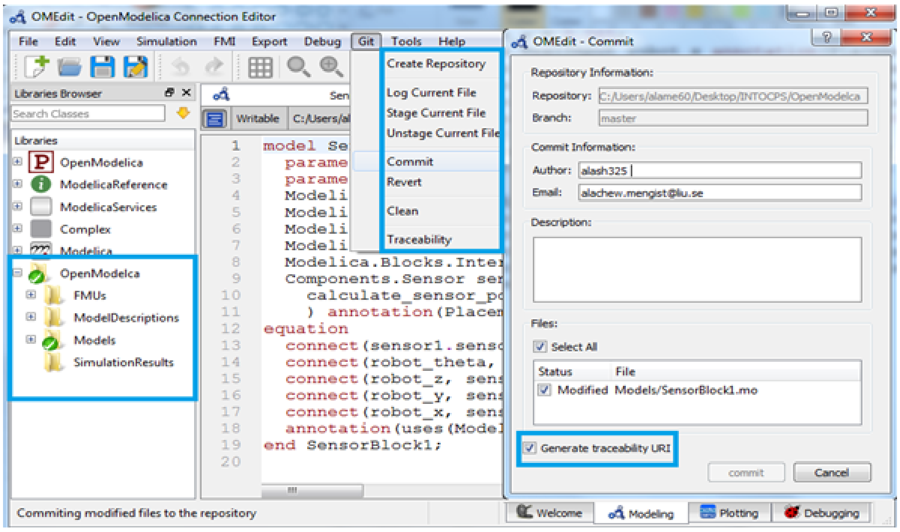

Figure 5.5: GUI of Git Integration in OpenModelica and functions available to create traceability URI.

5.6.2 Model Management with Git Integration

One of the objectives of the traceability tooling is to manage the development process in terms of modeling activities within the modeling tools. In order to achieve this objective, access to the version control system is required in OpenModelica. Therefore, the OpenModelica Connection Editor OMEdit has been enhanced to support Git version control, as shown in Figure 5.5.

The OMEdit Git integration is currently in an early stage of development but already supports some basic functionality (see Figure 5.5) such as staging modified tracing operations on files for commit, committing, and reverting changes. It is useful to provide viewing of status and version history, which can be used to create the resource URIs for the modeling activities on each new commit.

The implemented prototype also allows creation of a local Git repository by selecting Git -> Create New Repository from the menu bar. Since the URI, as presented in [39], is the combination of the git commit hash and the unique path for every file in the project, creating a Git repository for traceability purposes automatically adds a structure to the Git repository for models, simulation results, FMUs, and model description XML files (see the left part of Figure 5.5).

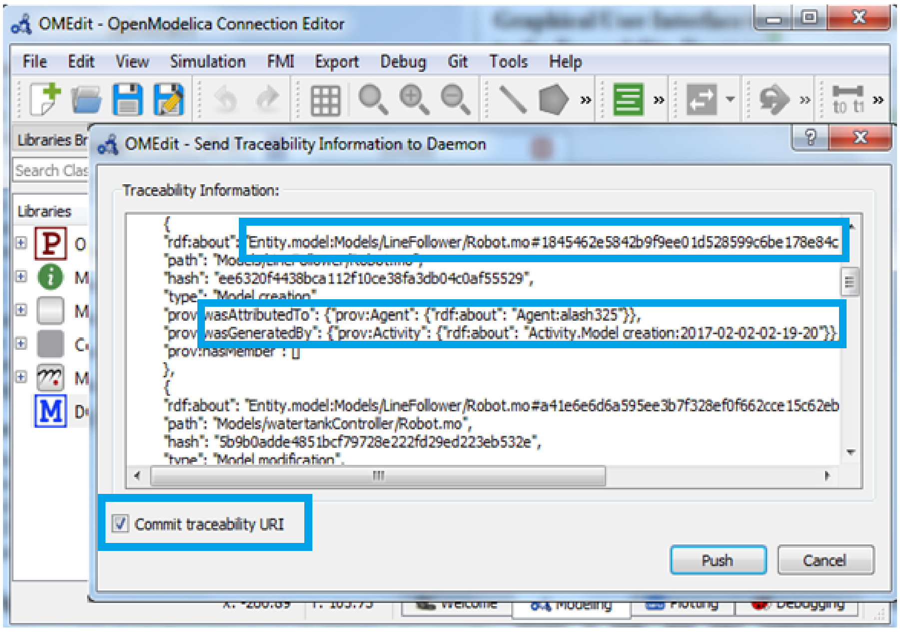

Figure 5.6: GUI to send traceability information to the traceability Daemon.

5.6.3 Traceability Support in OpenModelica

The traceability support in OpenModelica provides a graphical user interface to interact with other lifecycle modeling tools.

As already mentioned in Section 5.5, OpenModelica supports traceability in the OSLC sense, where traceability information is exchanged with external tools through a standardized interface and format. The implementation is based on the architecture and a common interface defined in [66] for exchanging traceability information. OpenModelica imports the modelDescription.xml and creates a Modelica model according to the FMU interface. The generated Modelica model is completed with behavior for the SysML block and the final model is exported in the FMU form. The generated FMU is then used in a whole-system simulation connected according to the SysML connection diagram. The FMU master simulation algorithm component performs the simulation via the INTO-CPS App. This whole chain is traced using OSLC.

We have designed a graphical user interface shown in Figure 5.6 which allows the user to record the traceability information and send it to the traceability Daemon (OSLC triples in JSON format), describing the activity using the URIs generated in the GUI shown in Figure 5.5. The PROV and OSLC relations that are mainly used in this work can be found in [39].

These traces are then sent through the traceability Daemon to the database via HTTP POST http://localhost:8080/ traces/push/json, where they are stored. Figure 5.7 shows an example of traceability information sent from OpenModelica to the traceability Daemon and visualized in the Neo4j database.

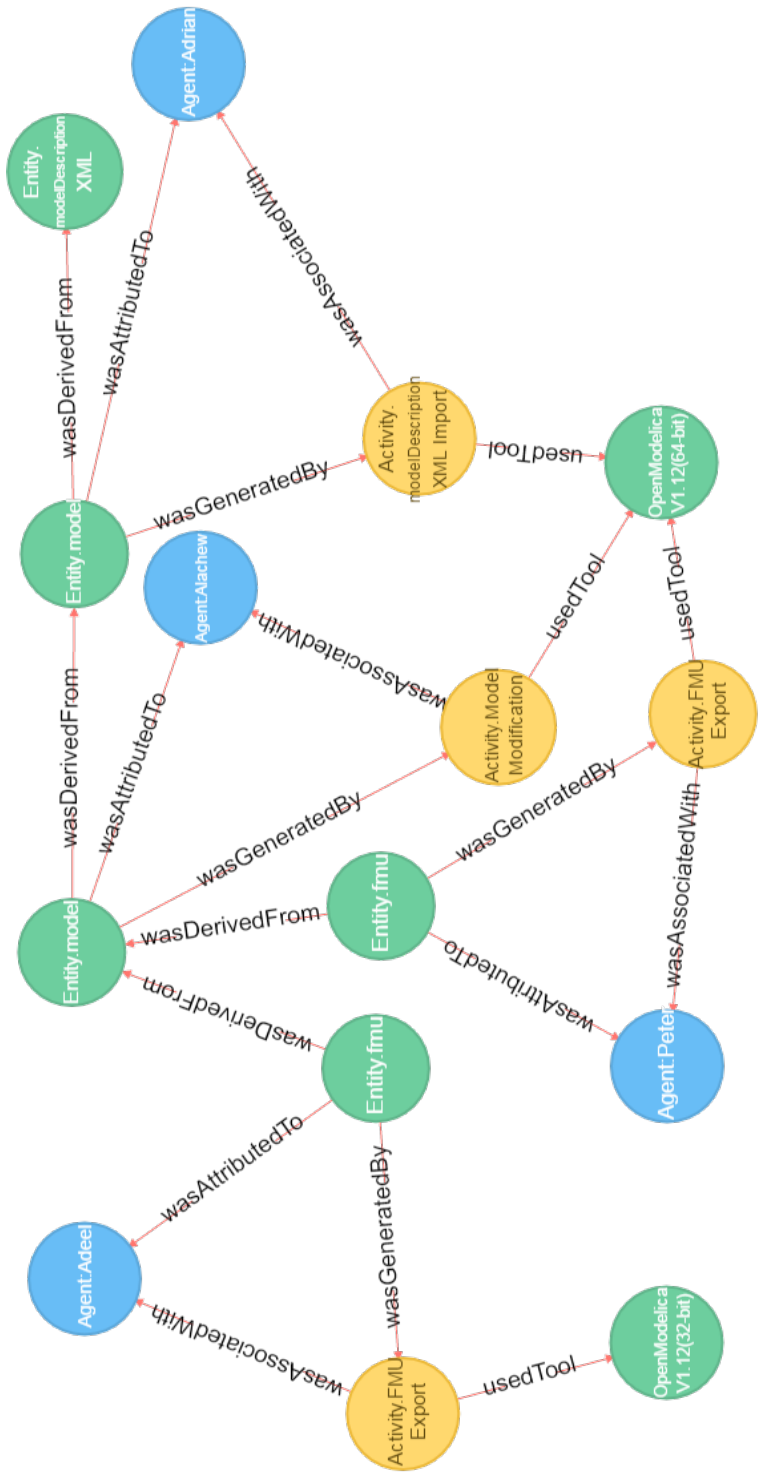

Figure 5.7: An example of traceability information sent from OpenModelica to the traceability Daemon and visualized in the Neo4j database.

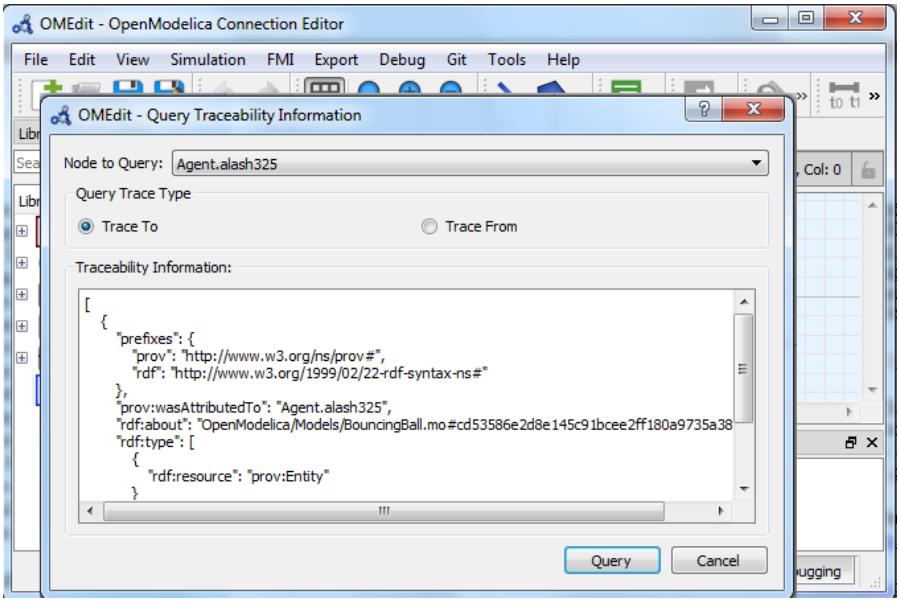

Figure 5.8: GUI to query traceability information from the traceability Daemon.

