KTH Industrial Engineering
and Management

Modeling and Simulation of Physical Systems in a Mechatronic Context

CARL-JOHAN SJÖSTEDT

Doctoral Thesis
Stockholm, Sweden 2009

TRITA-MMK 2009:12
ISSN 1400-1179
ISRN/KTH/MMK/R-09/12-SE
ISBN 978-91-7415-361-3

KTH School of Industrial Engineering
and Management
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i maskinkonstruktion tisdagen den 9 juni 2009 klockan 13.00 i Salongen, KTH Biblioteket, Kungliga Tekniska Högskolan, Osquars Backe 31, Stockholm.

Tryck: Universitetsservice US AB

Abstract

This thesis gives different views on the modeling and simulation of physical systems, especially together with embedded systems, forming mechatronic systems. The main considered application domain is automotive. One motivation behind the work is to find suitable representations of physical systems to be used in an architectural description language for automotive embedded systems, EAST-ADL2, which is implemented as a UML2 profile, and uses concepts from both UML and SysML.

As a part of the thesis, several languages and tools are investigated, including bond graphs, MATLAB/Simulink, Ptolemy II, Modelica, MATLAB/Simscape and SysML. For SysML, the modeling of continuous-time systems and how it relates to MATLAB/Simulink and Modelica is evaluated. A case study of an electric power assisted steering is modeled to show the differences, the similarities and the usage of the above mentioned languages and tools. To be able to classify the tools and languages, five realization levels were developed:

- Physical modeling models
- Constraint models
- Continuous causal models
- Discretized models
- Discretized models with solver and platform implementation

By using these realization levels, models, tools and modeling languages can be classified, and transformations between them can be set up and analyzed.

As a result, a method to describe the simulation behavior of a MATLAB/Simulink model has been developed using SysML activity diagrams as an approach to achieve integrated system models. Another result is an evaluation of the parametric diagrams of SysML for continuous-time modeling, which shows that they do not enable "physical modeling", i.e. modeling the topology of the system and getting the underlying equations out of this topology. By including physical ports and physical connectors to SysML internal block diagrams, this could be solved.

The comparison also shows many similarities between the languages. The results led to a more detailed investigation on conjugate variables, such as force and velocity, and electric current and voltage, and how these are treated in various languages.

The thesis also includes two industrial case studies: one of a twin-screw compressor, and one of a simulation environment for automotive fuel-cell systems. Conclusions are drawn from these models, referring to the realization levels.

Keywords: mechatronics, MATLAB/Simulink, SysML, bond graphs, Modelica, Simscape, simulation, modeling, EAST-ADL2, physical modeling

Contents

Acknowledgements

I would like to express my gratitude to several people that were involved in the work behind this thesis.

- Martin Törngren, who has been the main supervisor for the major part of this thesis.
- Jan-Gunnar Persson, supervisor for the first part of the thesis.
- Sören Andersson, Bengt Eriksson and Hans Johansson for many interesting discussions on modeling and simulation and input to the work.
- Jan Wikander, for reviewing the thesis and providing valuable feedback and Vicki Derbyshire, for proofreading.
- All project colleagues in the NFCCPP, ATESST and ATESST2 projects.
- Co-authors of publications: De-Jiu Chen, Peter Prenninger, Ian Faye, Thomas Hülshorst, Ashley Kells, Ian Harkness, Carsten Schönfelder, Jan-Gunnar Persson, Philippe Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, David Servat, Martin Törngren, Jianlin Shi, and Viktor Ahlsten.
- My room colleagues over the years: Jan Johansson, Jesper Brauer, Anna Hedlund-Åström, De-Jiu Chen, Jianlin Shi, Tahir Naseer, Viktor Ahlsten and Matthias Biehl for many fruitful discussions on various topics.
- All the other colleagues at Machine Design, thank you all for a good time and for helping me out in many ways.

Finally I would like to thank my family for all support, especially my mother Gun and my wife Tania for proofreading, and my son Ivar for providing motivation to finish this thesis.

Part I

Thesis Summary

Chapter 1

Introduction

This thesis is on the subject of mechatronics, and therefore it starts out by describing and defining the term "mechatronics". Compared with other interdisciplinary subjects, such as Thermodynamics or Biomedicine that perhaps do not need such an introduction, this in itself could be seen as a failure for Mechatronics as a subject. But it can also be seen through the perspective that the evolution in information technology has been enormous since the 1960s, when the mechatronics subject was invented. The subject has been redefined many times in academia over the years [33], and there are also textbooks and research that have been relabelled as mechatronics (see e.g. [44] and [45]).

1.1 Mechatronics and related terms

The word *mechatronics* is made up of mechanics and electronics. Information technology could be seen as the third element of the mechatronics subject [93], and sometimes control theory is also referred to as a fourth element (e.g. [73]).

A mechatronic system is typically composed of an *embedded system*, controlling a *physical system,* the *plant*, through *actuators*, and measuring the result by *sensors.* This way it is also a *control system*, which can be analyzed and synthesized using *control engineering,* which can be seen as a *view* of the system. The embedded

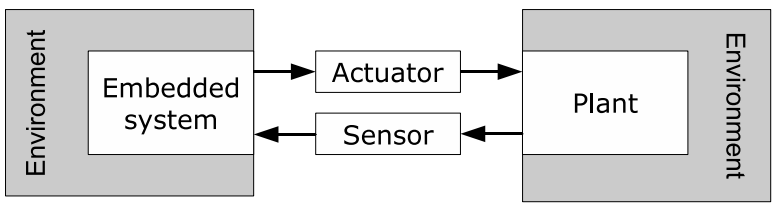

Figure 1.1: A generic mechatronic system.

system consists of one or many *microcontrollers*. The mechatronic system interacts with the *environment*. Besides the *design*, the *product development* of a mechatronic product includes identifying customer needs and production development. When the requirements are derived, the synthesis of a product can also be called *systems engineering*.

A central characteristic of a mechatronic system is the tight copuling between the embedded system and the plant, which leads to that both have to be considered and analyzed concurrently when developing such systems.

For a more strict interpretation of some of these terms, the following definitions are given. They are exemplified in the following section.

Mechatronics: The synergistic combination of mechanical and electrical engineering, computer science, and information technology, which includes control systems as well as numerical methods used to design products with built-in intelligence[1] [35].

Embedded system: A special-purpose computer system designed to perform one or several dedicated functions, often with real-time computing constraints.

Systems engineering: A multidisciplinary approach to develop balanced system solutions in response to diverse stakeholder needs. Includes both management and technical processes [29].

Product development: The set of activities beginning with the perception of a market opportunity and ending in the production, sale and delivery of a product [92].

Control engineering: An engineering discipline that applies control theory to design systems with predictable behaviors. The engineering activities focus on the mathematical modeling of systems of a diverse nature.

Plant: The controlled physical system, being a part of the product (as opposed to the physical environment).

View: A representation of a whole system from the perspective of a related set of concerns [40].

1.1.1 An example technical system: Steering of a car

To exemplify the terms defined in the previous section and to illustrate the issues involved in developing mechatronical products, the steering of a car is investigated.

[1] This is perhaps the most accepted definition today, used in VDI 2206. Wider definitions are those of [5] and [89].

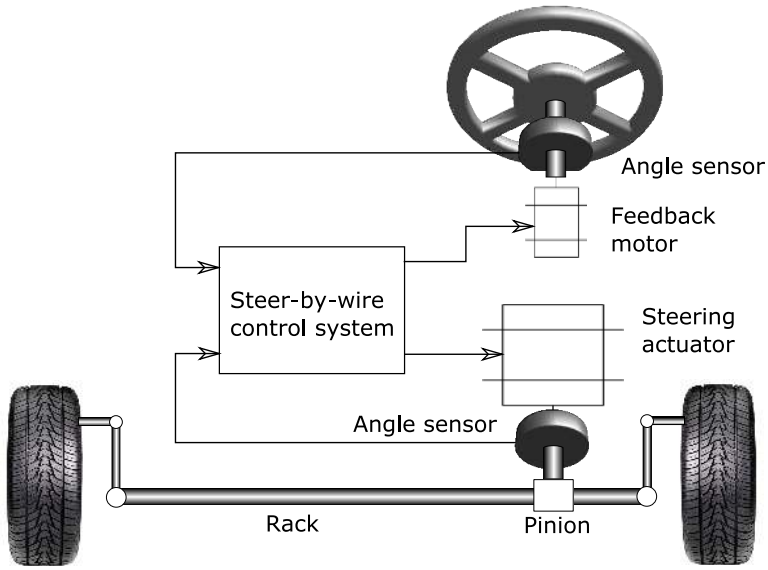

Figure 1.2: A mechatronic system: Steer-by-wire. There is no mechanical connection between the steering wheel and the steering rack.

The *product development* of a car involves market activities such as identifying the customer and their needs, current trends, pricing and financing alternatives. When it comes to selection of steering technology, this can be seen as the realization of the steering function. The steering function can be realized in many different ways, e.g. conventional steering with a hydraulic servo, electric power assisted steering (see Figure 3.1) or steer-by-wire (in Figure 1.2) which all have their different advantages and disadvantages. The generation and evaluation of these alternatives can be seen as *systems engineering.*

When considering the electric power assisted steering, or steer-by-wire alternatives, we are now into *mechatronics,* which then can be regarded as a specialized branch of systems engineering. Mechatronics includes the selection of actuators, gears and microcontrollers, and the development of the software to control the wheels. To get a satisfying response from the steering wheel to the steering of the car in a steer-by-wire system, *control engineering* must be considered. All these aspects correspond to different *views* of the system.

Using the mechatronics perspective, completely new solutions can emerge. An example of this is BMW's active steering system with variable gain, which has an all-mechanical steering linkage, but where the gain is varied between 10:1 to 20:1 (standard is 14:1) using an electric actuator [90].

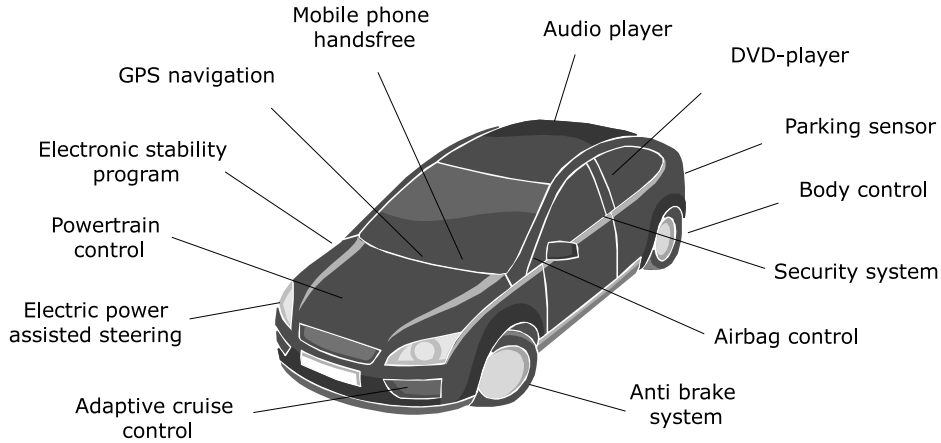

Figure 1.3: Car - today a software product?

1.2 Automotive embedded systems

As seen in Figure 1.3, embedded systems are used extensively in modern vehicles. While the mechanical features in a vehicle are getting more mature and only incrementally developed, it is in the embedded/mechatronic system that innovations take place, and many new features are introduced. Moreover, new propulsion methods, such as electric, hybrid or fuel-cell-driven cars also change the optimal system solutions of components. For example, in an electric vehicle, there may not be an engine-shaft to connect a hydraulic pump to, but instead a larger battery capacity to take advantage of.

One scope of the new electronic functions is to assist the driver in the traction and braking of the car, using functions such as anti-lock braking system (ABS), electronic stability program (ESP), electric power assisted steering (EPAS), "by-wire" solutions (steer-by-wire, gas-by-wire) active suspensions, or engine control. Another application for electronic systems is to control devices in the body of a vehicle such as lights, wipers, doors and windows. Functions that just a few years ago were considered as high-tech luxury are now considered to be standard, such as ESP, seat belt reminders and whiplash protection. Cars without them are even labeled as "dangerous" by The National Society for Road Safety in Sweden [26]. Furthermore, *infotainment* products such as GPS navigation systems, DVD, hands-free phones and music players are also becoming important functions to add value for the customer.

In modern cars, more than 2500 signals, carrying information such as the speed of the vehicle, could be exchanged by more than 70 Electronic Control Units

(ECUs[2]) [61], and these figures are constantly increasing. One ECU can contain many functions, but functions could also be distributed among several ECUs. Moreover, many sensor signals are shared, e.g. the velocity of the car might be used to adapt the steering effort, set the right wiper speed, or turn off the DVD screen for the driver. In the same way, actuators, such as the engine controller, might get inputs from different systems such as the anti-skid system, the adaptive cruise control, external peripherals, not to mention the driver. Somewhere an arbitration of these requested values has to be made, and it is not trivial to decide how this should be carried out.

The bottom line is that the systems have become complex to maintain, develop and test to make sure that they do not have unintended behavior[3]. For this reason, the automotive industry considered an industry-wide standardized software infrastructure as a means to reduce the structural complexity of automotive electronics [36]. As a result, the AUTOSAR development partnership was established in 2003, and it consists today of 9 core members and 59 premium members [6], involving major parts of the automotive industry. In addition to AUTOSAR, there are also several efforts that address model-based development and integration. An example of this is the EAST-ADL2 architectural description language, which is developed in the ATESST and ATESST2 projects [3]. Much of the research behind this thesis has been carried out inside these two projects. Another important initiative is the forthcoming automotive safety standard ISO26262[4], highlighting the need for special care with the increasing use of embedded systems in safety related applications.

1.3 Raising the abstraction level

A well-proved way of dealing with the complexity of engineering systems is to find a suitable abstraction level to work with [51]. Many engineering subjects can themselves be seen as higher abstraction levels of Physics, e.g. Solid mechanics, Tribology and Thermodynamics. At the higher abstraction level, details can be masked or hidden, and a big picture view of the system can be achieved, without getting lost in details.

For embedded control systems, the programming language has changed from machine-code over assembler to C-code, and further to C-code in a real-time operating system, which can be considered as current state-of-practice today. This can be seen as a raise of the abstraction level for the developer. Efforts have been made to move to even more high-level object-oriented languages such as C++ or Java, but it is often difficult to combine with real-time constraints. Another approach is to generate code from models, e.g. Simulink, UML diagrams or some other action language.

[2]Electronic Control Unit, the automotive word for microcontroller.

[3]An example of this is that in 2004 Mercedes-Benz had to remove 600 functions from one of their car models to ensure the function of important electronic parts [74].

[4]Since this standard is not publicly available today, it is not referenced.