Entities (e.g., Modelica files, FMUs, modelDescription XML file) are shown in green, actions (e.g., model creation, FMU export, modelDescription XML import) are shown in yellow, agents (e.g., users with the names "Alachew", "Adrian", "Peter", and "Adeel") are shown in blue, and their relationships "what came from what" and "what used what" (e.g., "wasGeneratedBy", "wasDerivedFrom", "usedTool") are shown with red arrows.

In order to view and analyze traceability data, we have also designed a graphical user interface, shown in Figure 5.8, which allows the user to query traceability information (traces to and traces from) from the traceability Daemon to the database (via HTTP GET):

- http://localhost:8080/traces/from/<URI>/json and
- http://localhost:8080/traces/to/<URI>/json

5.7 Summary

This chapter has presented a framework for traceability and model management based on the OSLC specification standard combined with the Git version control system. All operations on artifacts of interest, integrated with different tools that are used in CPS design, are traced. The traceability information is exchanged with external tools through a standardized interface and format. A message schema is defined that ensures that all tools use the same format for sending their data. The traceability information is stored in a graph database that can be queried in order to generate various reports, such as impact analysis, variant handling, etc. A first prototype, which queries specific traceability links (traces to and traces from different entities, such as models, users, FMUs, or simulation results) from the database and displays the results to end-users in JSON format, is also presented.

Acknowledgments

This work has been supported by the European Union in the H2020 INTO-CPS project. Support from Vinnova in the ITEA3 OPENCPS project has been received. The OpenModelica development is supported by the open-source Modelica Consortium. Special thanks to Kenneth Lausdahl, Peter Niermann, Jos Höll, Carl Gamble, Oliver Möller, Etienne Brosse, Tom Bokhove, and Luis Diogo Couto for collaboration and valuable input on traceability-related tool design.

6 Advanced Modeling Simulation Analysis

This chapter is based on the following papers:

- Bernt Lie, Sudeep Bajracharya, **Alachew Mengist**, Lena Buffoni, Arun Kumar, Martin Sjölund, Adeel Asghar, Adrian Pop, Peter Fritzson. **API for Accessing OpenModelica Models From Python.** In Proceedings of 9th EUROSIM Congress on Modeling and Simulation, September 12-16, 2016, Oulu, Finland.

- Adeel Asghar, Andreas Pfeiffer, Arunkumar Palanisamy, **Alachew Mengist**, Martin Sjölund, Adrian Pop and Peter Fritzson. **Automatic Regression Testing of Simulation Models and Concept for Simulation of Connected FMUs in PySimulator.** In Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015.

6.1 Introduction

EOO languages such as Modelica have relatively little support for the advanced model analysis and synthesis needed for control systems design, particularly for optimal control problems (OCPs). Examples of such desirable analysis capabilities include (i) study of model sensitivity, (ii) random number generation and statistical analysis, (iii) Monte Carlo simulation, (iv) advanced plotting capabilities, (v) general optimization capabilities, (vi) linear analysis and control synthesis, etc. Scripting languages such as MATLAB and Python include most of these desirable analytical capabilities, and it is of interest to integrate Modelica models with such scripting languages. For the purposes of this work, we have chosen to use Python and OpenModelica due to their open-source availability.

A Python API for controlling Modelica simulation and analysis from Python was proposed in February 2015[1]. Based on this proposal, we have developed an initial version of a Python API for operating on Modelica models in Python. In this chapter, we present how to generate a Python object from an EOO Modelica model, how to set operating conditions, and how to run the simulation to produce a result object. Furthermore, we describe how to carry out linearization of a model object, the possibility of carrying out parameter sensitivity studies, and how to extract results from the result object to Python. The proposed solution has been tested on several industrially-relevant models, and its use for automatic analysis of Modelica models from Python is illustrated using a simple water tank model.

6.2 Description of the API

The API is described in the subsections below.

6.2.1 Python Class and Constructor

The name of the Python class that is used for operation on Modelica models is *ModelicaSystem*. This *class* is equipped with an object constructor with the same name as the class. In addition, the class is equipped with a number of methods for manipulating the instantiated objects. In this subsection, we discuss how to import the class, and how to use the constructor to instantiate an object. The object is imported from package OMPython, i.e., with Python commands.

```
>>>from OMPython import ModelicaSystem
```

Other Python packages to be used, such as *numpy*,*matplotlib*, *pandas*, etc. must be imported in a similar manner. The object constructor requires a minimum of 2 input arguments which are strings, and may need a third string input argument.

- The *first input argument* must be a string with the file name of the Modelica code, with Modelica file extension*.mo*. If the Modelica file is not in the current directory of Python, then the file path must also be included.

[1] Python API for Accessing OpenModelica Models, by B. Lie, February 20, 2015.

- The *second input argument* must be a string with the name of the Modelica model, including the namespace if the model is wrapped within a Modelica package.
- A *third input argument* is used if the Modelica model builds on other Modelica code, e.g., the Modelica Standard Library.

Example 1: Use of constructor. Suppose we have a Modelica model with name CSTR wrapped in a Modelica package *Reactors* — stored in file *Reactor.mo*:

```
package Reactors
        // ...
        model CSTR
                /// ...
        end CSTR;
        //
end Reactors;
```

If this model does not use any external Modelica code and the file is located in the current Python directory, the following Python code instantiates a Python object *mod*:

```
>>> mod = ModelicaSystem('Reactors.mo', 'Reactors.CSTR')
```

The user is free to choose any valid Python label name for the Python object. All methods of class *ModelicaSystem* refers to the instantiated object, in standard Python fashion. Thus, method *simulate()* is invoked with the Python command:

```
>>>mod.simulate()
```

In the subsequent overview of methods, the object name is not included. In practice, of course, it must be included in order to operate on the object in question. Methods may have no input arguments, one, or several input arguments. Methods may or may not return results — if the methods do not return results, the results are stored within the object.

6.2.2 Utility routines, converting Modelica ↔ FMU:

Two utility methods convert files between Modelica files with file extension *.mo* and Functional Mock-up Unit (FMU) files with file extension *.fmu*.

1. *convertMo2Fmu()* — method for converting the Modelica model of the object, say ModelName, into FMU file.
    - Required input arguments: none, operates on the Modelica file associated with the object.

- Optional input arguments:
  - *className*: string with the class name that should be translated,
  - *version*: string with FMU version, "1.0" or "2.0"; the default is "1.0".
  - *fmuType*: fmuType: string with FMU type, "me" (model exchange) or "cs" (co-simulation); the default is "me".
  - *fileNamePrefix*: string; the default is ćlassNamé.
  - *generatedFileName*: string, returns the full path of the generated FMU.
- Result: file *ModelName.fmu* in the current directory

2. *convertFmu2Mo(s)* — method for converting an FMU file into a Modelica file.

   - Required input arguments: string s, where s is the name of an FMU file, including extension *.fmu*.
   - Optional input arguments: a number of optional input arguments, e.g., the possibility to change working directory for the imported FMU files.
   - Result: Assume the name of the file is *fmuName.fmu*. Then file *fmuName_me_FMU.mo* is generated in the current Python directory.

3. *Getting and setting information:* Quite a few methods are dedicated to getting and setting information about objects. With two exceptions — *getQuantities()* and *getSolutions()* — the use of input arguments and results is identical for all get methods, while input arguments are used identically for all of the set methods, with results stored in the object.

   The Method *getQuantities()* does not accept input arguments, and returns a list of dictionaries, one dictionary for each quantity. Each dictionary has the following keys — with values being strings, too.

   - *Changeable* — value *'true'* or *'false'*,
   - *Description* — the string used in Modelica to describe the quantity, e.g., *'Mass in tank, kg'*,
   - *Name* — the name of the quantity, e.g., *'T'*, *'der(T)'*, *'n[1]'*, *'mod1.T'*, etc.,
   - *Value* — the value of the quantity, e.g., *'None'*, *'5.0'*, etc.,
   - *Variability* — *'continuous'*,*'parameter'*.

When applying the *Pandas method DataFrame* to the returned list of dictionaries, the result is a conveniently typeset table in Jupyter notebooks. Modelica *constants* are not included in the returned quantities. Standard get methods *getXXXs()*, where *XXXs* is in *Continuous, Parameters, Inputs, Outputs,SimulationOptions, OptimizationOptions,LinearizationOptions* are considered. Thus, methods *getContinuous(), getParameters()*, etc. Two Standard get methods are accepted.

- *getXXXs()*, i.e., without input argument, returns a dictionary with names as keys and values as ... values.
- *getXXXs(S)*, where S is a sequence of strings of names, returns a tuple of values for the specified names.

Getting solutions: We consider method getSolutions(). Two calling possibilities are accepted.

- *getSolutions()*, i.e., without input arguments, returns a list of strings of names of quantities for which there is a solution = time series.
- *getSolutions(S)*, where S is a sequence of strings of names, returns a tuple of values = 1D numpy arrays = time series for the specified names.

Setting methods: The information that can be set is a subset of the information that can be set. Thus, we consider methods *setXXXs()*, where *XXXs* is in *(Parameters, Inputs, SimulationOptions, OptimizationOptions, LinearizationOptions=, thus methods setParameters(), setInputs()*, etc. Two calling possibilities are accepted.

- *setXXXs(K)*, with K being a sequence of keyword assignments of type *quantity name = value*. Here, the quantity name could be a parameter name (i.e., not a string), an input name, etc.
  - For parameters and simulation/optimization/linearization options, the value should be a numerical value or a string (e.g., a string of ODE solver name such as *'dassl'*, etc.).
  - For inputs, the value could be a numerical value if the input is constant in the time range of the simulation,
  - For inputs, the value could alternatively be a list of tuples (t_j, u_j), i.e., $[(t_1, u_1), (t_2, u_2), ..., (t_N, u_N)]$ where the input varies linearly between $(t_j; u_j)$ and (t_{j+1}, u_{j+1}), where $t_j \leq t_{j+1}$, and where at most two subsequent time indices t_j, t_{j+1} can have the same value. As an example, $[..., (1, 10), (1, 20), ...]$ describes a perfect jump in input value from value 10 to value 20 at time instance 1.