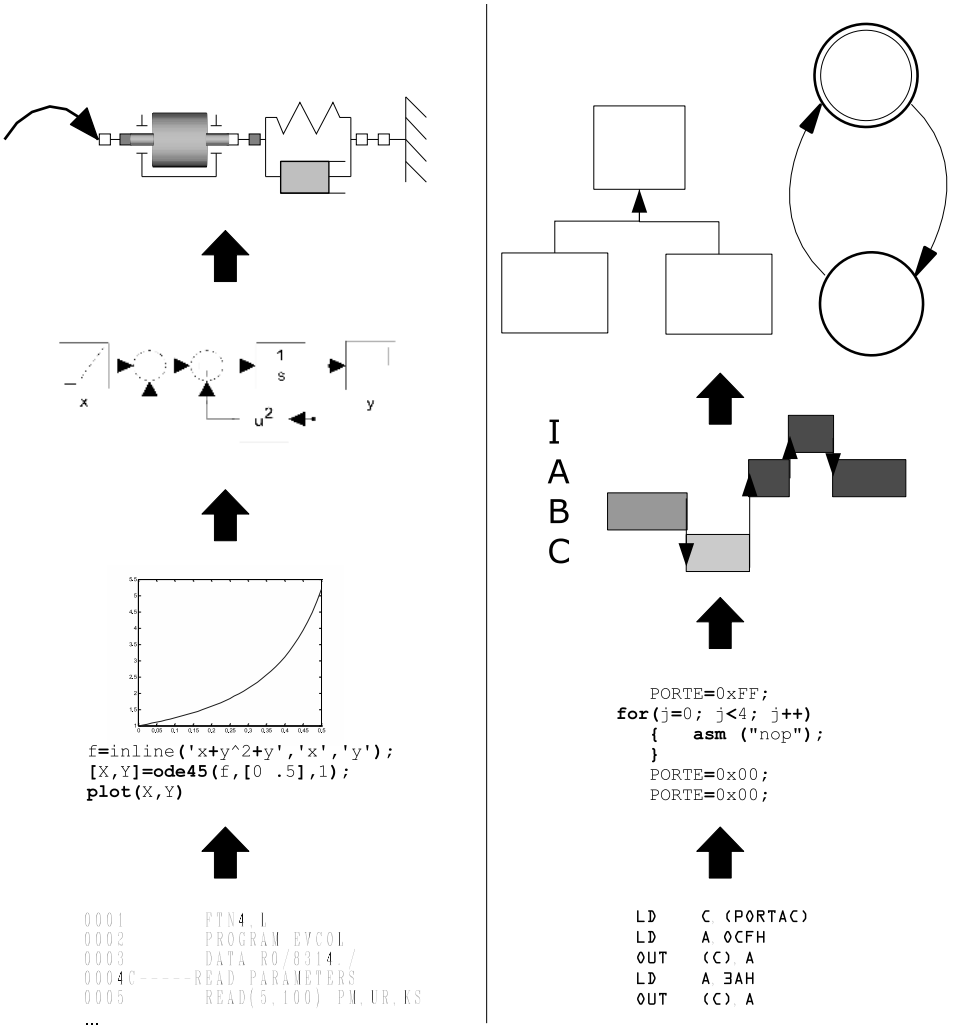

Figure 1.4: Historical raise of the abstraction level. For the modeling of physical systems, evolution has gone from general purpose software (FORTRAN, at the bottom left), to simulation software (MATLAB), to graphical simulation software (Simulink), to physical modeling. On the right hand side, a similar evolution is shown for the embedded system: Machine-code at the bottom (represented by Z80 programming from [65]), C-code program on an AVR microprocessor (note the embedded assembler instruction), RTOS with task model and topmost model-based development represented by a UML class diagram [62] and a finite state-machine [42]. This thesis studies the left side of this figure, and so some extent, how it fits with the right side.

For physical systems, one trend is "virtual engineering", meaning that instead of testing things in the lab, they are tested using simulation software on computers. Simulation software has, just like embedded control systems, also undergone a similar raise of abstraction level, see Figure 1.4. In the early days of computer simulation, the engineer had to program everything from scratch, including the solver, time discretization, error estimation, etc. An example can be found in [70], where a FORTRAN simulation program of a compressor is included. In this code, it is difficult to separate the simulation model from the solver. Using MATLAB, which was released in the mid-1980s [58], a more distinct separation between models and the simulation engine can be made by using the built-in ODE solvers, or the *fsolve* or *fzero* commands. The Simulink extension to MATLAB was introduced in 1990 [58], allowing users to create continuous causal models, which are created graphically, without writing any code. Different acausal modeling languages were merged into the Modelica language, released in 1997 [31], then only supported by the Dynasim tool [22]. Modelica enables physical modeling, i.e. to model the physical components and their interconnections, and let the tool find the equations, and eventually solve them.

For a mechatronic system, the combined raise of abstraction levels at both the embedded system and physical systems enables *model-based development*, where much of the engineering work is made and documented using models. One might claim that all development is model-based, the models varying in level of detail and level of abstraction. In model-based development, the transformations between models at different abstraction levels, such as compiling a program, generating code, or provide a specific view of a system become important.

As seen in Figure 1.4, some of these abstractions are used for both embedded systems and plant models, e.g. programming code, or Simulink models. Simulink is today used to synthesize code for the embedded system, and to test it virtually through Hardware-in-the-Loop and Software-in-the-Loop simulation techniques.

1.4 Scope

As mentioned previously, the overall scope of this work is to find new, higher level abstractions to model mechatronic systems, in particular to incorporate plant modeling in the context of automotive embedded systems. The abstractions have two purposes:

1. Documentation of the design, how it is intended to work, which requirements it fulfills, how it is related to other functions, etc.

2. To provides analysis and synthesis, such as simulation and generation of the final product.

Models of the first kind relate to *model-based development* or *model-based systems engineering*, while models of the second kind relate to *model-based design*. Since the documentation and intended functionality of a system is often captured and

contained in tools for model-based design, especially at the higher abstraction levels, model-based design can be regarded as being a subset of model-based development.

The focus of the thesis is on the "mechanics" part of mechatronics, which apart from pure mechanical systems also can be extended to include other domains like fluid, electrical or thermal systems, and hence named *physical systems*. An evaluation of the abstractions available for the modeling of physical systems has been made, including the integration with abstractions of the embedded system. Mainly, dynamical lumped models of physical systems are investigated. Geometric/kinematic models, like CAD (Computer Aided Design), MBS (Multi-Body Structure), CFD (Computer Fluid Dynamics) or FEM (Finite Element Method) models are not considered here.

1.5 Approach and research questions

The research leading to these results has received funding from three different European Community research projects: NFCCPP (ENK5-CT-2002-00692, 5th Framework Programme), ATESST (2004-026976 in the EC 6th Framework Programme) and ATESST2 (7th Framework Programme under grant agreement no. 224442).

The NFCCPP (Numerical Fuel-Cell Component Performance Prediction) project was about creating a modular simulation environment for fuel-cell vehicles, and the ATESST (Advancing Traffic Efficiency and Safety through Software Technology), and ATESST2 projects are about developing the architectural description language EAST-ADL[5], originally invented in the EAST-EAA project. The research topics have thus been closely related to the description of work in these projects.

The following research questions and sub-questions were formulated during the research work:

- How can one create a component-based simulation environment, using MATLAB/Simulink?
  - How should generic interfaces be designed in different physical domains, especially for fluid systems?
- How can one relate current modeling and simulation environments for physical systems to the EAST-ADL2 modeling language?

Since EAST-ADL2 today reuses concepts from SysML, e.g. flow ports, it would make sense to reformulate this question to; "How to relate current modeling and simulation environments to SysML". The sub-questions that have been investigated are:

- How can one relate Modelica and SysML/UML?
- How can one relate Simulink and SysML/UML?

[5]The language has been renamed to EAST-ADL2 during the ATESST project.

- What are suitable abstraction levels of a physical system, and how can they be related?
  - How can a visual language such as Simulink simplify modeling tasks?
  - How to choose the right tool for a modeling task?
  - How could a universal language be used?
- How can the behavior of continuous-time systems be specified?

These research questions are reviewed in section 6.1 on page 73.

1.6 Overview of the results

- Two simulation models have been developed: one of a twin-screw compressor and the other of a fuel-cell simulation environment.
- A method of protecting the intellectual property (I.P.) of simulation components has been developed.
- A survey of modeling languages for continuous-time systems has been carried out. An example model of an electric power assisted steering system has been implemented in these languages.
- Transformations between SysML and Simulink as well as between SysML and Modelica have been investigated.
- Based on experience from the case studies, methodological concerns when modeling continuous-time systems have been identified and elaborated.

The validity of these results is discussed in section 6.3 on page 75.

1.7 Thesis outline

Following this introductory chapter of the thesis summary, Chapter 2, "Model-based Development of Mechatronic Systems", is presented as a frame of reference. In Chapter 3, the tools and languages investigated are presented, using an example system. The case studies performed during the research work are presented in Chapter 4. Using results from both Chapter 3 and Chapter 4, various methodological concerns are derived for the modeling of continuous-time systems. These concerns are presented in Chapter 5. Chapter 6, "Conclusions and Future Work", summarizes and concludes the thesis summary, and provides suggestions for future work.

After these chapters the following papers are appended:

1.7.1 Appended papers

Paper A: Modelling of Displacement Compressors Using MATLAB/ Simulink Software Sjöstedt, C.-J. *Product Development in Changing Environment*, 2004

Originally presented at the NordDesign conference 2004. This paper is a case study of modeling and simulation of a physical system, giving some insights on the choice of abstraction level. This paper is further discussed in section 4.1 on page 47.

Paper B: Virtual Component Testing for PEM Fuel Cell Systems: An Efficient, High-Quality and Safe Approach for Suppliers and OEM´s Sjöstedt, C.-J., Chen, D.-J., Prenninger, P., Faye, I., Hülshorst, T., Kells, A., Harkness, I. and Schönfelder, C. 3rd European PEFC Forum, electronic proceedings, 2005

As stated previously, much of the research work was carried out in the NFCCPP project. This is the publication from this project, originally presented at the third European PEFC forum in Lucerne 2005. In this thesis, the NFCCPP simulation environment is used as a second case study, and it is further discussed in section 4.2 on page 50

The paper was co-written by all authors; Sjöstedt was the presenting author, mostly involved in the chapters "Standardization of simulation modules & interfaces" and "Model encryption and protection".

Paper C: Developing Dependable Automotive Embedded Systems using the EAST ADL; representing continuous time systems in SysML Sjöstedt, C.-J., Chen, D.-J., Cuenot, P., Frey, P., Johansson, R., Lönn, H., Servat, D. and Törngren, M., *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools,* Berlin, Germany, 2007

This paper was used to start the discussion on how to relate the SysML constructs for continuous systems: parametric diagrams and constraint blocks, with Modelica. The approach used in this paper is presented in section 4.3.1 on page 53.

The paper was co-written by all authors; Sjöstedt was the presenting author, and developed parts related to Modelica and SysML.

Paper D: Mapping Simulink to UML in the Design of Embedded Systems: Investigating Scenarios and Structural and Behavioral Mapping Sjöstedt, C.-J., Shi, J., Törngren, M., Servat, D., Chen, D-J., Ahlsten, V. and Lönn, H., *OMER4 Post-proceedings*, 2008

This paper originates from work done by Shi on transforming Simulink to UML structure. The behavioral mapping was added in this extensively revised version of the paper.

Sjöstedt revised the structural mapping, and developed the behavioral part of the transformation.

1.7.2 Additional publications

This section describes publications that were published during the research work, but for various reasons were not included as a part of this thesis.

The Design of Modular Dynamical Fluid Simulation Systems Sjöstedt, C.-J. and Persson, J.-G., *Proceedings from OST 05 conference,* 2005

Here, equations for a lumped system of dynamical fluid systems are derived, and a technique to reduce simulation time using *virtual mass* is presented. This is simlar to the *tearing* concept, further described in section 2.2.3.

The work presented in the paper and the writing was made by Sjöstedt. Persson provided feedback and supervision.

On the Modular Modelling for Dynamical Simulation with Application to Fluid Systems *Sjöstedt, C.-J.*

Licentiate thesis presented in December 2005, summarizing Paper A and Paper B and the paper above [83].

Automotive Fuel Cell System simulation, component and compressor modelling Persson, J.-G., Chen, D.-J. and Sjöstedt, C.-J., *Schraubenmaschinen, VDI Verlag GmbH*, 2006

This paper was presented at Schraubenmaschinen 2006 [71], and could be seen as a successor to Paper A, together with results on code-protection included in Paper B. This paper is discussed in section 4.1.

The code-protection methods were developed by Chen and Sjöstedt, together with the NFCCPP project. All twin-screw compressor calculations are performed by Sjöstedt, using ideas from Persson.

Experiences from Model supported Configuration Management and Production of Automotive Embedded Software Larses, O., Sjöstedt, C.-J., Törngren, M. and Redell, O., *SAE Word Congress & Exhibition,* 2007

This paper was presented at the SAE World congress 2007 [49]. It describes the SAINT project, which was run as three capstone courses for final-year students in mechatronics. Sjöstedt co-supervised two of these courses. The project is about creating a small scale AUTOSAR-like environment, connected to a commercial Product Data Management database.

The project was initiated by Törngren, Larses and Redell, of which the latter two had left the department when the paper was written. The paper is mainly written by Törngren and Sjöstedt.

Managing Complexity of Automotive Electronics Using the EAST-ADL Cuenot, P., Chen, D., Gérard, S., Lönn, H., Reiser, M.-O., Servat, D., Tavakoli Ko-

lagari, R., Sjöstedt, C.-J., Törngren, M. and Weber, M., *ICECCS, IEEE Computer Society,* 2007

Presented at an IEEE Conference on Engineering Complex Computer Systems 2007 [20]. This is one of many publications describing EAST-ADL2.

The paper was co-written by all authors; Sjöstedt wrote the environment modeling parts.

Chapter 2

Model-based Development of Mechatronic Systems

This chapter includes an overview of design methodologies for mechatronic systems, as well as a section devoted to modeling languages of mechatronic systems that support model-based development. The intention is to find out how modeling languages can support the development process, or possibly improve it by providing new ways of working.

2.1 Design methodology for mechatronic systems

Mechatronics is often considered to be a special subject, requiring special attention to how it is taught [33], and how such products are developed [73]. As said in section 1.1, mechatronics spans over mechanical engineering, electrical engineering, control engineering and computer science. The difference between software and mechanical systems, and the development methodologies and traditions in these domains, is perhaps the most important factor, making the development of mechatronic systems difficult to describe.

2.1.1 Traditional engineering design methodology

An excellent overview of the history of design research and a comparison of different processes can be found in [8]. In short, traditional design methodology roots back to the mid-1960s, when prescriptive models on development processes were introduced. They consist of a number of stages or activities, or a combination of them. The name and number of the stages varies, but [8] identifies three main stages.

- *Problem definition stage*, including getting the requirements
- *Conceptual design stage*, where a solution principle, or concept is generated
- *Detail design*, resulting in a full product description

Representative examples of such design processes are those of Pahl & Beitz[66], Andreassen [60] and Ulrich / Eppinger [92]. It is emphasized to work in a structured way, an thus finding issues and alternative solutions early in the process.

2.1.2 VDI 2206

VDI 2206 [93] is a standard design methodology for mechatronic systems, developed by a committee in VDI, The Association of German Engineers. It is a 118-page document written in both English and German, and it was released in 2004. Compared with other standards, such as ISO9000, VDI 2206 does not contain many prescriptive phrases (e.g. "shall" statements).

The first part of the standard is an introduction to the mechatronics topic, including definitions, and a relatively big part of the standard is dedicated to case studies of mechatronic systems. The methodology chapter starts out with the sentence:

> Both the experiences of industrial practice and the results of empirical design research from recent years have made it clear that there is no "canonizable" optimal form of the design process which the designer can follow in a fixed schedule.

This could possibly be seen as a criticism to the classical design methodologies in section 2.1.1. Instead of providing a complete design process, VDI 2206 defines a procedural model, supported by three elements, described below.

General problem-solving cycle on the micro-level

By dividing the project in smaller subtasks, these tasks can be solved using a general problem-solving cycle. These smaller cycles can be predicted and thus scheduled, and unforeseeable problems can also be assigned to new problem-solving cycles. The problem-solving cycle presented is adopted from systems engineering [21].

V model as a macro-cycle

The V model is adopted from software development, to fit mechatronic systems. The contents of the V model: Requirements, System design, Domain-specific design, System integration, Assurance of properties and Modeling and model analysis are all displayed in Figure 2.1. For complex products, the V model is run through many times, increasing the maturity of the product. The V model shows that system integration - Verification and Validation, are important, and matched against the requirements and system design.

Process modules for recurrent working steps

VDI 2206 points out that there are process modules for recurrent working steps, such as system design, modeling and modeling analysis, domain-specific design,

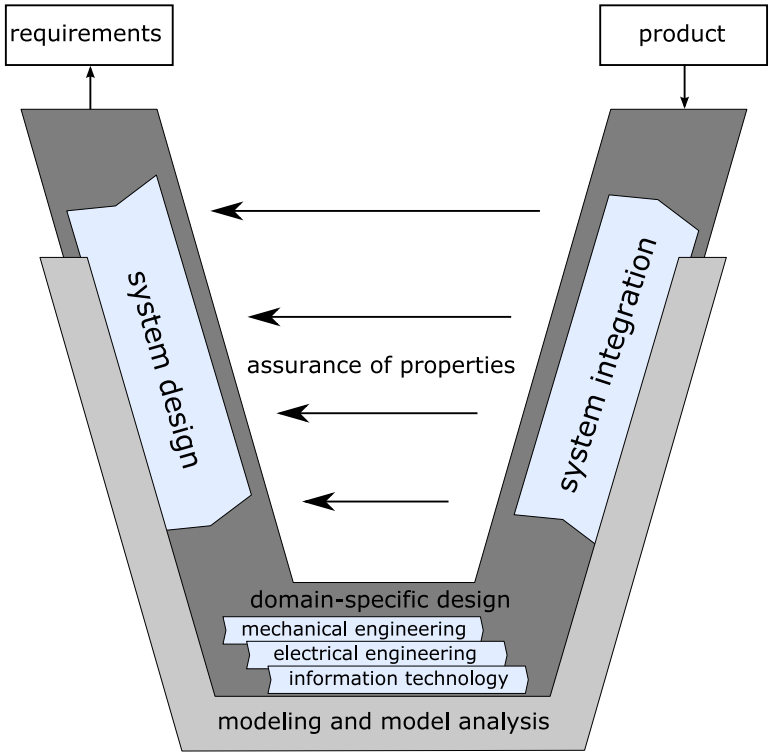

Figure 2.1: The V model [93].

system integration and assurance of properties (verification and validation, e.g. Hardware-in-the-Loop). In the chapter *Model-based system design* of VDI 2206, abstraction levels for models are defined, which are reviewed in section on page 67.

The procedure chapter also contains a section on integrative design of product and production systems. This is an important aspect, not trivial to deal with, since it spans many different competences, from production engineers to software system architects. For software systems the variable cost is zero, and for electronic systems (circuit boards etc.), which are designed in procedural way it is predictable. However, for a complete system, like the electronic system of a car, the production cost is very complex to estimate. It depends on many factors, e.g. how many ECUs are used, and where they are located. There are also other things to consider, like modularity, safety concerns, cabling cost, accessibility etc..

Additional topics

In addition, VDI 2206 also includes a chapter on model-based system design, an overview of common tools, and a section on organization of development teams.

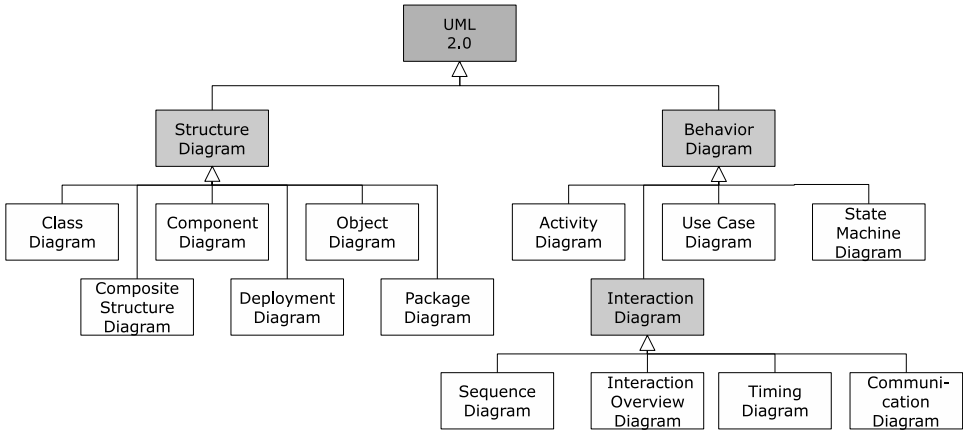

Figure 2.2: The 13 diagrams of UML 2.0, described using a class diagram.

2.2 Modeling languages for mechatronic systems

Since mechatronics is a diverse topic involving many different disciplines, there are many modeling languages that support different views of the system. There are programming languages for embedded systems, including *imperative languages* such as C, C++ or Java, *functional languages* such as Hume and Erlang, *synchronous languages*, e.g. Lustre and Esterel, and *model-based languages* like Simulink and Ptolemy II. See e.g. [15] for a recent overview of such languages. In [91], a survey of different languages for co-design of control systems and their real-time implementation is given, the languages mentioned here are AIDA, Jitterbug, ORCCAD, Ptolemy, Targetlink, Torsche and Truetime.

Languages that aim to describe and document systems, rather than simulate them, include the Unified Modeling Language (UML). UML is a standardized general-purpose modeling language, originating from object oriented software engineering, and it is maintained by the Object Management Group (OMG). Today, UML is used for many purposes, such as business modeling and systems modeling. As seen in Figure 2.2, the language includes six structural diagrams and seven behavior diagrams.

Another means to describe large heterogeneous systems, is by using an architectural description language, such as AADL [2] or EAST-ADL2. SysML (see section 3.6) is a systems engineering language, and is thus also aimed at describing systems spanning different domains.

2.2.1 The model - metamodel concept

One way to define modeling languages in a stricter form is to define a metamodel for the language. This metamodel can in turn be defined using another metamodel,

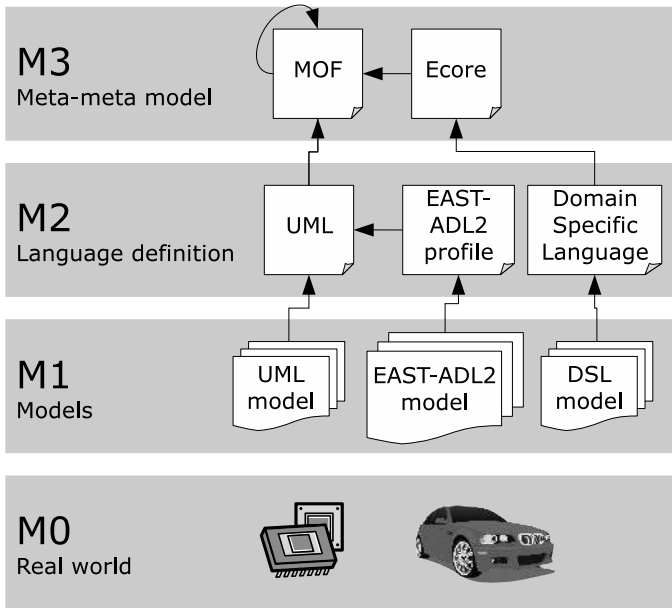

Figure 2.3: The OMG four-layered architecture. Models are classified by languages, which in turn are classified by meta-meta models.

which then is called the meta-metamodel. OMG developed the Meta Object Facility (MOF) standard as a means to define the UML language. In Figure 2.3, the four-layered architecture is displayed. As MOF can be defined using itself, it is the highest layer. In the more recent MOF 2.0 standard, the role of the four-layered architecture is toned down [63]:

> One of the sources of confusion in the OMG suite of standards is the perceived rigidness of a 'Four layered metamodel architecture' which is referred to in various OMG specifications. Note that key modeling concepts are Classifier and Instance or Class and Object, and the ability to navigate from an instance to its metaobject (its classifier). This fundamental concept can be used to handle any number of layers (sometimes referred to as metalevels). ... Note that most systems use a small number of levels (usually less than or equal to four).

Using these concepts, a modeler can create a language suitable for a particular task. This technique is called *Domain-Specific Modeling* [32]. In metamodeling tools, the modeler can create a metamodel, and then the tools generate a language environment where concepts defined in the metamodel can be used. Examples of such tools are GME, MetaEdit+, ATOM3 and GMF [32]. One of the advantages

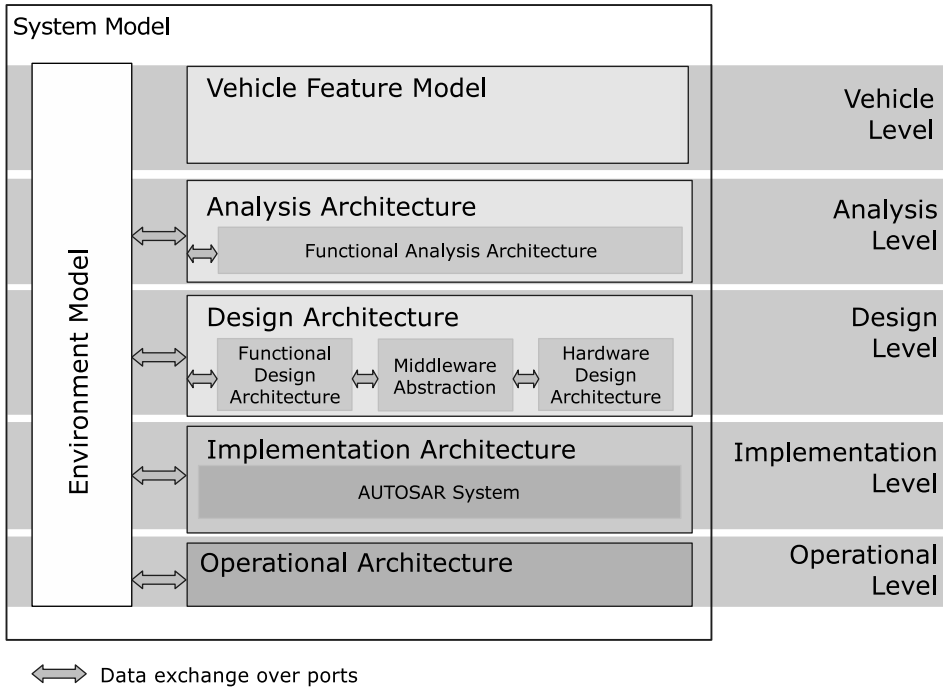

Figure 2.4: The EAST-ADL2 model structure.

of using the metamodeling concept is that it facilitates transformations between languages, which can be defined in a declarative way. The Atlas Transformation Language [4], used in Paper D, is an example of a transformation language that utilizes such technique.

2.2.2 EAST-ADL2

EAST-ADL2 and AUTOSAR both have roots from the EAST-EAA European project. EAST-ADL2 is an Architecture Description Language for handling all engineering information required to sustain the evolution of vehicle electronics.

As shown in Figure 2.3, the EAST-ADL2 *domain model* is defined at the M2 level, in other words it is a metamodel. Basic concepts of UML, such as classes, compositions and associations, are used to define the domain model. EAST-ADL2 also reuses a subset of UML and SysML. Based on the domain model, a UML profile is designed, which uses UML extensions, stereotypes with properties and constraints. This way, EAST-ADL2 can be modeled using a standard UML tool. A reference implementation of EAST-ADL2, including some language specific plugins, is made in the Eclipse-based open-source tool Papyrus [67].

An EAST-ADL2 system model is described using five abstraction levels:

Vehicle level In the *Vehicle Feature Model*, vehicle electronic features such as Brake, Wiper, Collision warning etc. are organized. A variability mechanism supports the definition of rules for inclusion in different vehicles, enabling Product Line Architecting.

Analysis level The *Functional Analysis Architecture* describes functions, *ADL-Functions*, that realize the features. The ADLFunctions are connected to the environment via sensors and actuators, modeled as *Functional Devices*. Interfaces and interaction can be defined and simulated in legacy tools, e.g. Simulink. In fact, this abstraction level is much influenced by modeling in Simulink.

Design level One component of the *Design Architecture* is the *Functional Design Architecture*, where functions defined at the Analysis Level are decomposed and allocated. In the *Middleware Abstraction*, interfaces to a middleware are defined. The Design Architecture also includes a *Hardware Design Architecture*.

Implementation level In the *Implementation Architecture,* reusable code (platform independent) and AUTOSAR compliant software are defined.

Operational Level The *Operational Architecture* is the final binary software deployment.

When analyzing EAST-ADL2 models, the controlled system, i.e. the plant, must also be considered. The function analysis is performed at any of the four lower levels of abstraction, so accordingly these models want to communicate with the *Environment model*. The environment, as defined in this context, consists of both the *plant* and the *environment* as defined in Figure 1.1 on page 3. The environment can therefore also contain other embedded systems. When considering a single car, the plant is the vehicle and the environment consists of e.g. the road and other vehicles. When there are functions where many cars and traffic signs cooperate, this separation is not so clear. This is on-going research in the ATESST2 project, which is focused on cooperative systems. As seen in Figure 2.4, the environment/plant is part of the system model, and that is in short the background of the work presented in section 4.3: *Modeling physical systems with embedded systems*.

2.2.3 Modeling dynamical systems

Although this thesis is not intended to be a textbook of the modeling of dynamical systems, there are a few topics that will be mentioned and defined for further reference within this thesis.

Modeling in the time and frequency domains

The key characteristic of dynamical systems is that they are time-dependent. Because of this, it is interesting to observe the evolution of a dynamical system, over a certain time span. Another way of analyzing dynamical systems is to see how they behave when the frequency of the input is varied. This is called modeling in the frequency domain, and is a common method in control engineering. The tools presented in this thesis work mainly in the time domain.

Causality

A central term when modeling dynamical systems is *causality*, which means the relation between cause and effect. A plant model typically has one or more inputs to actuators and one or more outputs to the sensors. In the steer-by-wire model in Figure on page 5, the inputs to the plant model consist of two voltages, which assert torques on the steering actuator and the feedback motor. The angles from the steering wheel and the steering actuators are the two outputs from the plant model. The plant model consists of many mathematical expressions, to calculate intermediate variables as the inputs propagate to the outputs. Assuming a particular design, this can be modeled in different ways, e.g.:

Voltage → Current → Torque → Torque → Angle
Voltage → Torque → Angle → Torque → Angle

It is up to the modeler how to choose this, but the choice of causality can affect the simulation performance and accuracy of the model. Integral causality is preferred, which means that the causality is chosen so that the solver only needs to integrate.

Ill-formulated problems

By using the causality rules of integral causality, a problem can become *ill-formulated*, when two components with the same "preferred causality" are connected. The typical problem is two adjacent masses connected stiffly to each other, or two capacitances in series in electrical systems. One solution to this is to reformulate the problem by lumping together these masses or capacitances. Another possibility is to use Transmission Line Modelling [47] to cope with this issue.

Algebraic loops

Another concern when modeling dynamical systems is *algebraic loops*. An algebraic loop typically occurs when dynamics are unaccounted for, or when connecting component models to each other. In Simulink, different techniques to solve algebraic loops are shown in Figure 2.5. Using *tearing* [24], algebraic loops can be avoided by inserting additional components, and thus introducing additional dynamics.