  - This type of sequence of input arguments does not work for certain quantity names, e.g., *'der(T)', 'n[1]', 'mod1.T'*, because Python does not allow for label names *der(T), n[1], mod1.T*, etc.

- *setXXXs(**D)*, with *D* being a *dictionary* with quantity names as keywords and values as described with the alternative input argument *K*.

4. *Operating on Python object: simulation, optimization:* The following methods operate on the object, and have no input arguments. The methods have no return values, instead the results are stored within the object. To retrieve the results, method *getSolutions()* is used, as described previously.

   - *simulate()* — simulates the system with the given simulation options.
   - *optimize()* — optimizes the Optimica problem with the given optimization options.

5. *Operating on Python object: linearization:* The following methods are proposed for linearization:

   - *linearize()* — with no input argument, returns a tuple of 2D numpy arrays (matrices) A, B, C and D.
   - *getLinearInputs()* — with no input argument, returns a list of strings of names of inputs used when forming matrices B and D.
   - *getLinearOutputs()* — with no input argument, returns a list of strings of names of outputs used when forming matrices C and D.
   - *getLinearStates()* — with no input argument, returns a list of strings of names of states used when forming matrices A, B, C and

6.3 Case Study: Python API usage for Model Analysis

We consider the simple tank in Figure 6.1 filled with water. Water with initial mass $m(0)$ is emptied by gravity through a hole in the bottom at effluent mass flow rate $\dot{m}_e$, while at the same time water is filled into the tank at influent mass flow rate $\dot{m}_i$.

Our modeling objective is to find the liquid level h. This objective is illustrated by the functional diagram in Figure 6.2. The functional diagram depicts the causality of the system ("Tank with influent and effluent mass flow"), where inputs (green arrow) cause a change in the system and is observed at outputs (orange arrow)[2]. Here, the input variable is the influent

[2]Although Modelica is an acausal modeling language, it is useful to think in terms of causality during model development.

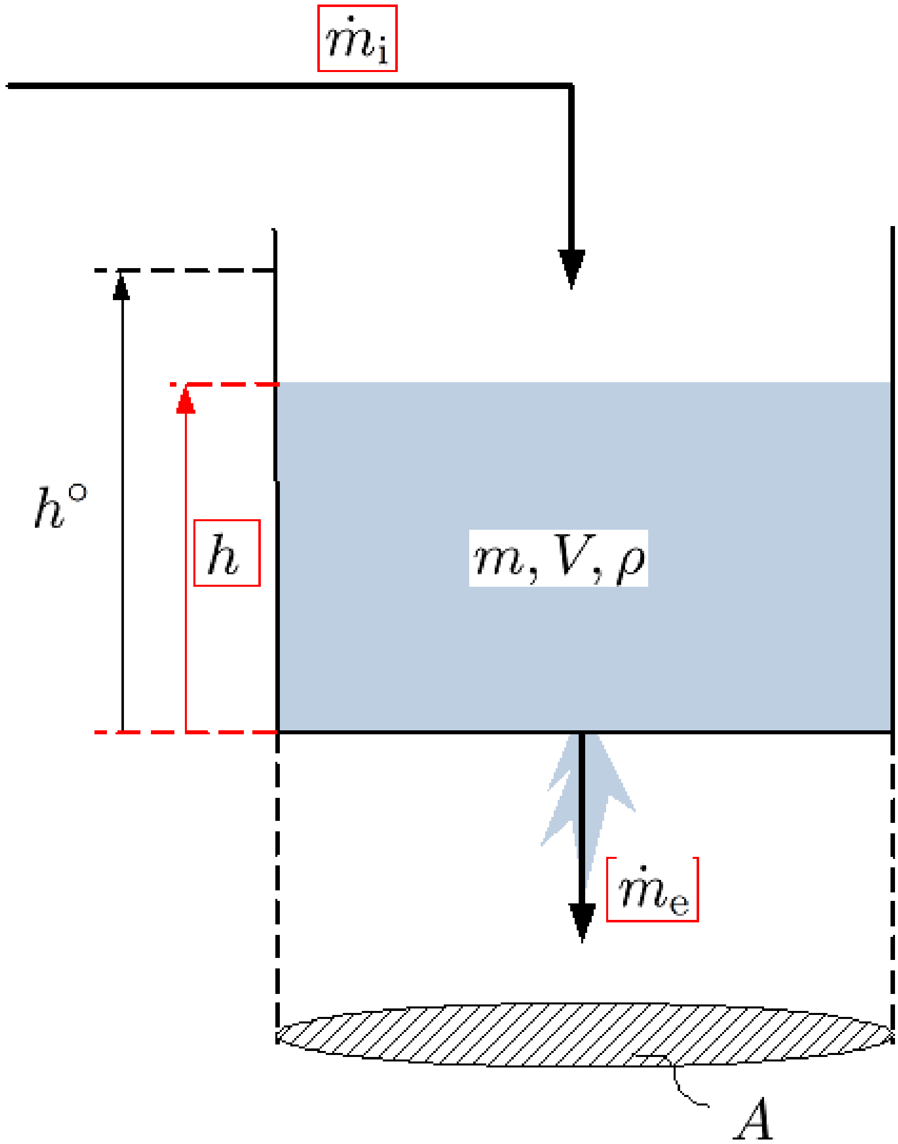

Figure 6.1: Driven water tank, with externally available quantities framed in red: initial mass is emptied through bottom at rate $\dot{m}_e$, while at the same time water enters the tank at rate $\dot{m}_i$

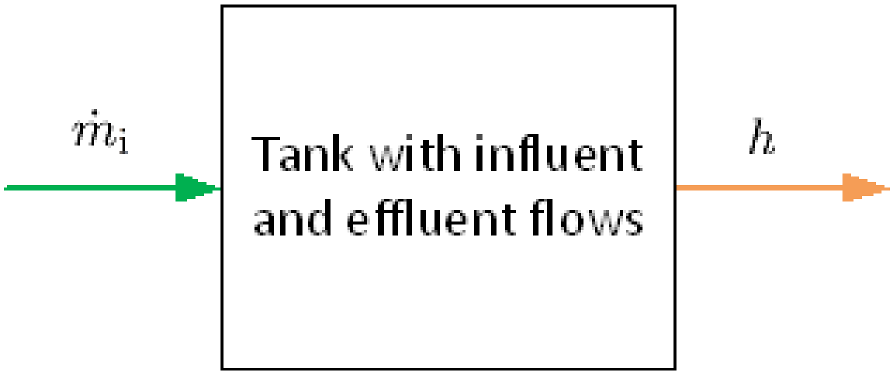

Figure 6.2: Functional diagram of tank with influent and effluent flow.

Table 6.1: Parameters for driven tank with constant cross sectional area.

Parameter	Value	Unit	Comment
p	1	kg/L	Density of liquid
A	5	dm^2	Constant cross sectional area
K	5	kg/s	Valve constant
h^{∇}	3	dm	Level scaling

mass flow rate $\dot{m}_i$, while the output variable is the quantity we are interested in, h.

6.3.1 Model Summary

The model can be summarized in a form suitable for implementation in Modelica as:

$$\begin{aligned}\frac{d_m}{d_t} &= \dot{m}_i - \dot{m}_e \\ m &= pV \\ V &= Ah \\ \dot{m}_e &= K\sqrt{\frac{h}{h^{\nabla}}}.\end{aligned}$$

To complete the model description, we need to specify model parameters and operating conditions. Model parameters (constants) are given in Table 6.1, and the operating conditions are given in Table 6.2.

Table 6.2: Operating condition for driven tank with constant cross sectional area.

Quantity	Value	Unit	Comment
$h(0)$	1.5	dm	Initial level
$m(0)$	$ph(0)A$	kg	Initial mass
$\dot{m}_i(t)$	2	kg/s	Nominal influent mass flow rate; may be varied

6.3.2 Water Tank Model expressed in Modelica

The Modelica code describes the core model of the tank, *ModWaterTank*, and consists of a first section where constants and variables are specified, and a second section where the model equations are specified.

```
model ModWaterTank
        // Main driven water tank model
        // author: Bernt Lie
        // University College of
        // Southeast Norway
        // April 18, 2016
        //
        // Parameters
        constant Real rho = 1 "Density";
        parameter Real A = 5 "Tank area";
        parameter Real K = 5 "Valve const";
        parameter Real h\_max = 3 "Scaling";
        // Initial state parameters
        parameter Real h\_0 = 1.5 "Init.level";
        parameter Real m\_0 = rho*h\_0*A "Init.mass";
        // Declaring variables
        // -- states
        Real m(start = m\_0, fixed = true) "Mass in tank, kg";
        // -- auxiliary variables
        Real V "Tank liquid volume, L";
        Real md\_e "Effluent mass flow";
        // -- input variables
        input Real md\_i "Influent mass flow";
        // -- output variables
        output Real h "Tank liquid level,dm";
        // Equations constituting the model
        equation
        // Differential equation
        der(m) = md\_i - md\_e;
```

```
        // Algebraic equations
        m = rho*V;
        V = A*h;
        md\_e = K*sqrt(h/h\_max);
end ModWaterTank;
```

As seen from the first section of model *ModWaterTank*, the model has 4 essential parameters ($rho - h_max$), one of which is a Modelica constant (rho) while the other 3 are design parameters (compare this to Table I). Furthermore, the model contains 2 "initial state" parameters, where 1 of them can be chosen at liberty, h_0, while the other one, m_0, is computed automatically from h_0, see Table II. The purpose of the "free parameter" h_0 is that it is easier for the user to specify level than mass. Also, free "initial state" parameters make it possible for the user to change the initial states from outside of model ModWaterTank, e.g., from Python.

Next, one variable is given with an initial value — the state m — which is initialized with the "initial state" parameter m_0. Then, 2 variables are defined as auxiliary variables (algebraic variables), V and md_e[3]

One input variable is defined — md_i — which is the influent mass flow rate $\dot{m}_i$, see Table 6.2. Inputs have the characteristic that their values are not specified in the core model — here*ModWaterTank*. Instead, their values must be given in an external model/code — we will specify this input in Python. Finally, 1 output is given — h.

In the second section of model *ModWaterTank*, the model equations exactly map the mathematical model given in Section 6.3.1. For purposes of illustration, the core model *ModWaterTank* is wrapped within a package named *WaterTank* and stored in file *WaterTank.mo*,

```
package WaterTank
        // Package for simulating
        // driven water tank
        // author: Bernt Lie
        // University College of
        // Southeast Norway
        // April 18, 2016
        //
        model ModWaterTank
        // Main driven water tank model
        // ....
        ....
        end ModWaterTank;
        // End package
end WaterTank;
```

[3] md is notation for m with a dot, $\dot{m}$, i.e., a mass flow rate.

6.3.3 Use of Python API

First, for example on Jupyter notebook, the following Python statements are executed:

```
from OMPython import ModelicaSystem
import numpy as np
import numpy.random as nr
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
LW = 2
```

Here, we use *NumPy* to handle simulation results, etc. The random number package will be used in a sensitivity/Monte Carlo study. The *magic* function *%matplotlib inline* is used to embed Matplotlib plots within the Jupyter notebook; to save these plots into files, simply right-click the plots. However, more options for saving files are available if the magic function is *excluded*, and the *plt.show()* command is instead added after the plot commands have been completed. The pandas library is used to illustrate presenting data in tables in a Jupyter notebook. Finally, label *LW* is used to give a conform line width in plots.

6.3.4 Basic Simulation of Model

We instantiate object *tank* by running the OMC Server with the following command:

```
tank = ModelicaSystem('WaterTank.mo','WaterTank.ModWaterTank')
```

Next, we are interested in which *quantities* are available in the model. Python prompt *»>* is used when Jupyter notebook actually uses *In[*]* — where * is some number, while the response in Jupyter notebook is prepended with *Out[*]*.

```
>>> q = tank.getQuantities()
>>> type(q)
list
>>> len(q)
11
>>> q[0]
{'Changeable': 'true',
'Description': 'Mass in tank, kg',
'Name': 'm',
'Value': None,
'Variability': 'continuous'}
>>> pd.DataFrame(q)
```

```
In [9]: pd.DataFrame(q)
Out[9]:
```

	Changeable	Description	Name	Value	Variability
0	true	Mass in tank, kg	wt.m	None	continuous
1	false	Mass in tank, kg	der(wt.m)	None	continuous
2	false	External input, passed on to instantiated mode...	_md_i	None	continuous
3	false	Tank liquid level, dm	wt.h	None	continuous
4	false	Effluent mass flow rate from tank, kg/s	wt.md_e	None	continuous
5	true	Cross sectional area of tank, dm2	wt.A	5.0	parameter
6	true	Valve constant, kg/s	wt.K	5.0	parameter
7	true	Initial tank level, dm	wt.h_0	1.5	parameter
8	true	Scaling level, dm	wt.h_s	3.0	parameter
9	false	Initial tank mass, kg	wt.m_0	None	parameter
10	false	Tank liquid volume, L	wt.V	None	continuous
11	false	Influent mass flow rate to tank, kg/s	wt.md_i	None	continuous

Figure 6.3: Typesetting of Data Frame of quantity list in Jupyter notebook.

The last command leads Jupyter notebook to typeset a tabular presentation of the quantities, Figure 6.3. The results in Figure 6.3 should be compared to the Modelica model in 6.3.2. Observe that Modelica *constants* are not included in the quantity list.

Next, we check the simulation options:

```
>>> tank.getSimulationOptions()
{'solver': 'dassl',
'startTime': 0.0,
'stepSize': 0.002,
'stopTime': 1.0,
'tolerance': 1e-06}
```

It should be observed that the *stepSize* is the frequency at which solutions are stored, and is not the step size of the solver. The number of data points *stored* is thus *(stopTime-startTime)/stepSize* with appropriate rounding. This means that if we increase the *stopTime* to a large number, we should also increase the *stepSize* to avoid storing a large volume of information.

To this end, we want to simulate the system for a long time, until the level reaches a steady state. Possible inputs are:

```
>>> tank.getInputs(){'md_i': None}
```

where value None implies that the available input, md_i, has not yet been set. We could use *None* as input, which will be interpreted as zero. But let us instead set $\dot{m}_i = 3$, simulate for a long time, and change the "initial state" parameter $h(0)$ to the steady state value of h:

```
>>> tank.setInputs(md_i=3)
>>> tank.setSimulationOptions\
(stopTime=1e4, stepSize=10)
>>> tank.simulate()
>>> h = tank.getSolutions('h')
>>> tank.setParameters(h_0 = h[-1])
```

Next, we set back the stop time to 10, and specify an input sequence with a couple of jumps:

```
>>> tank.setSimulationOptions\(stopTime=10, stepSize=0.02)
>>> tank.setInputs(md_i = [(0,3),(2,3),(2,4),(6,4),(6,2),(10,2)])
```

Finally, we simulate the model with the time varying input, and plot the result:

```
>>> tank.simulate()
>>> tm, h = tank.getSolutions('time','h')
>>> plt.plot(tm,h,linewidth=LW,color='blue', label=r'$h$')
>>> plt.title('Water tank level')
>>> plt.xlabel(r'time $t$ [s]')
>>> plt.ylabel(r'$h$ [dm]')
```

The result is displayed in Figure 6.4.

6.3.5 Parameter Sensitivity/Monte Carlo Simulation

It is of interest to study how the model behavior varies with varying uncertain parameter values, e.g., the effluent valve constant K. This can be done as follows:

```
>>> par = tank.getParameters()
>>> K = par['K']
>>> KK = K + (nr.randn(10)-0.5)*K/20
>>> tank.simulate()
>>> tm, h = tank.getSolutions('time','h')
>>> plt.plot(tm,h,linewidth = LW,color = 'red', label=r'$h$')
>>> for k in KK:
        tank.setParameters(K=k);
        tank.simulate()
        tm, h = tank.getSolutions('time','h')
        plt.plot(tm,h,linewidth=LW,
        color='red',linestyle='dotted',label='_nolabel_')
>>> plt.title('Tank level sensitivity')
>>> plt.xlabel(r'time $t$ [s]')
```

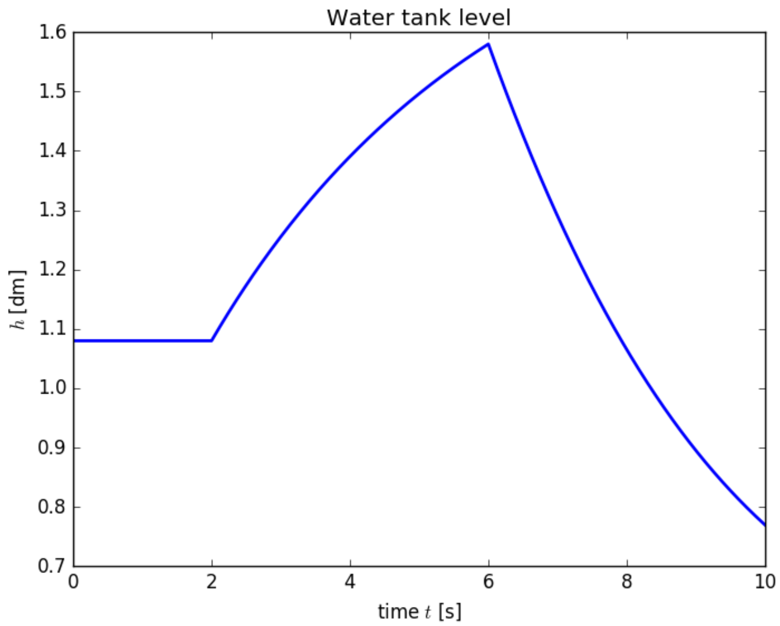

Figure 6.4: Tank level when starting from steady state, and $\dot{m}_i(t)$ varies in a straight line between the points $(tj, \dot{m}_i(t_j))$ given by the list $[(0;3);(2;3);(2;4);(6;4);(6;2);(10;2)]$.

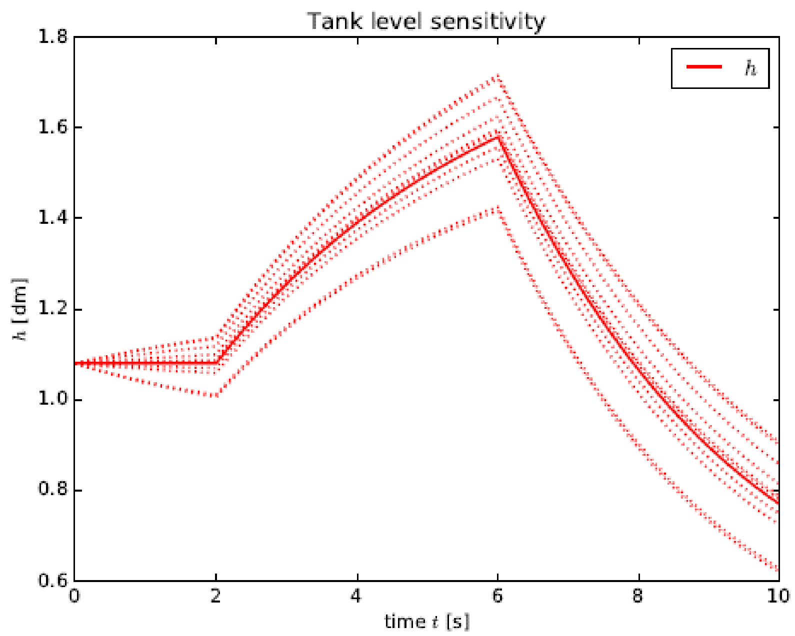

Figure 6.5: Uncertainty in tank level with a 5% uncertainty in valve constant K. The input is the same as in Figure 6.4

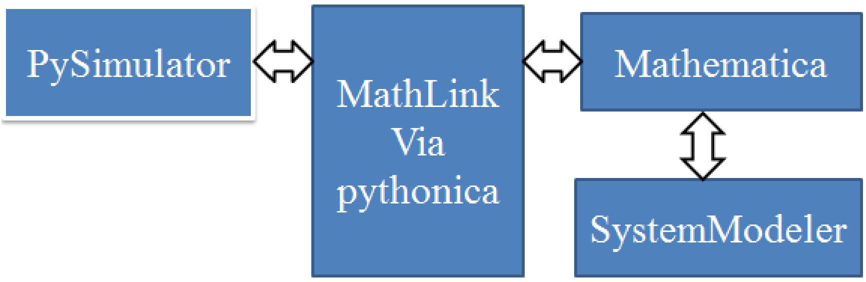

Figure 6.6: Communication setup with SystemModeler.

```
>>> plt.ylabel(r'$h$ [dm]')
>>> plt.legend()
```

The result is shown in Figure 6.5.

6.4 Integration of Wolfram SystemModeler Simulator in PySimulator

PySimulator supports simulation of models in FMU form or using different Modelica tools via extension plugins. Simulator plugins for tools such as Dymola, SimulationX, and OpenModelica are available from previous work. This section presents a new simulator plugin developed for Wolfram SystemModeler.

Wolfram SystemModeler has its own symbolic mathematical computation program, Mathematica [125], which can be used to perform actions via the Wolfram SystemModeler link (WSMLink) API [126], such as loading, compilation and simulation of models, or plotting of results. WSMLink provides functionality for integrating Wolfram SystemModeler and Mathematica with complete access to models and simulations.