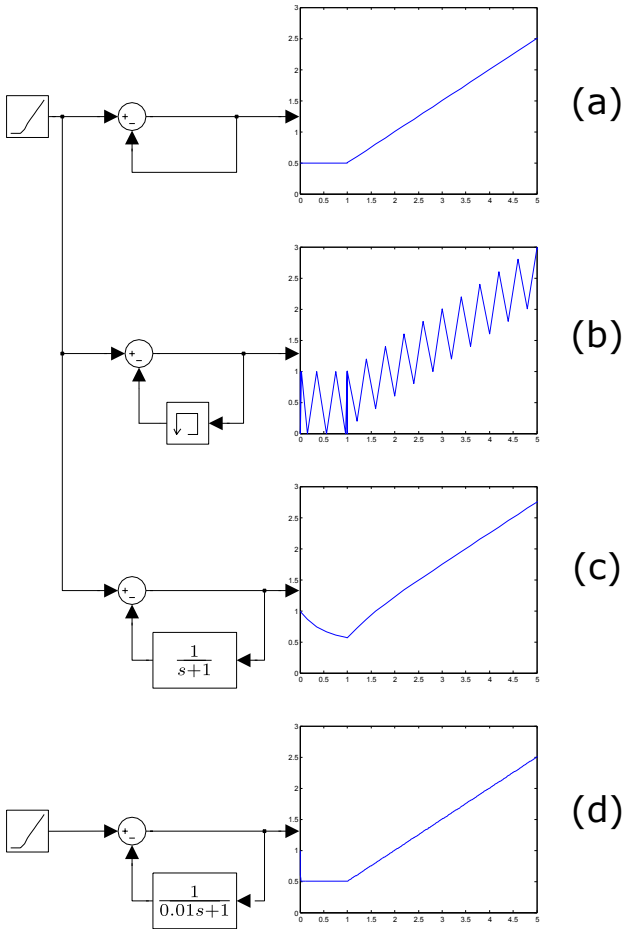

Figure 2.5: Different ways of approaching algebraic loops: In (a) Simulink's algebraic loop solver works well for simple problems like this, but gets slow and in practice unusable for more complex models. In (b) a simplistic way to break the loop by using a memory block, which is easy to implement but gives oscillations with the periodicity of the time step. Next in (c) a transfer function is used, which turns the algebraic equation into a differential equation, which distorts the output slightly; a large time constant is used to show this. Using a smaller time-constant (d) the result is indistinguishable from the real solution, except for the first few time-steps. However, a smaller time-step is needed for this solution, hence it is simulated separately from the others (a-c).

Hybrid systems

Physical systems can be modeled with discrete behavior, such that the system variables change instantaneously for certain state or time values. A classical example is a bouncing ball; another is a gear-box. An on/off switch, or a step input function also makes the system hybrid, so any system controlled by a microcontroller is a hybrid system, although they might be abstracted to a continuous system.

Hybrid systems can not only be described using equations; some event logics, such as "if - then", or "when" causes must be included. This also has implications for the system solver, which can not only be a differential equation solver, but must also take into account this hybrid behavior. The behavior of hybrid models is tightly coupled with the type of solver that is used, and thus the simulation result depends on the solver to a higher extent than for pure differential equations. In Simulink, this mechanism is called zero-crossing detection [88].

For a more detailed approach on how to define continuous systems, see [9], where five different types of continuous systems are defined. In [17], more information on hybrid systems, including a formal definition, can be retrieved.

Chapter 3

Modeling Tools and Languages for Continuous-Time Systems

There are many modeling languages, tools and methods available for modeling mechatronic systems, covering different and overlapping aspects. In this section the focus is on how the language/tool handles continuous-time systems. The purpose of this study is to compare how SysML's constructs for modeling continuous-time systems match with the tools and languages that are available today. The following tools and languages were chosen: Bond Graphs, Ptolemy II, MATLAB/Simulink, MATLAB/Simulink/Simscape (Simscape) and Modelica. These languages represent different ways of describing continuous-time systems at high levels of abstraction.

The evaluation criteria, and related questions are:

- Language scope
  - What is the purpose of the language?
  - What other kinds of models are included in the language?
- Level(s) of abstraction(s)
  - Which models of computations are used?
- Modularity
  - How can components be separated and reused?
- Tool support.
  - Is it possible to run the model and/or to generate C-code?
- Tool transparency

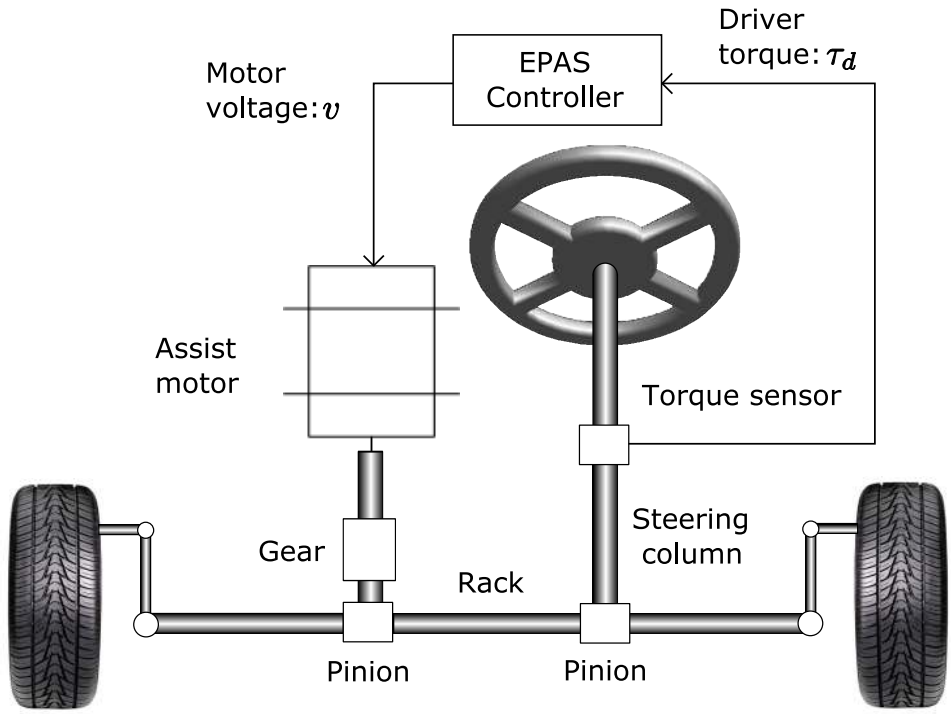

Figure 3.1: A schematic view of double-pinion-type EPAS. The controller function controls the motor voltage, getting information from a torque sensor mounted on the steering column.

– Is the source code open?

– Are the simulation results predictable; does the user know what is happening inside the simulation engine?

These questions are reviewed in the last section of this chapter on page 44. A similar study can be found in [17], where different languages and tools are also evaluated with a focus on hybrid systems, an aspect that not is covered here. In that publication the simulation tools Simulink and Stateflow, Modelica, HyVisual[1], Scicos, Shift and Charon, and the formal verification tools HyTech, PHAVer, HSOLVER, CheckMate, d/dt and Hysdel are evaluated.

About the case study

To show the differences and similarities between the modeling languages, and to serve as a benchmark, an example was modeled using the languages. The following criteria were set when choosing the example model:

- An automotive mechatronic system, with an extensive plant model to be controlled, preferably in multiple physical domains.
- An authentic modeling scenario with accurate parameters.
- A simple enough model for the reader to grasp, but still complex enough to be non-trivial, requiring thorough analysis.

The choice fell on the modeling of power assisted steering. This is a typical example of a system which is about to be converted from traditional mechanical and hydraulic solutions, *hydraulic power assisted steering (HYPAS)*, to a mechatronic solution: *electric power assisted steering (EPAS)*. Typical advantages of using EPAS (compared with HYPAS) include [16]:

- Better fuel economy, since power is taken from the engine on demand, and not continuously from an engine-driven pump.
- Savings in development time, since steering characteristics can be tuned in-vehicle, through software.
- Reduced number of parts for the manufacturer, since the EPAS system can be made to automatically select its software configuration and calibration to match different vehicle variants.
- Reduced system weight and volume.
- Improved functionality, e.g. speed sensitivity, yaw damping, active return and optional steering "feel" settings.
- Reduced environmental impact, since no hydraulic fluid is used, fuel consumption is reduced, and recyclability is increased.

There are different configurations of the EPAS: in this example, taken from [68], a double-pinion configuration where the assist motor is placed beside the steering column is used. This configuration is used for heavy vehicles; in lighter vehicles, a single pinion is used, and the assist motor is packaged on the steering column, steering rack or in the pinion.

When modeling physical systems, it is crucial to find the right level of detail. Too detailed models will result in a *modeling swamp* [48], where the modeling work seems like an endless process, giving complex models that are difficult to analyze.

[1] HyVisual is a customized version of Ptolemy II for hybrid systems modeling [13].

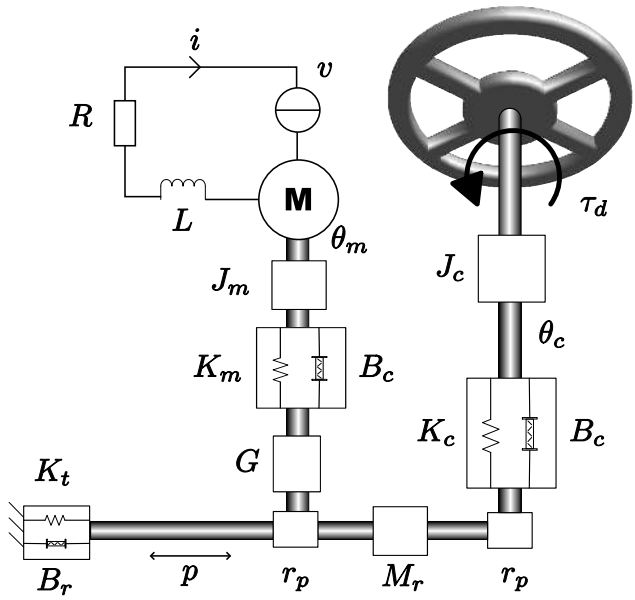

Figure 3.2: Schematic lumped model of the plant model of the EPAS system in Figure 3.1.

On the other hand, too simplistic models will not capture the actual behavior, and will not produce any solutions to the problem. It is up to the engineer to decide what assumptions to make in a particular problem. In this example, these assumptions are made by the authors in [68], leaving us with the simplified lumped model in Figure 3.2. Equations 3.1, 3.2, 3.3 and 3.4 describe the system. These equations are derived using Lagrange's method. In the following sections different tools and methods will be investigated, where some start out with these equations, while others start out with the simplified system in Figure 3.2.

$$J_c\ddot{\theta}_c + B_c\dot{\theta}_c - K_c\left(\theta_c - \frac{p}{r_p}\right) = \tau_d \tag{3.1}$$

$$J_m\ddot{\theta}_m + B_m\dot{\theta}_m - K_m\left(\theta_m - \frac{pG}{r_p}\right) = ki \tag{3.2}$$

$$M_r\ddot{p} + B_r\dot{p} + K_t p = \frac{K_c}{r_p}\left(\theta_c - \frac{p}{r_p}\right) + \frac{K_m G}{r_p}\left(\theta_m - \frac{pG}{r_p}\right) \tag{3.3}$$

$$L\dot{i} + Ri + k\dot{\theta_m} = v \tag{3.4}$$

3.1 Bond graphs

Bond graphs were devised by Professor H. Paynter at Massachusetts Institute of Technology (MIT) in 1959. His former PhD-students Karnopp, Margolis and Rosenberg, now all professors at different institutions, and authors of [45], have continued to develop the method, together with research communities [11].

Bond graphs can be regarded as a graphical method to describe the energy flow in a lumped description of a physical system. One of the key features of bond graphs is the systematic approach. Different physical domains are described using the same building blocks, such as *effort source*, *flow source*, *capacitor*, *resistor*, *inertia*, *transformer*, *1-junction*, *0-junction*, and *gyrator*. There are predefined methods to convert physical models into bond graphs; to convert from the lumped representation in Figure 3.2, the following method was used for the mechanical part [50]:

- For each distinct velocity establish a 1-junction
- Insert 1-port junctions for differences in velocities and use 0-junctions to construct these differences
- Insert inertia (I) elements in 1-junctions, associated with masses
- Insert capacitance (C) and resistor (R) elements
- Make simplifications (junctions with two connections can be removed; two adjacent junctions can be lumped together)

There are similar methods for electrical systems, which were used for the electrical part of the EPAS system. Causalities were introduced and the end result is shown in Figure 3.3. Using the information provided by this bond graph, many actions can be performed:

- The derivation of system equations from a bond graph is systematic and can thus be algorithmized. In other words, equations 3.1 to 3.4 can be derived from Figure 3.3. In contrast with Lagrange's method, which only applies to mechanical systems, this can be done for both the mechanical and the electrical domain.
- A bond graph can also systematically be converted into a signal flow representation [45], such as the one in Figure 3.4.
- In [50], an automated transformation from bond graphs to a sorted list of equations is shown. Using this sorted list of equations, a solver program can be programmed using e.g. C-code.
- There are also tools for direct simulation of bond graphs, e.g. 20-sim [1], See [38] for a complete list of available software.

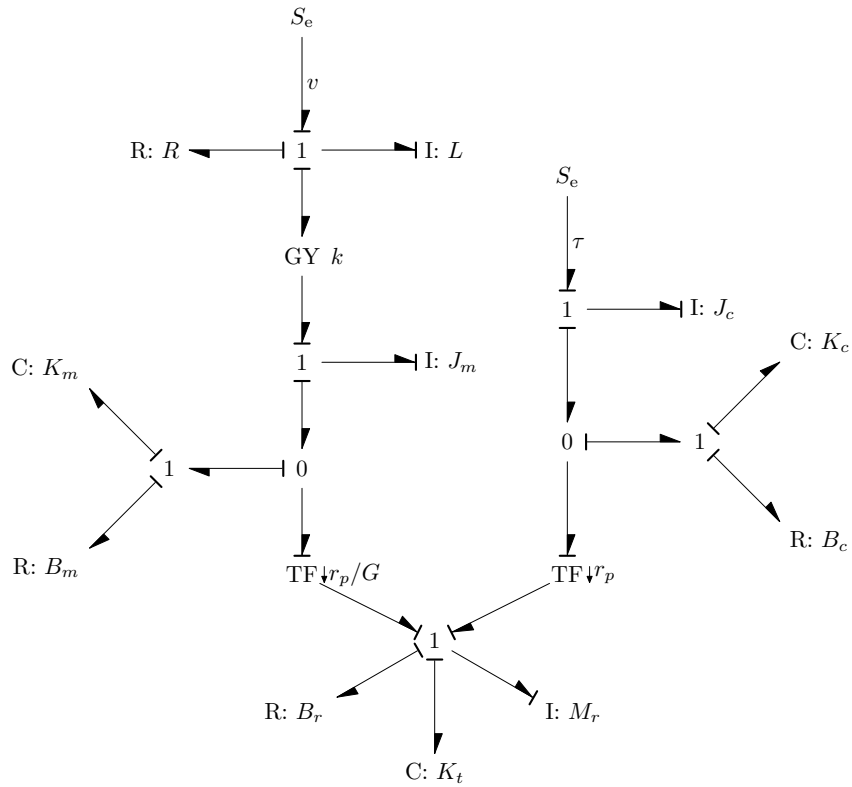

Figure 3.3: Bond graph representation of the EPAS plant.