Wolfram SystemModeler can be interfaced to other tools via MathLink [124], which is a library of functions that implement a protocol for sending and receiving Mathematica expressions. MathLink [48, 124] allows external programs to both call Mathematica and be called by Mathematica. It is possible to create a front end that implements your own user interface and communicates with the Mathematica kernel via MathLink [123].

Using the existing interface for simulator plugins in PySimulator, a new simulator plugin has been implemented: the Wolfram plugin. It enables PySimulator to load and numerically simulate Modelica models using Wolfram SystemModeler.

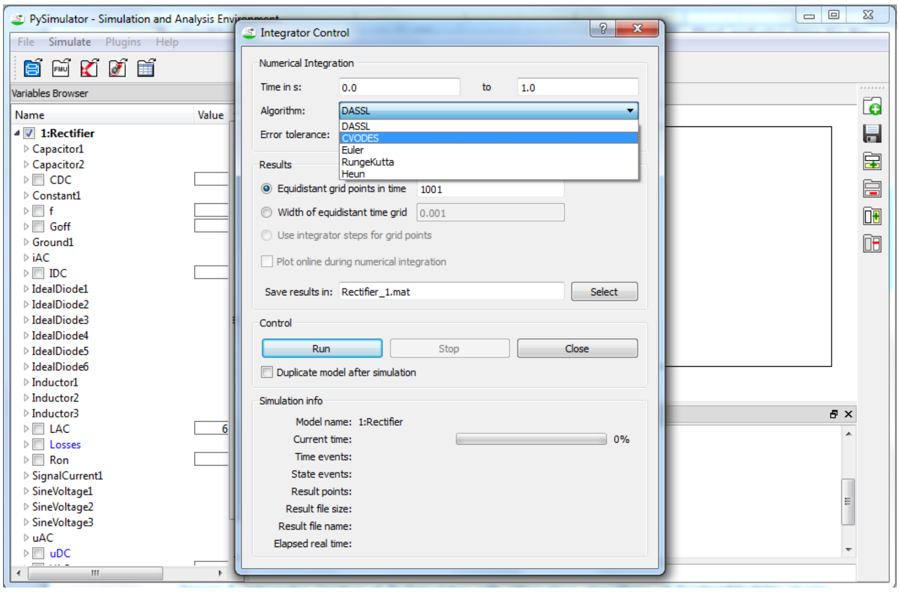

Figure 6.7: Integrator Control of PySimulator with integration algorithms of SystemModeler plugin.

The Wolfram plugin is integrated into PySimulator via MathLink and Pythonica [34], which connects to Mathematica and SystemModeler. We used the Wolfram SystemModeler API to support loading a Modelica model, simulating it, and reading the simulation setting file (*.sim*), which is an XML file, to build the variable tree in the variables browser of PySimulator. The overall communication setup with SystemModeler is given in Figure 6.6.

All of the simulator plugins of PySimulator are controlled by the same Integrator Control GUI. The Wolfram SystemModeler simulator supports five different numerical integration methods (*DASSL, CVODES, Euler, RungeKutta, and Heun*), all the simulation menu options are supported (*error tolerance, fixed step size*, etc.), see Figure 6.7.

The *start* and *stop time* for the integration algorithm can be changed and one of the integration algorithms can be selected. Depending on the integration algorithms the user can change the *error tolerance* or the *fixed step size* before running the simulation.

It is also possible to simulate the list of models using the Wolfram plugin. The existing PySimulator interface automatically includes the new plugin in the simulators list for simulating a list of models.

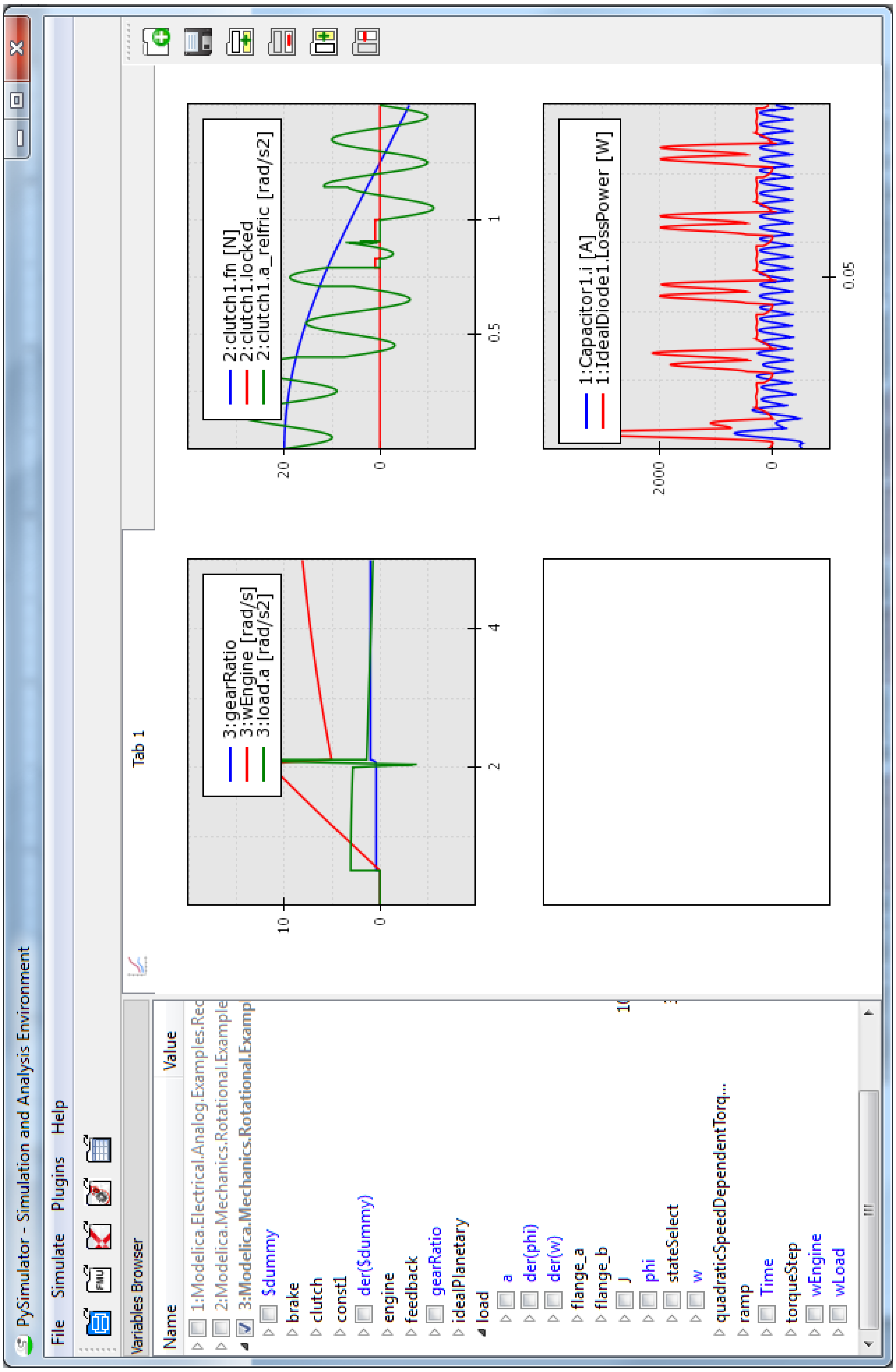

Figure 6.8: List of simulate models via Wolfram plugin.

6.5 Summary

In this chapter, we have introduced an enhanced Python interface for interacting with FMUs and Modelica models for further analysis and post-processing of simulation results. We have presented how a modeler can use the Python interface to simulate and access Modelica models using Python objects. The idea behind this interface is to provide the modeler with the ability to manipulate and exchange data with Modelica models before and after simulations. A case study of a Modelica model is presented to illustrate the tool's usage.

We have also presented a simulator plugin for a commercial simulation engine for Modelica models - the Wolfram SystemModeler within PySimulator. The existing PySimulator interface automatically includes the new plugin in the simulators list for simulating lists of models. We have successfully tested by loading all of the result files generated from the Wolfram plugin and verified the results by comparing with several other base line tools. The integration of the Wolfram SystemModeler simulator plugin uses the regression analysis framework within PySimulator to compare results of the same model generated from SystemModeler with several other tools or different versions of the same model from the SystemModeler tool.

Acknowledgments

Part of the PySimulator work is financed by the CleanSky Joint Undertaking project PyModSimA (JTI-CS-2013-2- SGO-02-064). This support is highly appreciated. The authors thank Jakub Tobolar (DLR Institute of System Dynamics and Control) for his tests and support of the regression testing feature in an earlier stage and his implementation of the automatic generation of the simulation setup file by Dymola.

7 Conclusions and Future Work

7.1 Conclusions

The main goal of this thesis was to design and implement tools and methods, up to the point of proof of concept and prototype demonstrations, which would increase the efficiency and quality of model-based development of complex and multi-domain cyber-physical systems.

With respect to the objective "to ensure automatic solution of dynamic optimization problems by reusing simulation models for optimization", we have developed a model-based dynamic optimization approach by integrating optimization into the model development process. The feasibility of our approach was demonstrated by a prototype implementation that was employed in the solution of industrially-relevant optimal control problems, including a diesel engine model. While the parameter sweep static design optimization method uses many simulation runs, the dynamic optimization approach presented in this thesis uses a direct optimization of a whole solution trajectory iteratively to obtain the optimal solution with minimum computation and time. OpenModelica coupling with CasADi has shown that it's possible to use an XML-based model exchange format for model-based dynamic optimization with state of the art optimization algorithms. The approach contributes to enabling mathematical simulation models expressed in Modelica with the Optimica language extension to be used efficiently for simulation-based optimization. The use of a language-neutral model exchange format simplifies tool interoperability and allows modelers to conduct experiments with different optimization algorithms and choose the one that is best suited for their particular problem, without the need to re-encode the problem formulation. As compared to traditional optimization frameworks, which typically require modelers to encode the model, the cost function, and the constraints in an algorithm-specific manner, the approach presented in this thesis significantly increases flexibility.

With respect to the objective "to ensure reusing and combining existing simulation models formalized by different experts in different modeling languages and tools for a unified system simulation", we have developed a general open-source graphical and textual editor, and a co-simulation framework for composite modeling and simulation of several connected subsystems using detailed models. Several tool-specific simulation sub-models can be integrated and connected by means of a composite model, represented in XML, which defines the physical interconnections between them. The approach is based on a general external interface definition that can be implemented for many different simulation tools using the TLM method. This enables de-coupling of sub-models from the full system and allows them to be independently simulated and coupled in a numerically stable way via co-simulation techniques. Currently, most simulation tools for model-based development of cyber-physical systems are bound to a specific tool vendor. An open-source modeling and co-simulation environment for composite models will change that, since it enables integration of models defined in a specific language from many different simulation tool vendors in the design process. It has been successfully implemented and tested for several simulation tools.