3.2 Ptolemy II

The Ptolemy project labels themselves as being an "informal group of researchers at University of California, Berkeley" [13]. The project studies heterogeneous modeling, simulation and design of concurrent systems, especially embedded systems. Ptolemy II, developed since 1996, is the third generation of design software to emerge from this group. Executable models are constructed under a *model of computation* (MoC), which governs the interaction of components in the model. The MoC determines when actors perform internal computation, update their internal state, and perform external communication. The MoC also defines the nature of communication between components. Most MoC:s in Ptolemy II support *actor-oriented design*. An *actor* is in principle a block, which executes and communicates with other actors, using ports, interfaces and parameters. This forms the *abstract syntax* of actor-oriented design, which is the structure of the model. A set of rules that govern the interaction of components is called the *semantics* of the MoC. The

semantics is largely orthogonal to the syntax, and is determined by the MoC. The MoC is executed by a *director*. The Ptolemy group defines Simulink and LabVIEW as also being actor-oriented programs [13].

3.2.1 The Ptolemy II continuous-time domain

The Ptolemy II Domain documentation [14] refers to [39], defining two ways to specify a continuous-time system: using the conservation-law model or the signal-flow model. Using the terms defined later in this thesis, this would correspond to constraint models and causal models respectively. Ptolemy II uses the signal-flow model as interaction semantics, and specifies four major reasons for this:

1. The signal-flow model is more abstract
2. The signal-flow model is more flexible and extensible
3. The signal-flow model is consistent with other models of computation in Ptolemy II
4. The signal-flow model is compatible with the conservation law model

These reasons are reviewed in section 5.4. In short, modeling in the continuous-time domain with Ptolemy II is very similar to modeling with Simulink. The model in Figure 3.4 could also be modeled using Simulink.

Ptolemy II is "first and foremost a laboratory for experimenting with design techniques" [13], and as such a demonstration tool, it is very competent. The simulation engine, written in Java, is reasonably fast, although it is slower than Simulink.

3.3 MATLAB/Simulink

MATLAB is marketed as "the language for technical computing" [88], and it is definitely an important tool/language today, together with Simulink ("Simulation and model-based design") and the many toolboxes.

The Simulink extension to MATLAB was introduced in 1990 [58], allowing users to create continuous causal models graphically, without writing any code. Today, Simulink has grown in many directions: adding more blocks, built-in algebraic loop solving, and a multitude of toolboxes for e.g. signal processing, rapid prototyping and physical modeling (Simscape, presented in section 3.5). These additions have made it more powerful, but also more difficult to overview, and to understand and define the simulation semantics.

As said in 3.2.1, a Simulink model of the EPAS system will be very similar to the Ptolemy model in Figure 3.4, so instead of just drawing this diagram again[2], another aspect is presented here, namely how this model could be restructured. To

[2]In fact, it is shown in Figure 4.8 on page 59.

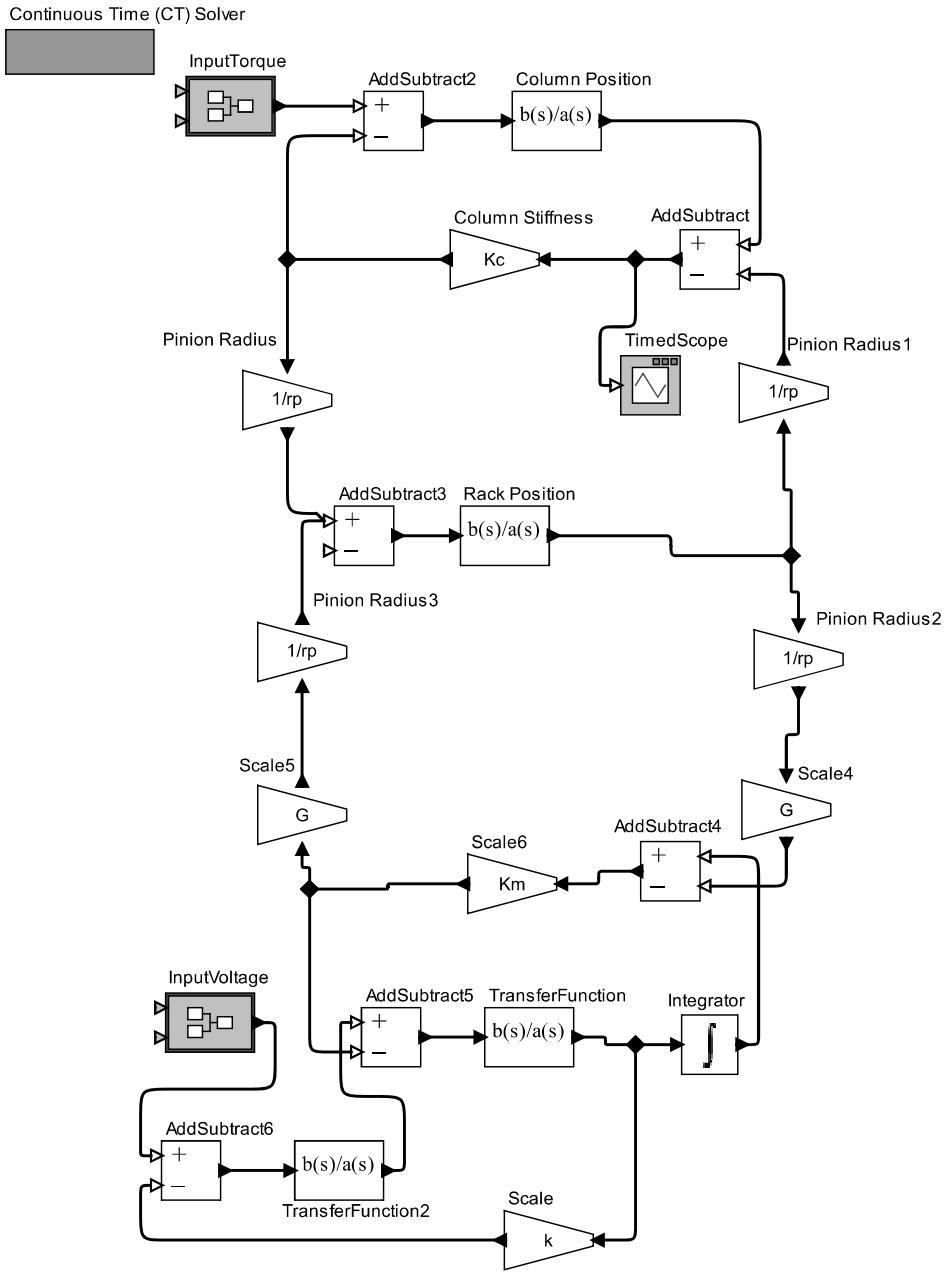

Figure 3.4: Ptolemy II model of the EPAS, as modeled in [68]. The model is derived from equation 3.1, 3.2, 3.3 and 3.4.

organize large Simulink models, they can be divided into subsystems. In *virtual* subsystems, this division is only graphical, and does not imply any simulation semantics.

A possible way of dividing the EPAS into subsystems is shown in Figure 3.5. Notable is that the two *Pinion* subsystems, representing two instances of the same physical object, here have different inputs and outputs. This could be a causality issue (as described in section 2.2.3 on page 22), but is to some extent also a subjective choice by the modeler. The definition of interfaces between components in a modular and convenient way is a typical caveat when modeling physical systems in Simulink. In Paper B, we try to bring order to this, it is also described in short in section 4.2.

3.4 Modelica

Modelica is a modeling language that allows specification of mathematical models for the purpose of computer simulation of dynamical systems. Modelica is a "free object-oriented modeling language with a textual definition to describe physical systems in a convenient way by differential, algebraic and discrete equations" [56]. One could say that Modelica consists of three parts, which are interrelated: The language, the standard library and the Modelica tools. By being a standard language, it can be seen as an enabling technology to build a library of reusable components. As the Modelica language would not be very usable if there were no tools to execute and analyze the models, the language design is also influenced on how the models are solved in tools.

The Modelica language One key feature of the Modelica language is that it is based on equations, instead of assignment statements as in conventional programming languages. Another feature is that it is object-oriented, which facilitates reuse of components and evolution of models. It also has a strong software component definition, with constructs for creating and connecting components [30].

The Modelica library The free Modelica Standard Library contains 777 models and 549 functions for mechanical (1D and 3D) components, electrical components, heat transfer, fluid flow and more [57]. More importantly, the Modelica Standard Library defines interfaces for these domains, which enables *physical modeling*. Using the Modelica language *connect* statement, simulation components can be connected like the real components would be connected physically, which makes it intuitive to model these systems.

Modelica tools The people behind the company Dynasim were (and still are) active in developing the Modelica standards, and for many years their Dymola tool was the only tool supporting the Modelica language [22]. The Modelica language and the Dymola tool could then easily be mixed up.

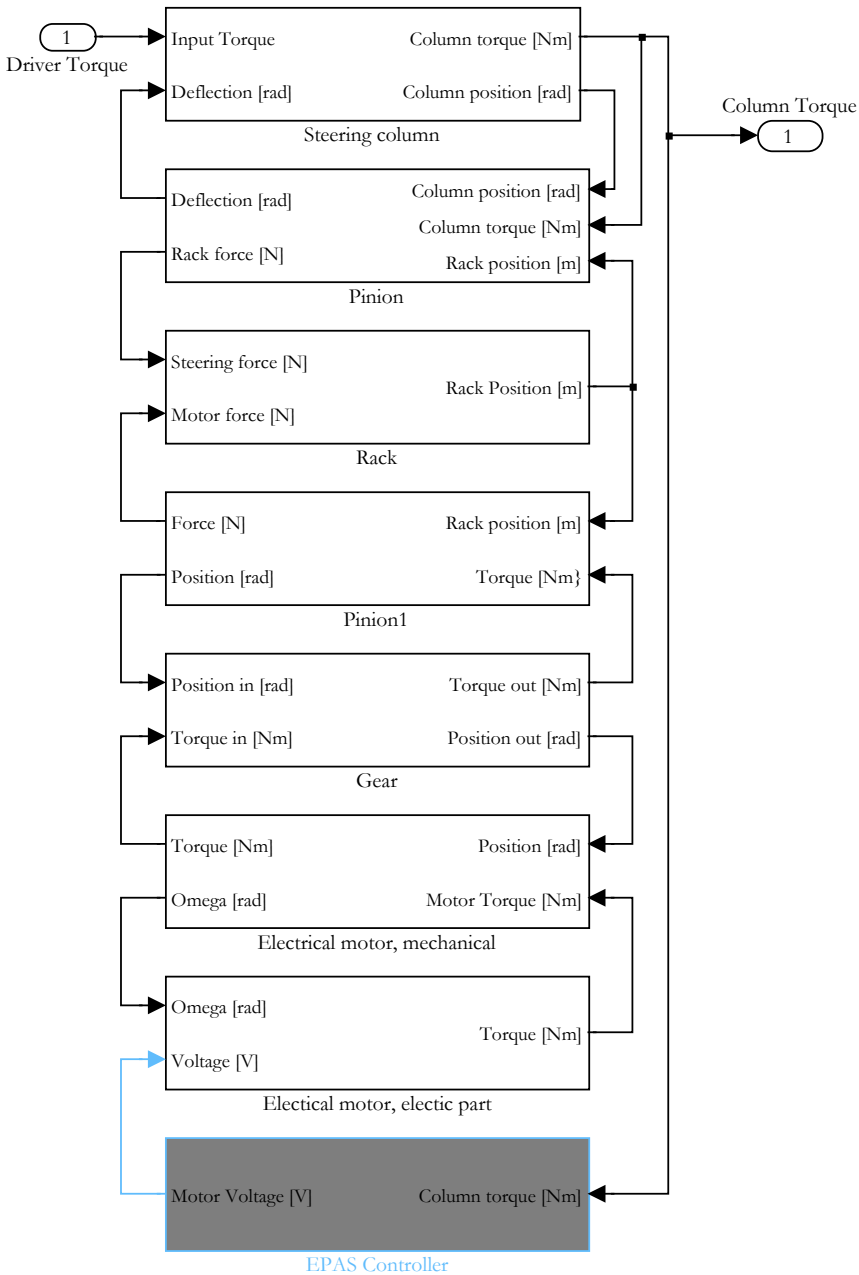

Figure 3.5: Open-loop EPAS system model [68], modeled in Simulink. Compare with Figure 3.4 to see the underlying blocks.

```
model Inductor "Ideal linear electrical inductor"
extends Interfaces.OnePort;
parameter SI.Inductance L=1 "Inductance";
/*The linear inductor connects the branch voltage v with the
branch current i by  v = L*di/dt.
The Inductance L is allowed to be positive, zero, or negative.*/
equation
  L*der(i) = v;
end Inductor;
```

Figure 3.6: A model of an inductor, from the Modelica standard library [57].

Today, there are five commercial and three open-source Modelica environments available [56]. Recently, Maplesoft also introduced the MapleSim toolbox, which is related to Modelica in the sense that some component models are based on the Modelica standard library, and that models can be exported to Modelica. Maplesoft promises support for third-party Modelica libraries "in the near future".

3.4.1 About the model

The model is shown in Figure 3.7. Compared with the lumped representation of the system (Figure 3.2 on page 28), the transformation into a Modelica representation is rather straight-forward.

3.5 Simscape

Simscape is a toolbox for physical modeling developed by the MathWorks for Simulink, and it has been available since version R2007A of the MATLAB suite [88]. It includes a *foundation library*, which contains basic components for electrical, hydraulic, mechanical and thermal systems. There are also more specialized toolboxes for physical modeling (such as SimDriveline, SimHydraulics, SimElectronics and SimMechanics) that now are considered as parts of the Simscape product family (albeit that some of them had been around before Simscape).

In R2008b, a major upgrade of Simscape was made, introducing the *Simscape language* which allows the user to create their own physical models, and even new physical domains, with new conserving ports. The language is based on MATLAB syntax, but it is possible to define acausal equations. Simscape distinguishes between the variables as:

Through — Variables that are measured with a gauge connected in series to an element.

Across — Variables that are measured with a gauge connected in parallel to an element.

Examples of these variables are conveyed in Table 3.1.

Technology	Across Variables	Through Variables
Mechanical	Position Linear velocity Linear acceleration Angle Angular velocity Angular acceleration	Force Torque
Hydraulic	Pressure	Flow rate Volume
Electrical	Voltage Flux	Current Charge
Thermal	Temperature	Heat flow Enthalpy Entropy

Table 3.1: Examples of Across and Through variables in different domains, from the Simscape documentation.

Port Type	Across Variable	Through Variable
Electrical	Voltage	Current
Hydraulic	Pressure	Flow Rate
Thermal	Temperature	Heat Flow
Mechanical translational	Translational velocity	Force
Mechanical rotational	Angular velocity	Torque

Table 3.2: The Physical Conserving ports used in Simscape Foundation library blocks, and the corresponding Across and Through variables. These are a subset of those in Table 3.1. Table 5.1 compares how such conjugate variables are represented in other languages and tools.

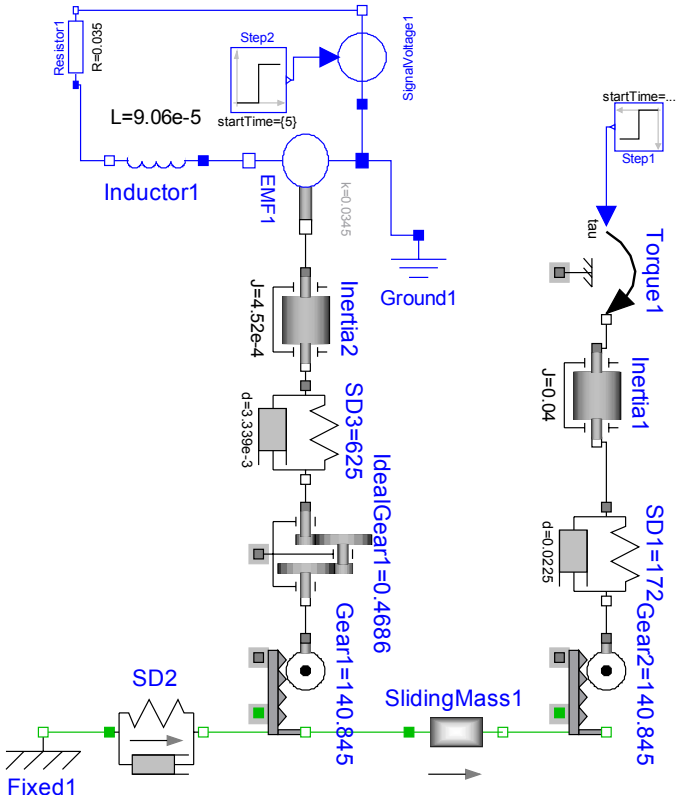

Figure 3.7: Modelica model of the EPAS system, modeled in Dymola [22].

3.5.1 About the model

The Simscape model in Figure 3.9 is somewhat similar to the Modelica model (Figure 3.7) and the lumped model if Figure 3.2 on page 28. There are some subtle differences: one is that the inertia and mass components only have one port in Simscape. There are also problems when trying to simulate a system with unconnected components. Also, to view a signal, a sensor has to be modeled, and then the sensor needs to be converted to a standard Simulink signal to be monitored using e.g. a *scope* block.

3.6 SysML

SysML is a rather new modeling language, it origins from a strategic decision by the International Council on Systems Engineering's (INCOSE) Model Driven Design workgroup in 2001. This led to a request for proposal of a UML for Systems

```
component ideal_inductor
% Ideal Inductor
  nodes
    p = foundation.electrical.electrical; % +:top
    n = foundation.electrical.electrical; % -:bottom
  end

  parameters
    L = { 1, 'H' };   % Inductance
    V0 = { 0, 'V' };  % Initial voltage
  end

  variables
    i = { 0, 'A' }; % Current through variable
    v = { 0, 'V' }; % Voltage across variable
  end

  function setup
    if L <= { 0, 'H'}
        error( 'Inductance must be greater than zero' )
    end
      through( i, p.i, n.i ); % Through variable i from node p to node n
      across( v, p.v, n.v );  % Across variable v from p to n
      i = I0;                 % i(t=0) == I0
  end

  equations
    v == L*i.der;     % Equation
  end
end
```

Figure 3.8: Example use of the Simscape language. Compare with the Modelica component Figure 3.6.

Engineering in 2003, and the first open source SysML specification drafts were distributed in 2004. In September 2007 OMG SysML 1.0 was listed as an "Available Specification" [3] [29].

As a UML for Systems Engineering profile, some software-specific diagrams have been removed. One of the reasons is that the language should be easier to learn. The reason why it is even based on UML is that user experience and tools can be (re-)used [29]. Two new diagrams are introduced: The *requirements diagram* and the *parametric diagram*. Furthermore, SysML uses the *block diagram* and *internal block diagram*, which are slightly modified versions of the UML *class diagram* and *composite structure diagram*, respectively. The motive for SysML working with blocks instead of classes is that the latter are more software-specific, while blocks could represent different things such as hardware, data, people or facilities.

Both Paper C and Paper D include results from our investigations on SysML; in this section we try to use the SysML diagrams in a conventional way, directly adopted from the specification.

[3] In fact, there are two SysML brands: OMG SysML™, trademarked and maintained by the Object Management Group, which is derived from open source SysML, and consequently also includes an open source license for distribution and use [87].

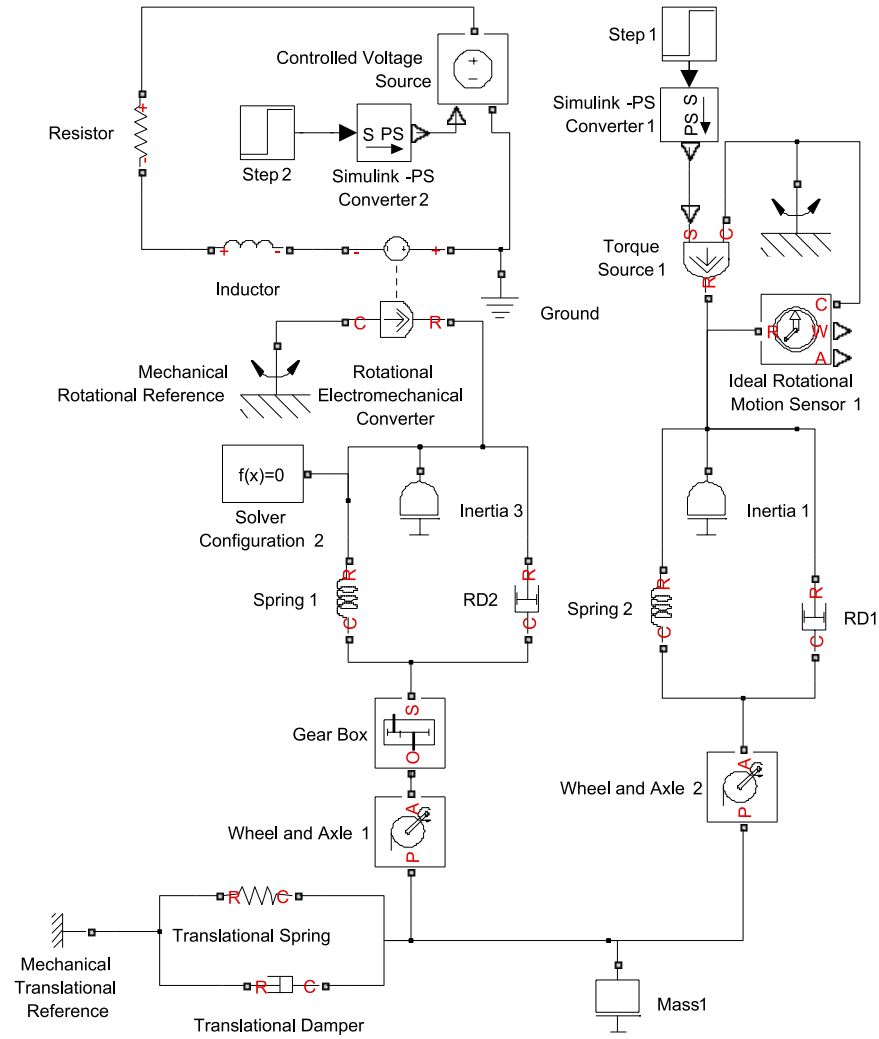

Figure 3.9: Simscape model of the EPAS system. Regular Simulink signals must be converted into physical signals using the Simulink-PS Converter blocks.

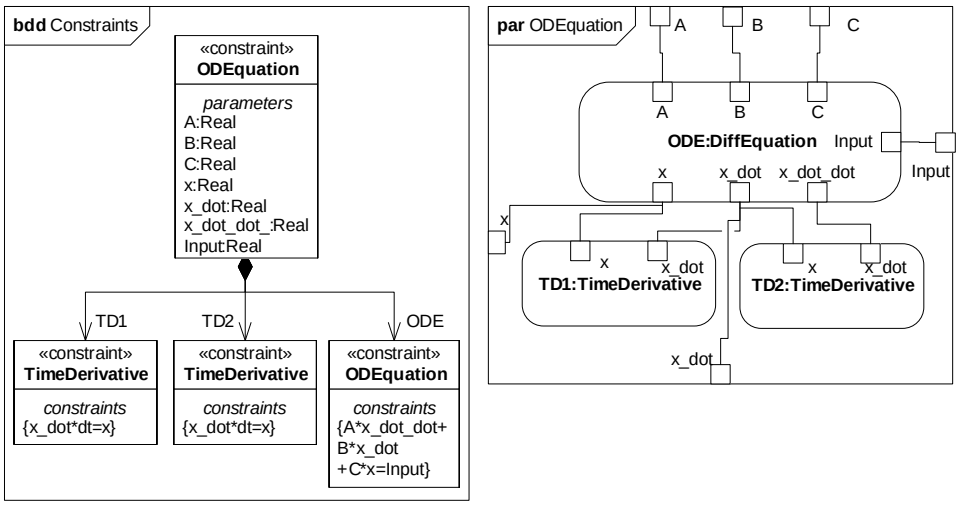

Figure 3.10: Modeling an Ordinary Differential Equation (ODE) in SysML, using constraint blocks and a parametric digram. This diagram is used in Figure 3.11.

3.6.1 The parametric diagram

The parametric diagrams' conceptual foundation is the composable objects representation (COBs), developed at Georgia Institute of Technology [69], the academic partner of the SysML team [62]. COBs provide five basic views of a system: Shape Schematic, Relations, Constraint Schematic, Lexical COB structure and Subsystem, of which Relations, Constraint Schematic and Subsystems have equivalent representation in SysML.

The parametric diagram describes constraints between variables, like equations, and how they are related to each other. The SysML specification gives an example of Newton's equation ($F = ma$), which can be modeled in continuous time. The constraints are acausal, and by combining many modular subsystems, acausal relationships for a large system can be achieved. In addition, state machines can tell which equations to use in the parametric diagram, and in this way describe hybrid systems.

Modeling the EPAS equations using parametric diagrams

Section 4.3.1 describes how we tried to map Modelica to SysML using parametric diagrams. This could possibly be seen as an unintended way of using those diagrams, so an attempt to evaluate SysML's native way of modeling using the parametric diagram is presented here. The physical layout of the components is then ignored, and only the constraints are modeled. In the SysML specification [62] it says "A constraint block is a block that packages the statement of a constraint

so it may be applied in a reusable way to constrain properties of other blocks". Equation 3.1, 3.2 and 3.3 are all Ordinary Differential Equations (ODE:s), with a right-hand side input function. These were then modeled in this reusable way (Figure 3.10 shows how it was made). The input functions, i.e. the right-hand side of Equation 3.1-3.3, were modeled as constraints *ColumnInput*, *MotorInput* and *RackInput* respectively. Equation 3.4 was modeled as a constraint *CurrentEquation*, Then a parametric diagram was made, which connects all these constraints, see Figure 3.11. Some conclusions from this exercise can be drawn:

- Although this is a fairly simple problem, the parametric diagram has too many lines to be readable. Besides the parametric diagram there is also a block definition diagram, defining the constraints *ColumnInput*, *MotorInput*, *RackInput* and *CurrentEquation*, like in Figure 3.10.
- All constraints need to be defined in a reusable way, due to the UML class-instance mechanisms. Here, the input functions (*ColumnInput*, *MotorInput* and *MotorInput*) origin from the particular configuration of this EPAS system, so they are not very reusable, making this rather inconvenient.
- Another way of modeling these equations would be to model each equation 3.1 to 3.4 as a constraint. The reusability would then decrease even further.
- Another possibility is to introduce hierarchies, e.g. by introducing a new constraint that represents one *ODEquation* and an input function. However, in order to describe this new constraint, another three parametric diagrams and three new constraint blocks must be modeled.

3.6.2 Using continuous activity diagrams

UML 2.0 was a major upgrade from previous versions; in literature it is referred to as UML 2 (or UML2) to distinguish it from previous versions. The activity diagram is one of the diagrams that were redefined in UML 2, in order to support flow modeling [9]. The activity diagram has been developed with the Enhanced Functional Flow Block Diagram (EFFBD), a widely-used systems engineering diagram [62], in mind. As a result, EFFBDs can now be described using SysML extension mechanism[4] by stereotyped activities; this is described in detail in the SysML specification [62]. The SysML activity diagram is based on the UML 2 activity diagram, adding a few extensions. Most notable from our point of view is the possibility of modeling continuous systems.

Activities describe behavior that specifies transformation of inputs to outputs. The building blocks of activities, describing how they execute, are *actions*. An action has *token semantics*, i.e. it accepts inputs and produces outputs (tokens) on their pins. Tokens can consist of data information, but also physical items

[4]The EFFBD extension does not address replication, resources, or kill branches.

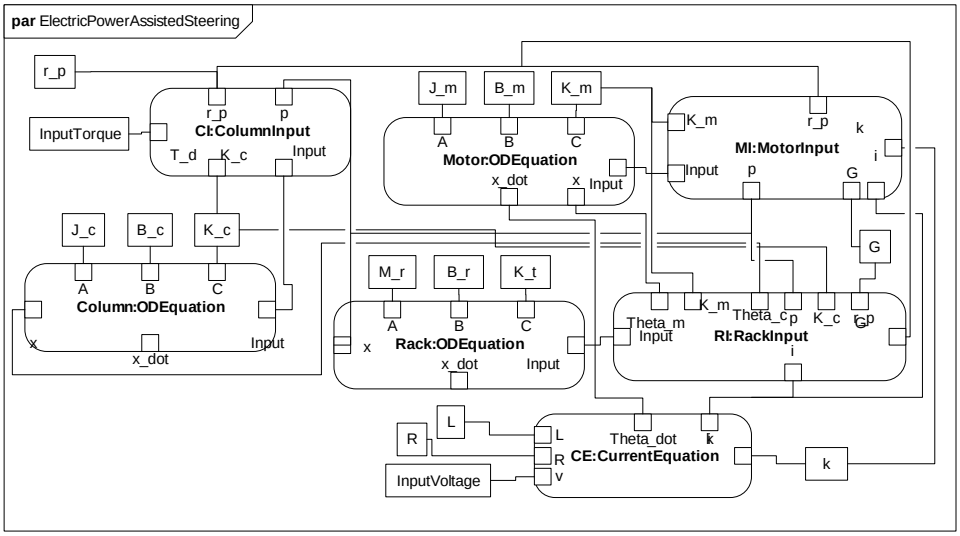

Figure 3.11: The EPAS system, modeled as a native SysML parametric diagram. This parametric diagrams is about to choke on complexity, if more components are added, an hierarchical decomposition would be necessary.

like gasoline. By setting the time between tokens to zero, continuous flow can be specified [29].

In both [10] and the appendix of [62], the numerical solution to the equation $\dot{x}(t) = -2x(t) + u(t)$ is modeled using activity diagrams. In Figure 3.12, the EPAS equations 3.1 on page 28 to 3.4 on page 28 are modeled using that technique. The diagram is very similar to the Simulink or Ptolemy II diagrams (Figure 3.4 on page 32), however the semantics is not coherent with how the Simulink simulation engine works, for example algebraic loops and zero-crossing detection (both explained in section 2.2.3 on page 21) are not taken into account. In order to address this, another mapping of Simulink models to SysML was developed, which is presented in Paper D.