With respect to the objective "to support advanced simulation modeling analysis", we have enhanced the Python interface to simulate and access EOO Modelica models using Python objects for further simulation modeling analysis. Our tool, OMPython, is developed as a library using a standard distribution of Python and targeted to the OpenModelica modeling and simulation environment. However, general concepts can be applied to any other language and tool. In order to ensure reusability, only the standard Python libraries were used. From a modeler's perspective, the Python interface makes scripting, plotting, and analysis of results straightforward. This gives the modeler the possibility to use EOO simulation models together with the more powerful and easier to use API and Python libraries, e.g., for tasks such as control design and post processing of simulation results.

We have also extended the list of simulator plugins for PySimulator by implementing a plugin for Wolfram's SystemModeler. The integration of Wolfram SystemModeler simulator plugin uses the simulation result analysis tools within PySimulator. Hence, comparing simulation results of the same model generated from SystemModeler with several other tools or different versions of the same model from the SystemModeler tool can be applied. Comparing results of model simulations is very important for model portability and model evolution. This makes it possible for simulation models from SystemModeler to be safely utilized and integrated into different tools in the design process.

With respect to the objective "to ensure automatic traceability between requirements, simulation models, FMUs, and simulation results artifacts and keep track of changes and integration of product design tools with modeling and simulation tools", we have developed a tool-supported method for multi-domain collaborative modeling and traceability support throughout the developments in CPSs. A design and implementation for seamless tracing and interoperability of lifecycle artifacts in OpenModelica, integrated with the INTO-CPS tool-chain of CPS design, has been developed based on a linked data approach. A tool interoperability approach based on the Linked data method for traceability improves the reusability of simulation models between tools in distributed collaborative development flows. Hence, system designers and analysts with expertise in different domains can effectively collaborate on the design of complex systems. The approach presented in this thesis contributes to an important step in the integration of different modeling tools that are used in the whole tool-chain of CPS design, from systems modeling down to co-simulation and test automation. This can be used to support several activities such as impact analysis, component reuse, verification, and validation.

The message format and schema for the traceability information has been standardized in order to ensure that all tools use the same format for sending their trace data. This schema, together with the use of standardized specifications and formats, allows other tool vendors to easily integrate their tools into the INTO-CPS tool-chain traceability environment. Currently, the traceability data is stored in a graph database that can be queried in order to generate various reports, such as impact analysis. Furthermore, users can easily query this database to retrieve specific information about the links between different entities, such as requirements, users, test results or models (FMUs).

7.2 Future Work

Bibliography

[1] Pfeiffer A., Hellerer M., Hartweg S., Otter M., and Reiner M. "PySimulator - A Simulation and Analysis Environment in Python with Plugin Infrastructure". In: *Proceedings of 9th International Modelica Conference* (Sept. 2012).

[2] Modelon AB. "JModelica Home Page". In: (2017). URL: `http://www.jmodelica.org` (visited on 11/28/2017).

[3] J. Åkesson. "OptimicaAn Extension of Modelica Supporting Dynamic Optimization". In: *Proceedings of 6th International Modelica Conference* (Mar. 2008).

[4] J. Äkesson, Ärzén. K.-E., M. Gäfvert, T. Bergdahl, and H Tummescheit. "Modeling and Optimization with Optimica and JModelica.org—Languages and Tools for Solving Large-Scale Dynamic Optimization Problems". In: *Computers and Chemical Engineering* 34.11 (Nov. 2010), pp. 1737–1749.

[5] Johan Åkesson and Ola Slätteke. "Modeling, Calibration and Control of a Paper Machine Dryer Section". In: *Proceedings of the 5th International Modelica Conference.* (Sept. 2006).

[6] J. Andersson, J. Äkesson, F. Casella, and M Diehl. "Integration of CasADi and JModelica.org". In: *Proceedings of the 8th International Modelica Conference* (Mar. 2011).

[7] J. Andersson, J. Åkesson, and M Diehl. “CasADi - A symbolic package for automatic differentiation and optimal control”. In: *Recent Advances in Algorithmic Differentiation, Lecture Notes in Computational Science and Engineering book series (LNCSE, volume 87) 297-307* (2012). DOI: https://doi.org/10.1007/978-3-642-30023-3_27.

[8] Adeel Asghar, Andreas Pfeiffer, Arunkumar Palanisamy, Alachew Mengist, Martin Sjölund, Adrian Pop, and Peter Fritzson. “Automatic Regression Testing of Simulation Models and Concept for Simulation of Connected FMUs in PySimulator”. In: *Proceedings of the 11th International Modelica Conference* (Sept. 2015). DOI: 10.3384/ecp15118671.

[9] Modelica Association. “Modelica - A Unified Object-Oriented Language for Physical Systems Modeling - Language Specification Version 3.4, 2017”. In: (). URL: https://www.modelica.org/documents/ModelicaSpec34.pdf (visited on 11/28/2017).

[10] Modelica Association. “Modelica Association Home Page”. In: (2017). URL: https://www.modelica.org/association (visited on 11/28/2017).

[11] Modelica Association. “Modelica Standard Library GitHub Page”. In: (2017). URL: https://github.com/modelica/Modelica%20/ (visited on 11/28/2017).

[12] D.M. Auslander. “Distributed System Simulation with Bilateral Delay-Line Models”. In: *Journal of Basic Engineering (June 1968)* (June 1968), pp. 195–200.

[13] Mikael Axin, Robert Braun, Petter Krus, Alessandro dellÁmico, Björn Eriksson, Peter Nordin, Karl Pettersson, and Ingo Staack. “Next Generation Simulation Software using Transmission Line Elements”. In: *Proceedings of the Bath/ASME Symposium on Fluid Power and Motion Control (FPMC)* (Sept. 2010).

[14] Bernhard Bachmann, Lennart Ochel, Vitalij Ruge, Mahder Gebremedhin, Peter Fritzson, Vaheed Nezhadali, Lars Eriksson, and Martin Sivertsson. “Parallel Multiple-Shooting and Collocation Optimization with OpenModelica”. In: *Proceedings of the 9th International Modelica Conference* (Sept. 2012).

[15] A. Bakshi, V. K. Prasanna, and A. Ledeczi. “MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems”. In: *Proceeding LCTES ’01 Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems* 36.8 (Aug. 1998), pp. 82–93. DOI: 10.1145/384196.384210.

[16] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema. “Developing Applications Using Model-Driven Design Environments”. In: *IEEE Computer Society* 39.2 (Feb. 2006), pp. 33–40. DOI: `10.1109/MC.2006.54`.

[17] P. I. Barton and C. C Pantelides. “Modeling of combined discrete/continuous processes”. In: *AIChE Journal* 40.6 (1994), pp. 966–979. DOI: `10.1002/aic.690400608`.

[18] Paul Inigo Barton. “The Modelling and Simulation of Combined Discrete/Continuous Processes”. In: *PhD thesis, Department of Chemical Engineering, Imperial Collage of Science, Technology and Medicine* (1992).

[19] Tim Berners-Lee. “Linked Data”. In: *W3C* (July 2006). URL: `https://www.w3.org/DesignIssues/LinkedData.html` (visited on 11/28/2017).

[20] L.T. Biegler. “Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes”. In: *Society for Industrial Applied Mathematics* (2010).

[21] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Claußand H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. “The Functional Mockup Interface for Tool independent Exchange of Simulation Models”. In: *Proceedings of the 8th International Modelica Conference* (2011).

[22] H.G. Bock and Plitt K.J. “A multiple shooting algorithm for direct solution of optimal control problems”. In: *Proceedings of the 9th IFAC World Congress* 17.2 (July 1984), pp. 1603–1608. DOI: `https://doi.org/10.1016/S1474-6670(17)61205-9`.

[23] Christian D. Bodemann and Filippo De Rose. “The Successful Development Process with Matlab Simulink in the Framework of ESAś ATV Project”. In: *55th International Astronautical Congress of the International Astronautical Federation, the International Academy of Astronautics, and the International Institute of Space Law, International Astronautical Congress (IAF)* (2004). DOI: `https://doi.org/10.2514/6.IAC-04-U.3.B.03`.

[24] Jan F. Broenink. “20-sim software for hierarchical bond-graph/block-diagram models”. In: *Simulation Practice and Theory* 7.5-6 (Dec. 1999), pp. 481–492. DOI: `10.1016/S0928-4869(99)00018-X`.

[25] Christopher Brooks, Chihhong Patrick Cheng, Thomas Huining Feng, Edward A. Lee, and Reinhard von Hanxleden. “Model Engineering Using Multimodeling”. In: *Proceedings of the 1st International Workshop on Model Co-Evolution and Consistency Management (MCCM 08)* UCB/EECS-2008-39 (Sept. 2008), pp. 611–633.

[26] Dag Bruck, Hilding Elmqvist, Hans Olsson, and Sven Erik Mattsson. “Dymola for Multi-Engineering Modeling and Simulation”. In: *Proceedings of the 2nd International Modelica Conference* (2002).

[27] Lena Buffoni, Adrian Pop, and Alachew Mengist. “Traceability and Impact Analysis in Requirement Verification”. In: *Proceedings of the 8th International Workshop* (Dec. 2017).

[28] Ernst Christen and Kenneth Bakalar. “VHDL-AMS - A Hardware Description Language for Analog and Mixed-Signal Applications”. In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* (1999).

[29] D. de Cogan, W.J. OĆonnor, and S. Pulko. “Transmission Line Matrix in Computational Mechanics”. In: *Taylor & Francis* (2006).

[30] Open-Source Modelica Consortium. “A non-profit, non-governmental organization with the aim of developing and promoting the development and usage of the OpenModelica”. In: (2017). URL: `https://openmodelica.org/home/consortium` (visited on 11/28/2017).

[31] Open-Source Modelica Consortium. “OpenModelica Home Page”. In: (2017). URL: `https://openmodelica.org` (visited on 11/28/2017).

[32] Controllab. “20-sim - Modeling and Simulation Environment for Mechatronic Systems”. In: (). URL: `http://www.20sim.com/` (visited on 11/28/2017).

[33] V. Duindam, A. Macchelli, S. Stramigioli, and H Bruyninckx. “Modeling and Control of Complex Physical Systems”. In: *Springer-Verlag Berlin Heidelberg* (2011). DOI: `10.1007/978-3-642-03196-0`.