3.7 Other continuous-time systems simulators

Today, there are many languages and tools available in which the EPAS model, or parts of it, could be implemented. For example, MSC ADAMS [59] or CarSim [54] could be used to model the mechanical parts of the system at a higher level of detail. Some tools are not mentioned here, and some are mentioned in short in the following section:

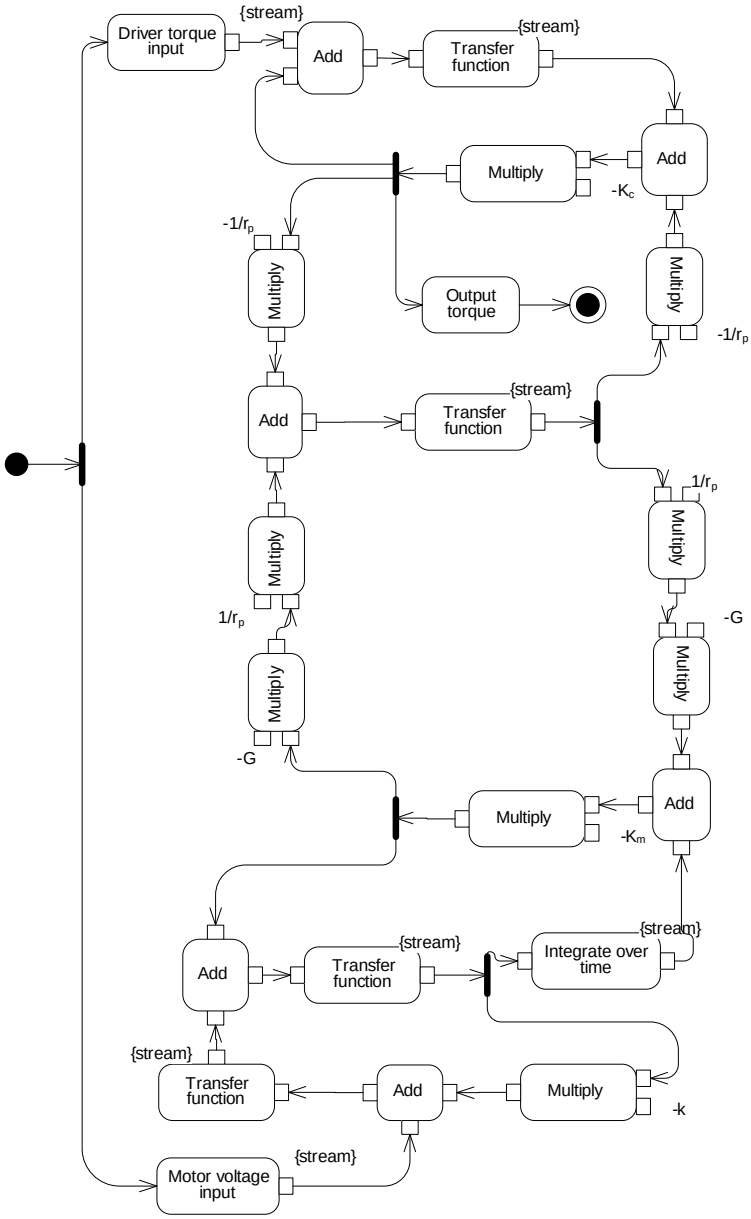

Figure 3.12: A SysML activity diagram of the EPAS system, adopted from [9].

```
library ieee;

use ieee.electrical_systems.all;

entity inductor is
        generic(L : inductance := 1.0E-3 ); -- [H]
        port(terminal t1, t2 : electrical);
end entity inductor;

architecture simple of inductor is
        quantity v across i through t1 to t2;
begin
        v==L*i'dot;
end architecture simple;
```

Figure 3.13: Example of VHDL-AMS code of an electrical inductor. Compare with Figure 3.6 and Figure 3.8.

SystemC-AMS and VHDL-AMS SystemC is a hardware description language for embedded systems, which is implemented as C++ functions and macros. This also makes it possible to simulate the model. The AMS extension stands for analog mixed-signals. Today, this extension is rather limited, including only linear signal flow and linear electrical networks [64]. The application is more focused on RF-design and sensor modeling than modeling mechatronic systems. VHDL-AMS is the implementation of SystemC architecture; a VHDL-AMS component definition is actually similar to a Simscape component definition, using across-through notation, see Figure 3.13.

ASCET ASCET is a tool which is similar to Simulink, but tailor-made for developing automotive embedded systems, and focusing on code-generation. [27].

Scicos Scicos is an open-source alternative to Simulink. Scicos also has some Modelica support, and recently a new Modelica toolbox called Coselica was released [79].

3.8 Conclusions

Here, the different evaluation criteria defined in the beginning of this chapter are summarized.

3.8.1 Language scope

Bond graphs

Using bond graphs is a means to convert from a “physical model” to a continuous causal model. Using the rules imposed by the bond graphs, it is an aid to the modeler to verify that he or she is creating a sane model, with right causality.

As opposed to signal-flow representation, there is a clear separation of signal and physical flow. For physical flow, the modeler is forced to consider the upstream physical flow. The bond graph method could be seen as a teaching tool, where experienced modelers eventually can skip the bond graph step, and make a signal-flow model directly.

Ptolemy II

The overall scope of Ptolemy II is to spread research results from Berkeley, and the results are different descriptions of Models of Computations. The result investigated here is a way to formalize the execution of a continuous causal system.

Simulink

Since Simulink is commercial software, the scope of the language is to sell software, and to satisfy all customer needs. From a tool-vendor perspective, making customers dependent on the software (customer lock-in) is a positive aspect, and that is why a strictly defined language definition using open interchange formats and a transparent simulation engine not necessarily are positive factors.

Modelica

The scope of the language is to have a unified way to represent models of technical systems, especially physical systems.

Simscape

The scope of this language is to extend Simulink with physical modeling and equation-based modeling. The language does not to comply with current standards for equation-based modeling, like Modelica or VHDL-AMS.

SysML

The scope of SysML is to be the systems engineering language of the future, where information of system components can be stored in a common model. It is more focused on information modeling than on simulation, although these aspects can coincide.

3.8.2 Levels of abstraction

In short, both Ptolemy II and MATLAB use the signal-flow model, and bond graphs also use the signal-flow model with added semantics. Modelica and Simscape use constraints/equations, which enable physical modeling. In SysML, it is possible to model signal-flow models using activity diagrams. The parametric diagram is basically the same abstraction level as constraint/equations, it is a means to visualize the constraints and how they are interrelated.

The included abstraction levels of these tools are further elaborated in section 5.2 on page 62, including a discussion on which abstraction level to use for different purposes.

3.8.3 Modularity

The *modularity* involves both the tightness of coupling between components, i.e. how they can be separated and reused, and to what extent the semantic rules of the system architecture enable or prohibit the mixing and matching of components [77].

When modeling physical systems, the physical modeling view (e.g. Simscape and Modelica) gives a higher potential for reuse of simulation components compared to the signal-flow tools MATLAB/Simulink and Ptolemy II. The major point is the interface definition, which makes the system convenient to compose. SysML also has reuse mechanisms, but at the constraint level, which is not as convenient for physical systems, as shown in section 3.6 on page 37. Bond graphs are reusable in the sense that parts of them can be reused, sometimes with changed causality.

3.8.4 Tool support

The MathWorks products Simulink and Simscape are both tools, rather than languages. Although there is a Simscape language, its purpose is to be able to run own-developed models inside Simscape and not by any other external tool. It is possible to generate C-code from both Simulink and Simscape using external toolboxes.

Ptolemy II can be seen as a language, implemented in a tool. Bond graphs and Modelica are general languages, which can be implemented in different tools, and from them it is possible to generate C-code. In SysML, however, there is no such possibility; although there are tools that support code generation from UML diagrams, the technique is immature today.

3.8.5 Tool transparency

Ptolemy II is the only open-source tool investigated in this thesis. C-code can be generated from different Ptolemy II domains, but not from the continuous-time domain. The transparent simulation engine is a key feature of Ptolemy II, so simulation results are thus predictable. It is rather well-documented how the Simulink simulation engine behaves, as elaborated in Chapter 4.3.5. How the Simscape and Modelica/Dymola simulation engines works is more hidden in the tool. Flattening a Modelica results in a vast set of equations, which have to processed and sorted by a tool. How Simscape sets up the governing equations for a system is not very well documented either.

Chapter 4

Case Studies

Throughout the PhD project, several case studies on modeling and simulation were conducted. They are presented in more detail in the associated publications. Here an overview of the cases is given, together with additional conclusions.

4.1 Modeling of a twin-screw compressor

A twin-screw compressor was modeled, using different detail and abstraction level. This case study highlights difficulties in making a usable simulation model of the right detail and abstraction level.

4.1.1 Overview

The twin-screw compressor was invented in Sweden in 1934, and then known as the Lysholm compressor. It has been used in many different fields: for compressing air, for refrigerating systems and more recently in fuel-cell systems. At a glance, the compressor has a rather simplistic design, but it has proven hard to thoroughly analyze the thermodynamic process in detail. In academia, state-of-the-art investigations are typically presented at the recurring Schraubenmaschinen conferences. In industry, the company SRM (Svenska Rotor Maskiner, which has historical connections to the Lysholm inventor) is active in the research and development of twin screw compressors, and provides licences for companies that develop and manufacture twin-screw compressors [86].

As a part of the NFCCPP project, a detailed model of a twin-screw compressor was to be created, in order to optimize its characteristics for fuel-cell applications. The model architecture was adapted from [46], where a twin-screw compressor is modeled using the custom-made software KaSim, implemented in C++. The concept is to model the compressor as a moving queue of chambers, where new chambers are instantiated and put first in the queue at the compressor inlet, and chambers eventually disappear at the outlet. These chambers have internal states,

which vary depending on the chamber volume, which in turn depends on the rotation angle of the compressor. Between the chambers, there is leakage, which is also geometry-dependent. In the KaSim model, both the chambers and the leakages are modeled as classes, which are instantiated and destroyed during simulation-time. In the Simulink model in Figure 4.1, the chambers and leakages are modeled as (virtual) subsystems. However, in Simulink, the model cannot be modified during simulation-time, so to model the instantiation and destruction of the chambers, the data (i.e. internal states) are exchanged between the subsystem blocks, at the instantiation and destruction times. The process is described in detail in Paper A.

From an engineering point of view, the results achieved from this model were not accurate compared with measured results. Maybe the model was not detailed enough; important phenomena seem to have been left out. For instance, the chamber model is one-dimensional, while the twin-screw compressor geometry is three-dimensional. It is also difficult to calculate the states in the chambers, as the volume approaches zero. Yet another problem was to get hold of the geometry curves, crucial for the simulation. Instead, in a follow-up paper [71], another model was developed that combines the adiabatic model (that consists of a single equation) with scaling factors. This way, the model is accurate "by design". The downside of that model is that it can not be used to test new designs, or to evaluate different geometry curves, which the chamber model can. Instead it could be used in system design, to predict performance of a similar design but different size.

4.1.2 Conclusions

Although the first model of the twin-screw compressor could be seen as a failure, there are important conclusions to draw from it:

- It is interesting to see that a fairly non-complex model is difficult to model using Simulink. Somehow we have reached the limit of the Simulink abstraction. Many other engineering domains use custom-made software (e.g. KaSim for displacement machines, GraspIt! for robotic grasp simulation [55]), possibly because the abstraction-level is set too high, or missing out important things, in general purpose tools.

- The developed model is a purely physical model - no control system is modeled. However, the model is hybrid in the sense that new chambers must be created during simulation time, in other words a self-modifying model is needed. In the C++ model where the user also controls the solver, this functionality could be implemented. In the Simulink model presented in [81], major workarounds had to be made. Likewise, using Modelica would lead to similar problems as it contains no mechanisms for instantiating new classes during simulation-time.

- This is also an example of how the usefulness of a model is not proportional to its level of detail. Although a relatively complex model was made compared

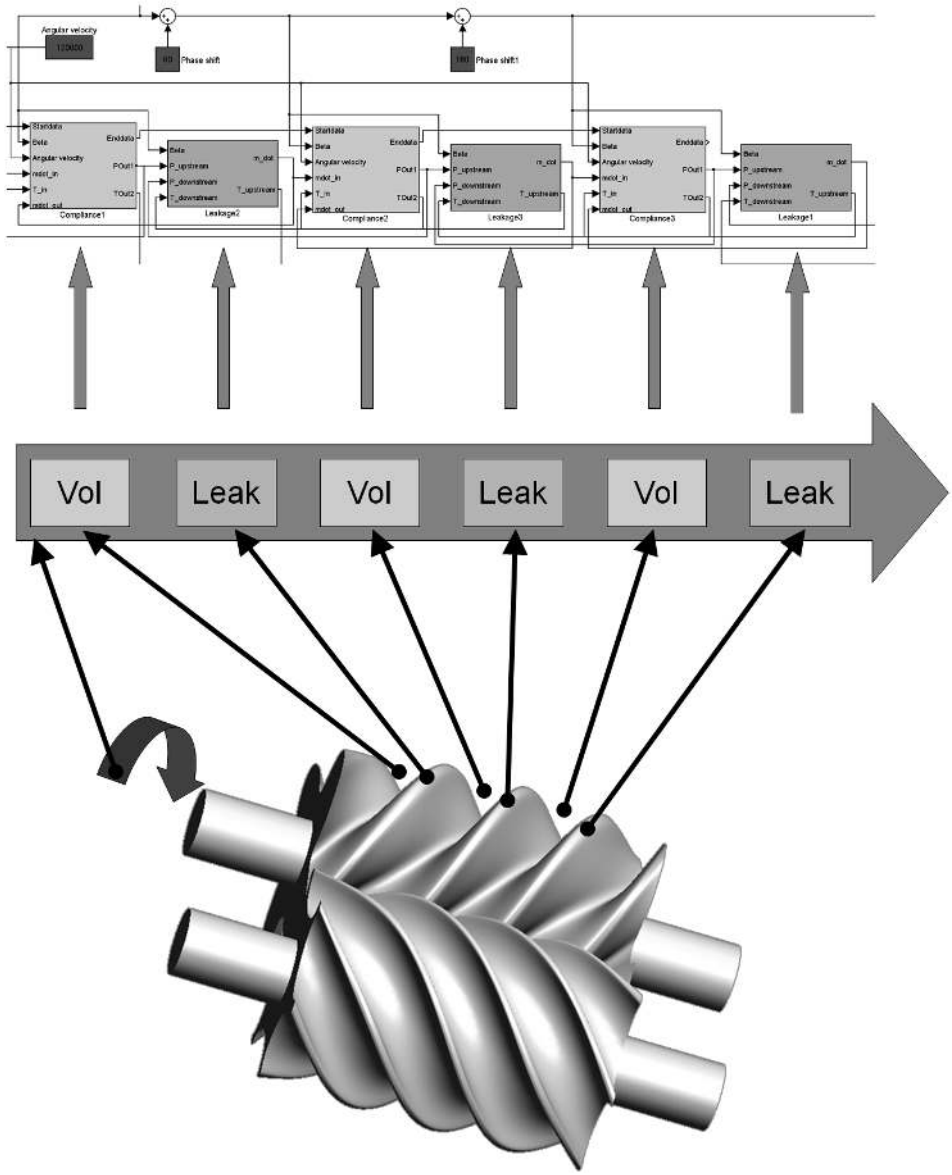

Figure 4.1: A Simulink model of the twin-screw compressor. As the compressor rotates, new chambers are modeled, and the *Enddata* is propagated to *Startdata*. This way a queue of chambers is modeled.

with an adiabatic model, it did not provide any useful simulation results.

4.2 Modeling of a fuel-cell test environment

In the NFCCPP project, a complete fuel-cell vehicle was modeled using MATLAB/ Simulink. The target application of the model is Hardware-in-the-Loop simulation, so component manufacturers can test their components in a complete system model.