[34] Benjamin Edwards. “Pythonica - Python package for the Sane Interface between Mathematica and Python via MathLink.” In: (2012). URL: `https://github.com/bjedwards/pythonica` (visited on 11/28/2017).

[35] Johan Eker, Jorn Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Sonia Sachs, and Yuhong Xiong. “Taming heterogeneity - the Ptolemy approach”. In: *Proceedings of the IEEE* 91.1 (Jan. 2003), pp. 127–144.

[36] Hilding Elmqvist, Sven Erik Mattsson, and Martin Otter. "Modelica - a language for physical system modeling, visualization and interaction". In: *Proceedings of the 1999 IEEE International Symposium on Computer-Aided Control System Design* (1999). DOI: 10.1109/CACSD.1999.808720.

[37] Process Systems Enterprise. "gPROMS- Advanced Process Modeling platform". In: (). URL: https://www.psenterprise.com/products/gproms (visited on 11/28/2017).

[38] J. A. Estefan. "Survey of Candidate Model-Based Systems Engineering (MBSE) Methodologies, rev. B". In: *International Council on Systems Engineering (INCOSE)* (2015).

[39] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. "Methods Progress Report 1". In: *Technical report, INTO-CPS Deliverable, D3.1b* (Dec. 2015). (Visited on 11/28/2017).

[40] Rüdiger Franke, Manfred Rode, and Klaus Krgüer. "On-line Optimization of Drum Boiler Startup". In: *Proceedings of the 3rd International Modelica Conference.* (Nov. 2003), pp. 287–296.

[41] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nystr ö m, L. Saldamli, D. Broman, and A Sandholm. "OpenModelica – A Free Open-Source Environment for System Modeling, Simulation, and Teaching". In: *Proceedings of the 2006 IEEE Conference on Computer Aided Control System Design* (Oct. 2006).

[42] Peter Fritzson. "Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach". In: *Wiley-IEEE Press* (2014).

[43] Peter Fritzson and Vadim Engelson. "Modelica - A Unified Object-Oriented Language for System Modeling and Simulation". In: *Proceedings of the 12th European Conference on Object-Oriented Programming(ECOOP)* 144 (July 1998).

[44] Peter Fritzson, Johan Gunnarsson, and Mats Jirstrand. "MathModelica An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming". In: *Proceedings of the 2nd International Modelica Conference* (2002).

[45] A.K Ganeson. "Design and Implementation of a User Friendly OpenModelica - Python interface". In: *Masterś Thesis, Department of Computer and Information Science, Linköping University* (2012).

[46] Anand Ganeson, Peter Fritzson, Olena Rogovchenko, Adeel Asghar, Martin Sjölund, and Andreas Pfeiffer. "Python Interface and its use in PySimulator". In: *Proceedings of 9th International Modelica Conference* (Sept. 2012).

[47] P.J. Gawthrop and G.P Bevan. "Bond-graph modelling: A tutorial introduction for control engineers". In: *IEEE Control Systems Magazine* 27.2 (1999), pp. 24–45. DOI: 10.1109/MCS.2007.338279.

[48] Todd Gayley. "A MathLink Tutorial". In: (). URL: http://edenwaith.com/development/tutorials/mathlink/ML_Tut.pdf (visited on 11/28/2017).

[49] ITI GmbH. "SimulationX Modeling and Simulation Tool". In: (). URL: https://www.simulationx.com/ (visited on 11/28/2017).

[50] Verified Systems International GmbH. "RT-Tester: A Test Automation Tool". In: (). URL: https://www.verified.de/products/rt-tester/ (visited on 11/28/2017).

[51] Peter Gorm-Larsen, Casper Thule, Victor Bandur Kenneth Lausdahl and, Carl Gamble, Etienne Brosse, Andrey Sadovykh, Alessandra Bagnato, and Luis Diogo Couto. "Integrated Tool Chain for Model-Based Design of Cyber-Physical Systems: The INTO-CPS project". In: *Proceedings of the 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)* (Apr. 2016). DOI: 10.1109/CPSData.2016.7496424.

[52] IEEE 1706.1 Working Group. "IEEE Std 1076.1-1999, IEEE Standard VHDL Analog and Mixed-Signal Extensions". In: *IEEE Press* (1999).

[53] Object Management Group. "OMG Unified Modeling LanguageTM (OMG UML), Superstructure". In: (). URL: http://www.omg.org/spec/SysML/1.3/ (visited on 11/28/2017).

[54] Object Management Group. "System modeling language specification v1.3". In: (). URL: http://www.omg.org/spec/SysML/1.3/ (visited on 11/28/2017).

[55] Tom Heath and Christian Bizer. "Linked Data: Evolving the Web into a Global Data Space". In: *Synthesis Lectures on the Semantic Web: Theory and Technology* 1.1 (2011), pp. 1–136. DOI: 10.2200/S00334ED1V01Y201102WBE001.

[56] HOPSAN. "The HOPSAN Simulation Program, Users Manual." In: *Linköping University, LiTH-IKP-R-387.* (1985).

[57] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. "ACADO Toolkit - An Open-Source Framework for Automatic Control and Dynamic Optimization". In: *Optimal Control Methods and Application* 32.3 (2011), pp. 298–312. DOI: 10.1002/oca.939.

[58] INCOSE. "Systems Engineering Vision 2020 (INCOSE-TP-2004-004-02.Version 2.03 ed.)" In: *INCOSE* (2007).

[59] into-cps.au.dk. "Integrated Tool Chain for Model-Based Design of Cyber-Physical Systems". In: *INTO-CPS Project* (2015). URL: http://intocps.au.dk/ (visited on 11/28/2017).

[60] J.C. Jensen, D.H. Chang, and E.A. Lee. "A Model-Based Design Methodology for Cyber Physical Systems". In: *Proceedings of the 1st IEEE Workshop on Design, Modeling and Evaluation of Cyber Physical Systems* (2011).

[61] P.B. Johns and M.A. OBrian. "Use Of The Transmission Line Modelling (T.L.M) Method To Solve Nonlinear Lumped Networks". In: *The Radio And Electronic Engineer(1980)* 50.1-2 (1980), pp. 59–70.

[62] Json.org. "JavaScript Object Notation (JSON)". In: (). URL: `http://www.json.org/` (visited on 11/28/2017).

[63] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. "Model-Integrated Development of Embedded Software". In: *Proceedings of the IEEE* 91.1 (Jan. 2003), pp. 145–164. DOI: `10.1109/JPROC.2002.805824`.

[64] Yongjoo Kim, Kyuseok Kim, Youngsoo Shin, Taekyoon Ahn, and Kiyoung Choi. "An Integrated Cosimulation Environment for Heterogeneous Systems Prototyping". In: *Design Automation for Embedded Systems* 3.2-3 (June 1998), pp. 163–186. DOI: `https://doi.org/10.1023/A:1008842424479`.

[65] Per-Ola Larsson, Johan Åkesson, Staffan Haugwitz, and Niklas Andersson. "Modeling and Optimization of a Grade Change for Multistage Polyethylene Reactors". In: *Proceedings of the 18th IFAC World Congress.* 44.1 (Jan. 2011), pp. 12331–12336. DOI: `https://doi.org/10.3182/20110828-6-IT-1002.00131`.

[66] Kenneth Lausdahl, Peter Niermann, Jos Höll, Carl Gamble, Oliver Möller, Etienne Brosse, Tom Bokhove, Luis Diogo Couto, Adrian Pop, and Christian König. "NTOCPS Traceability Design". In: *Technical report, INTO-CPS Deliverable, D4.2d* (Dec. 2016). (Visited on 11/28/2017).

[67] OPTEC K.U. Leuven. "ACADO Home Page." In: (). URL: `http://acado.github.io/` (visited on 11/28/2017).

[68] Bernt Lie, Sudeep Bajracharya, Alachew Mengist, Lena Buffoni, ArunKumar Palanisamy, Martin Sjölund, Adeel Asghar, Adrian Pop, and Peter Fritzson. "API for Accessing OpenModelica Models From Python". In: *Proceedings of 9th EUROSIM Congress on Modelling and Simulation* (Sept. 2016).

[69] linkeddata.org. "Linked Data - Connect Distributed Data across the Web". In: (2006). URL: `http://linkeddata.org/` (visited on 11/28/2017).

[70] Elaasar M. and Neal A. "Integrating Modeling Tools in the Development Lifecycle with OSLC: A Case Study". In: *Model-Driven Engineering Languages and Systems (MODELS) Springer Berlin Heidelberg* 8107 (2013). Ed. by Moreira A., Schätz B., Gray J., Vallecillo A., and Clarke P. (eds), pp. 154–169. DOI: https://doi.org/10.1007/978-3-642-41533-3_10.

[71] Frank Manola and Eric Miller. "RDF Primer. W3C Recommendation. World Wide Web Consortium". In: *W3C* (Feb. 2004). URL: https://www.w3.org/TR/2004/REC-rdf-primer-20040210/ (visited on 11/28/2017).

[72] Maplesoft. "MapleSim – Advanced System Level Modeling". In: (). URL: https://www.maplesoft.com/products/maplesim/ (visited on 11/28/2017).

[73] Wolfram Mathcore. "Wolfram SystemModeler-Modeling and Simulation Tool". In: (). URL: http://wolfram.com/system-modeler/ (visited on 11/28/2017).

[74] MathWorks. "Simulink - Simulation and Model-Based Design". In: (). URL: https://se.mathworks.com/products/simulink.html (visited on 11/28/2017).

[75] Jirstrand Mats, Johan Gunnarsson, and Peter Fritzson. "A New Modeling and Simulation Environment for Modelica". In: *Proceedings of the Third International Mathematica Symposium(IMS)* (1999).

[76] Alachew Mengist, Adeel Asghar, Adrian Pop, Peter Fritzson, Willi Braun, Alexander Siemers, and Dag Fritzson. "An Open-Source Graphical Composite Modeling Editor and Simulation Tool Based on FMI and TLM Co-Simulation". In: *Proceedings of the 11th International Modelica Conference* (Sept. 2015). DOI: 10.3384/ecp15118181.

[77] Alachew Mengist, Adrian Pop, Adeel Asghar, and Peter Fritzson. "Traceability Support in OpenModelica Using Open Services for Lifecycle Collaboration (OSLC)". In: *Proceedings of the 12th International Modelica Conference* (May 2017).

[78] MODELISAR. "Functional Mock-up Interface, Version 2.0. Interface specification". In: *MODELISAR* (2017).

[79] Luc Moreau, Paolo Missier, James Cheney, and Stian Soiland-Reyes. "An Overview of the PROV Family of Documents". In: *W3C* (Apr. 2013). URL: https://www.w3.org/TR/prov-n/ (visited on 11/28/2017).

[80] Iakov Nakhimovski. “Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis”. In: *Dissertation No. 1009, Linköpings universitet* (Jan. 2006). URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.429.3525&rep=rep1&type=pdf` (visited on 11/28/2017).

[81] Inc Neo Technology. “Neo4j Database”. In: (2007). URL: `https://neo4j.com/` (visited on 11/28/2017).

[82] M. Oh and Costas C. Pantelides. “A modelling and Simulation Language for Combined Lumped and Distributed Parameter Systems”. In: *Computers and Chemical Engineering* 20.6-7 (1996), pp. 611–633. DOI: `https://doi.org/10.1016/0098-1354(95)00196-4`.

[83] Open-services.net. “Open Services for Lifecycle Collaboration - Lifecycle Integration Inspired by the Web”. In: (2008). URL: `http://open-services.net/` (visited on 11/28/2017).

[84] Krus P. “Modelling of Mechanical Systems Using Rigid Bodies and Transmission Line Joints”. In: *Transactions of ASME, Journal of Dynamic Systems Measurement and Control* (Dec. 1999).

[85] Krus P. and Jansson A. “Distributed Simulation of Hydromechanical Systems”. In: *Proceedings of the 3rd Bath International Fluid Power Workshop* (1990).

[86] Parrotto, Åkesson J. R., and F. Casella. “An XML representation of DAE systems obtained from continuous-time Modelica models”. In: *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools* (Oct. 2010).

[87] Jan Peleska. “Industrial-Strength Model-Based Testing - State of the Art and Current Challenges”. In: *Electronic Proceedings in Theoretical Computer Science abs/1303.1006, 3-28 (2013)* (2013).

[88] Jan Peleska, Elena Vorobev, Florian Lapschies, and Cornelia Zahlten. “Automated Model-Based Testing with RT-Tester”. In: *Universitat Bremen, Technical Report* (2011). (Visited on 11/28/2017).

[89] M. Perera, Anushka S., T.A. Hauge, and C.F. Pfeiffer. “Parameter and State Estimation of Large-Scale Complex Systems Using Python Tools”. In: *Modeling, Identification and Control* 36.3 (2015), pp. 189–198. DOI: `http://dx.doi.org/10.4173/mic.2015.3.6`.

[90] M. Perera, Anushka S., B. Lie, and C.F. Pfeiffer. “Structural Observability Analysis of Large Scale Systems Using Modelica and Python”. In: *Modeling, Identification and Control* 36.1 (2015), pp. 53–65. DOI: `http://dx.doi.org/10.4173/mic.2015.1.4`.

[91] Jan Poland, Alf. J. Isaksson, and P. Aronsson. “Building and Solving Nonlinear Optimal Control and Estimation Problems”. In: *Proceedings of the 7th International Modelica Conference.* (Sept. 2009). DOI: 10.3384/ecp09430004.

[92] A. Pop and P. Fritzson. “ModelicaXML:A Modelica XML Representation with Applications”. In: *Proceedings of the 3rd International Modelica Conference* (Nov. 2003).

[93] Adrian Pop, Martin Sjölund, Adeel Ashgar, Peter Fritzson, and Francesco Casella. “Integrated Debugging of Modelica Models”. In: *Modeling, Identification and Control* 35.2 (2014), pp. 93–107. DOI: 10.4173/mic.2014.2.3.

[94] K. Prölss, H. Tummescheit, S. Velut, Y. Zhu, J. Åkesson, and C. D. Laird. “Models of a post-combustion absorption unit for simulation; optimization and non-linear model predictive control schemes”. In: *Proceedings of the 8th International Modelica Conference* (Mar. 2011). DOI: 10.3384/ecp1106364.

[95] J. Reedy and S. Lunzman. “Model Based Design Accelerates the Development of Mechanical Locomotive Controls”. In: *SAE Technical Paper 2010-01-1999* (2010). DOI: https://doi.org/10.4271/2010-01-1999.

[96] Leonard Richardson and Sam Ruby. “RESTful Web Services”. In: *OŔeilly* (2007).

[97] Levon Saldamli and Peter Fritzson. “A Modelica-Based Language for Object-Oriented Modeling with Partial Differential Equations”. In: *Proceedings of the 4th International EuroSim Congress* (June 2001).

[98] Levon Saldamli and Peter Fritzson. “Domains and partial differential equations in modelica”. In: *Proceedings of the 48th Scandinavian Conference on Simulation and Modeling (SIMS)* (Oct. 2001).

[99] R.S Sandford Friedenthal Alan Moore. “A Practical Guide to SysML”. In: *Morgan Kaufman OMG Press, Friendenthal, Sanford, First edn. (2008), ISBN 978-0-12-374379-4* (1968).

[100] W. Schamai. “Modelica Modeling Language (ModelicaML) A UML Profile for Modelica. Technical Report”. In: *Linköping University Electronic Press* (2009).

[101] Wladimir Schamai. “Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool”. In: *Department of Computer and Information Science, Linköping University* (2013). ISSN: 0345-7524. DOI: 10.3384/diss.diva-98107.

[102] Alachew Shitahun. “Template Based XML and Modelica Unparsers in OpenModelica”. In: *Master thesis. Linköping University* (Aug. 2012).

[103] Alachew Shitahun, Vitalij Ruge, Mahder Gebremedhin, Bernhard Bachmann, Lars Eriksson, Joel Andersson, Moritz Diehl, and Peter Fritzson. "Model-Based Optimization with OpenModelica and CasADi". In: *Proceedings of IFAC Conference in Tokyo* (Sept. 2013).

[104] Alachew Shitahun, Vitalij Ruge, Mahder Gebremedhin, Bernhard Bachmann, Lars Eriksson, Joel Andersson, Moritz Diehl, and Peter Fritzson. "Tool Demonstration – OpenModelica and CasADi for Model-Based Dynamic Optimization". In: *Proceedings of the 5th International Workshop on Equation-Based Ob jectOriented Modeling Languages and Tools* (Apr. 2013).

[105] A. Siemers and D. Fritzson. "A meta-modeling environment for mechanical system co-simulations". In: *The 48th Scandinavian Conference on Simulation and Modeling (SIMS 2007)* (Oct. 2007).

[106] A. Siemers, P. Fritzson, and D. Fritzson. "Meta-modeling for multiphysics co-simulations applied for openmodelica". In: *Proceedings of the ANIPLA 2006 International Congress on Methodologies for Emerging Technologies in Automation* (Nov. 2006).

[107] Alexander Siemers, Dag Fritzson, and Iakov Nakhimovski. "General meta-model based co-simulations applied to mechanical systems". In: *Simulation Modelling Practice and Theory* 17.4 (Apr. 2009), pp. 612–624.

[108] Alexander Siemers, Iakov Nakhimovski, and Dag Fritzson. "Metamodelling of Mechanical Systems with Transmission Line Joints in Modelica". In: *Proceedings of the 4th International Modelica Conference* (Mar. 2005). Ed. by Gerhard Schmitz.

[109] M. Sivertsson and L. Eriksson. "Time and Fuel Optimal Power Response of a Diesel-Electric Powertrain". In: *E-CoSM'12 – IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling* (2012). DOI: `https://doi.org/10.3182/20121023-3-FR-4025.00036`.

[110] Martin Sjölund. "Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models". In: *Doctoral thesis No 1664, Linköping University, Department of Computer and Information Science* (2015).

[111] Softeam. "Modelio Open-Source Modelling Environment". In: (). URL: `http://www.modelio.org/` (visited on 11/28/2017).

[112] MSC Software. "ADAMS - The Multibody Dynamics Simulation Solution". In: (). URL: `http://www.mscsoftware.com/products/adams.cfm` (visited on 11/28/2017).

[113] L-E. Stacke, D. Fritzson, and P. Nordling. "BEAST - a rolling bearing simulation tool". In: *Proceedings of the Institution of Mechanical Engineers Part K Journal of Multi-body Dynamics 213(2):63-71* (1999). DOI: 10.1243/1464419991544063.

[114] Dassault Systemes. "Dymola multi-engineering modeling and simulation". In: (). URL: https://www.3ds.com/products-services/catia/products/dymola (visited on 11/28/2017).

[115] J. Tamimi and P. Li. "A combined approach to nonlinear model predictive control of fast systems". In: *Journal of Process Control* 20.9 (Oct. 2010), pp. 1092–1102. DOI: https://doi.org/10.1016/j.jprocont.2010.06.002.

[116] Michael Tiller. "Introduction to Physical Modeling with Modelica". In: *Springer* (May 2001). DOI: 10.1007/978-1-4615-1561-6.

[117] TNI-Software. "Cosimate co-simulation software". In: (). URL: http://www.tni-software.com (visited on 11/28/2017).

[118] M. Verhoef, P.G. Larsen, and J Hooman. "Modeling and Validating Distributed Embedded Real-Time Systems with VDM++". In: *Formal Methods. FM 2006.Lecture Notes in Computer Science* 4085 (2006). DOI: https://doi.org/10.1007/11813040_11.

[119] A. Wächter and L.T Biegler. "On the implementation of an interior-point filter line-search algorithm for large scale nonlinear programming. Mathematical Programming". In: *Springer-Verlag* 106.1 (Mar. 2006), pp. 25–27. DOI: https://doi.org/10.1007/s10107-004-0559-y.

[120] Andreas Wächter and Carl Laird. "IPOPT Home Page." In: (Aug. 2005). URL: https://projects.coin-or.org/Ipopt (visited on 11/28/2017).

[121] Schamai Wladimir, Peter Fritzson, Chris Paredis, and Adrian Pop. "Towards Unified System Modeling and Simulation with ModelicaML Modeling of Executable Behavior Using Graphical Notations". In: *Proceedings of the 7th International Modelica Conference* (Sept. 2009).

[122] Schamai Wladimir, Philipp Helle, Peter Fritzson, and Christiaan J.J. Paredis. "Virtual Verification of System Designs against System Requirements". In: *Proceedings of the 2010 international conference on Models in software engineering* (2010). DOI: 10.3384/ecp12076385.

[123] Wolfram. "MathLink Development In C". In: (). URL: http://library.wolfram.com/infocenter/Books/8516/MathLinkDevelopmentInC.pdf (visited on 11/28/2017).

[124] Wolfram. "MathLink Mathematica Functions". In: (). URL: https://reference.wolfram.com/legacy/v6/guide/MathLinkMathematicaFunctions.html (visited on 11/28/2017).

[125] Wolfram. "Wolfram Mathematica Mathematical Symbolic Computation Program". In: (). URL: https://www.wolfram.com/mathematica/ (visited on 11/28/2017).

[126] Wolfram. "WSMLink - Wolfram SystemModeler Link". In: (). URL: http://reference.wolfram.com/system-modeler/WSMLink/guide/WSMLink.html (visited on 11/28/2017).