4.2.1 Overview

A complete fuel-cell system in a prime mover (e.g. a car) was modeled, including air management system, hydrogen supply from an on-board gasoline reformer, drive train, and driver. The control system and the power management system are modeled separately, see Figure 4.2. The system is simulated over the New European Driving Cycle [25], which duration is 1220 seconds and the execution time is about the same on a standard desktop computer. The model is described in more detail in Paper B.

4.2.2 IP protection of simulation components

As a part of the NFCCPP project, an investigation was carried out on how to protect the Intellectual Property of simulation models, allowing cross-enterprise simulation, and interchange of simulation models. To encourage the exchange of state of the art models across enterprise boundaries, component IP-protection is crucial. A vendor wants to control the usage of his component, so that only trusted partners can perform simulations. In addition, only approved simulations must be feasible to conduct.

IP-protection includes the protection of modeling know-how, such as what equations are used and what phenomena are modeled. The model should be a "black box", i.e. only input and and output should be revealed, and not the content inside. However, the black-box solution is not completely safe either. If the output and the input are recorded from the black box, the parameters inside the box could be identified. This technique of reverse-engineering the model is here referred to as *clamping*.

Complete IP-protection requires both social and technical measures. The social measures relate to business rules and laws for preventing illegal usages of IP, such as business contracts, patents, copyrights, and trademarks. The technical measures relate to technologies and tools that provide necessary enforcements of protection, such as cryptography. It should be noted that there are no completely secure technical solutions; however a solution can be classified as "secure enough" if the effort it takes to crack the code is higher than the value of the information inside it [78].

There are two main tasks that the protection solution needs to handle: encryption and access control. *Encryption* means that the models should be well

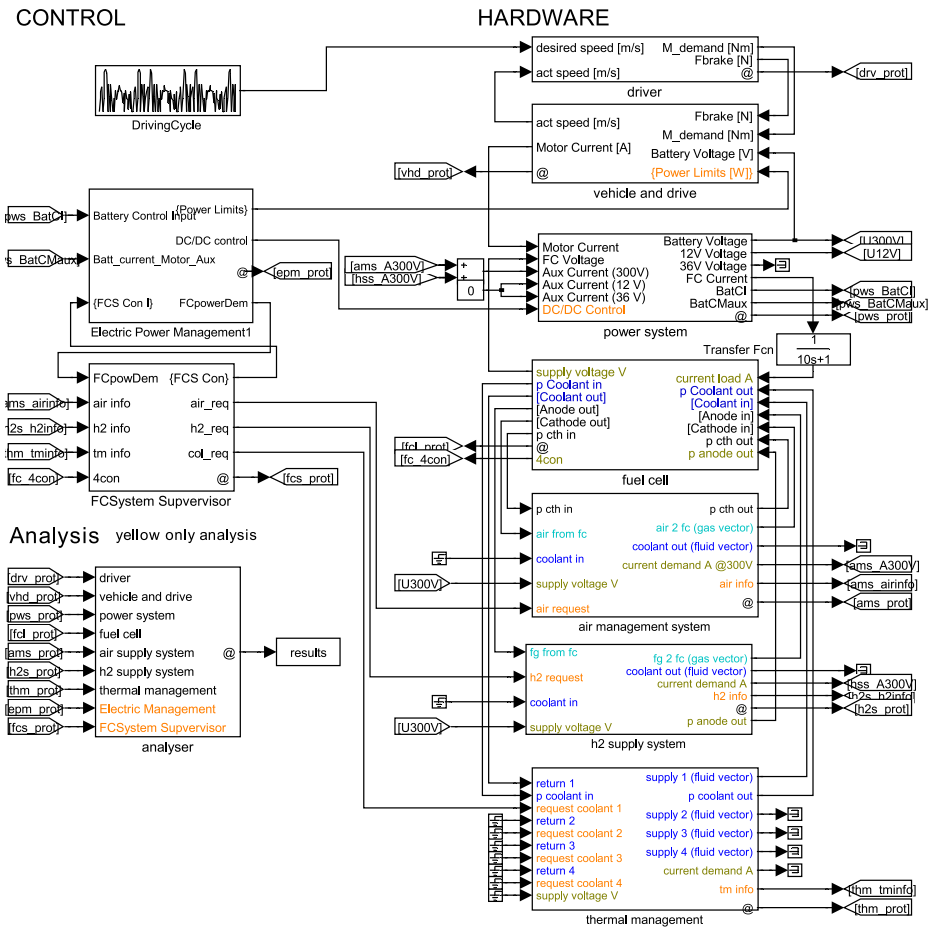

Figure 4.2: The NFCCPP system model: Electrical and embedded system to the left and the plant model on the right-hand side.

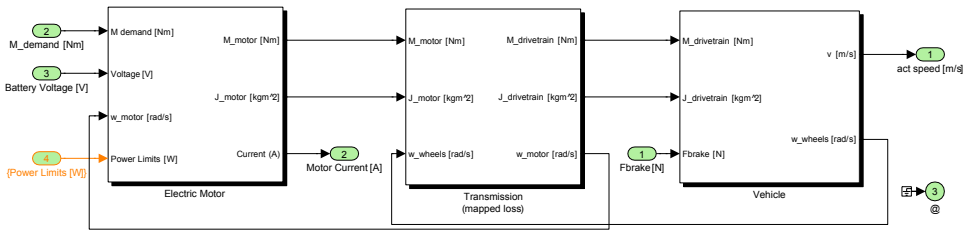

Figure 4.3: The drive line of the NFCCPP model. Apart from the torque and velocity power connection, the moment of inertia is also propagated, and the velocity is integrated as a function of the combined intertias and torque in the *Vehicle* block. This can be seen as a violation of bond-graph methodology, although a correct and somewhat modular solution is achieved.

hidden, that it will not be possible to reverse-engineer the code and find out how the component vendors' model is designed. *Access control* means that the model owner will be able to control the simulations that are performed and by whom they are conducted, and thus preventing clamping. In Paper B three different ways to achieve this are presented: *Centralized simulation with remote user control*, *Localized simulation with simulation-time model usage control* and *Parallel distributed simulation*. A proof-of-concept implementation of *Localized simulation with simulation-time model usage control* was made, see [82] for further reading on this.

4.2.3 Conclusions

During the project, the definition of interfaces between the components was a major issue, see also section 3.3 for the technical details why this is difficult. In Paper B, the outcome of the interface definition task is presented, showing clearly defined interfaces. In practice, ad-hoc interfaces were used, just like one is forced to do when using Simulink. An example is given in Figure 4.3. System integration of the components was also a bit of a challenge, since algebraic loops can occur when many components are assembled. In NFCCPP, these algebraic loops were solved by the technique shown at the bottom of Figure 2.5 on page 23.

On the positive side, the model runs well, and commercial fuel-cell simulation has been made on parts of the model. Since the interfaces to components are explicitly defined, it is not too difficult to plug in your own component model and see how it behaves in a fuel-cell environment. Another conclusion drawn is that although it is technically feasible to integrate models, it might not be feasible from a business perspective. Simulation models contain important intellectual properties such as knowledge of the product and its key characteristics, and the suppliers do not want to share this with either the competitors or the OEMs.

4.3 Modeling physical systems together with embedded systems

In the ATESST project, the EAST-ADL2 language, described in section 2.2.2 on page 20, has been developed to model automotive embedded systems. To ease integration between modeling languages for embedded systems and physical systems, model transformation techniques and means to describe languages using one another have been developed, tested and evaluated.

4.3.1 Our approach to model Modelica models using SysML

In Paper C, an attempt to implement a Modelica model using SysML diagrams is made. In comparison to ModelicaML[1] [72], we try to make use of the native SysML constructs, e.g. parametric diagrams and constraint blocks. The approach is to use SysML blocks to represent Modelica components, and parametric diagrams to model the equations. Modelica acausal ports were represented using bi-directional SysML *FlowPorts*, and Modelica connectors using *ItemFlow*, and thus a "physical modeling" view of the system could be achieved, see Figure 4.4. The Modelica equations are modeled as SysML constraints. In Modelica the TwoPin[2] class is used as a base for many electrical components, which have two pins, and an equational relation between the voltage and the current. A component inherits the TwoPin class, and its constraints. In SysML, we modeled this inheritance using a parametric diagram, see Figure 4.5. The parametric diagram also contains links to the ports. As stated in Paper C, this method is not satisfactory, since parametric diagrams do not contain any sum-to-zero connections. The result will thus be that the underlying equations do not match the modeled system. Workarounds must be made; see Paper C for more details.

4.3.2 SysML effort to model Modelica

In [43], a more thorough effort is made to model Modelica systems using a similar method to the one described above. The conclusions are similar: if components are to be connected in parallel, extra blocks must be modeled, see Figure 4.6. In Paper C, we call this extra component *FlowSplit*, in [43] it is called *MechJunction*. Another solution is to extend parametric diagrams with a special connector, called «*connectClause*», which has the semantics of a Modelica connector. One difference between our approaches is that [43] represents Modelica components, like springs and dampers, inside parametric diagrams. One reason we did not use parametric diagrams is that we interpreted the SysML specification that only constraints are

[1] There was little information available on ModelicaML when writing Paper C. ModelicaML is presented in section 4.3.3.

[2] This class is called OnePort in Modelica Standard Library 3.0 [57]; the TwoPin class is a similar class. The TwoPin name for this class is used in [30], and in Paper C, so we continue using it here.

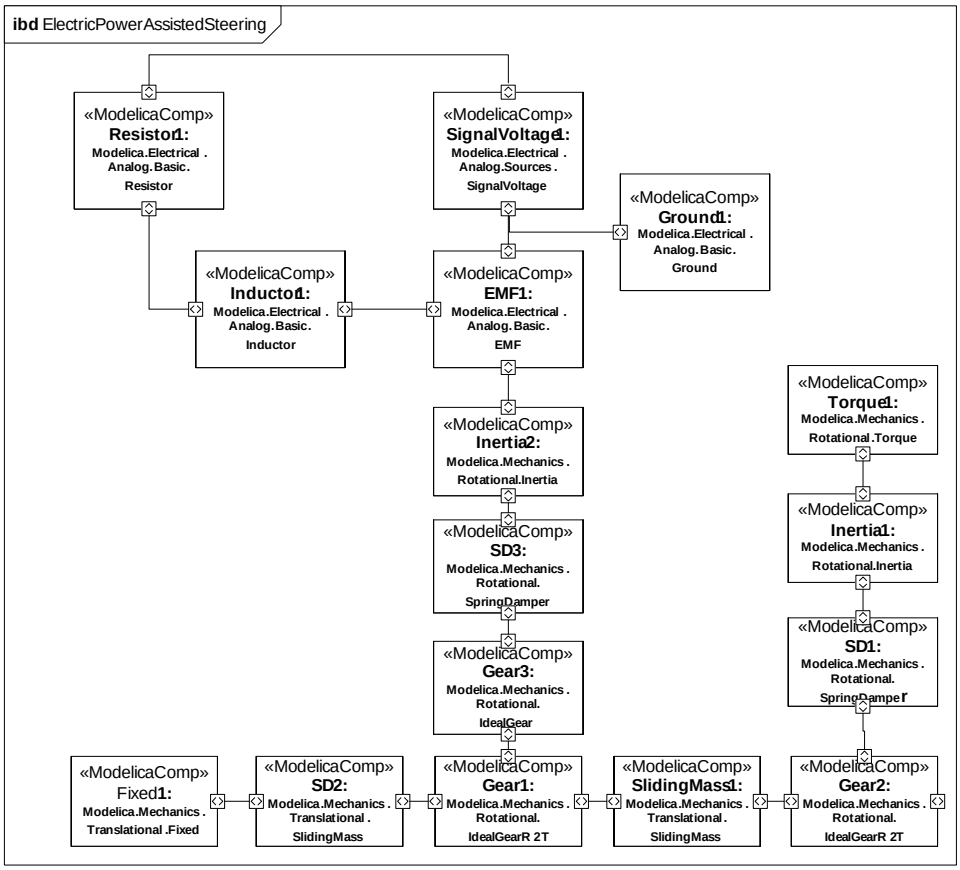

Figure 4.4: SysML Internal Block Diagram of the EPAS system, modeled as in Paper C, where SysML blocks represent components from the Modelica Standard Library. Compare with the original Modelica model in Figure 3.7.

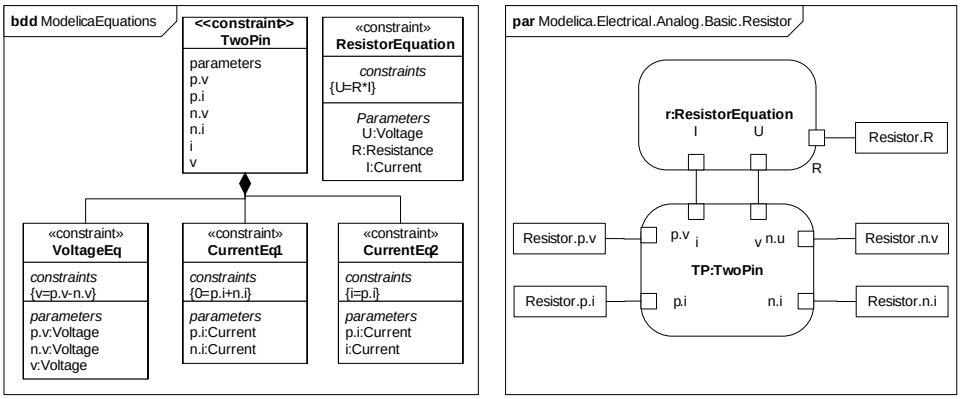

Figure 4.5: Block definition diagram and a parametric diagram describing the constraint blocks of the Modelica resistor in SysML, as in Paper C..

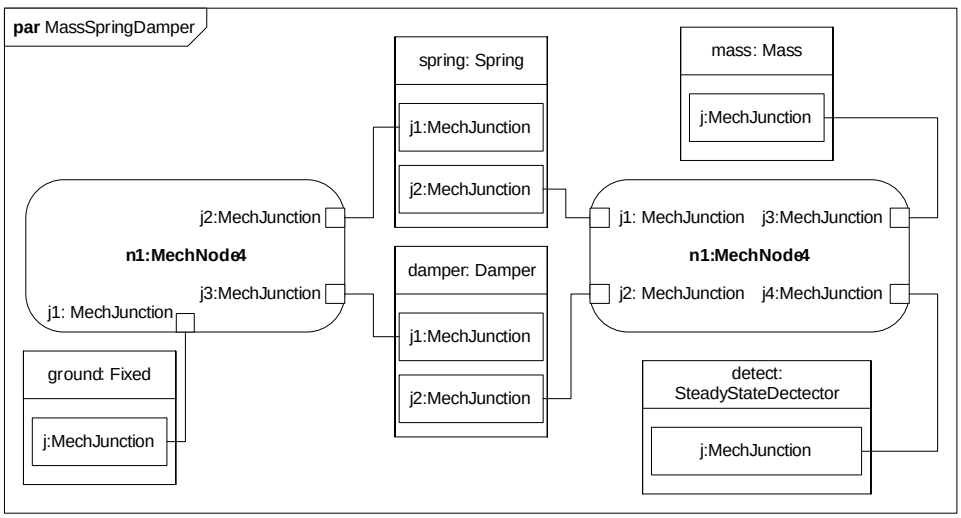

Figure 4.6: Modelica components in SysML, modeled like in [43]. This mass-damper system can be seen as the lower-left of the EPAS system in Figure 3.7. To get the sum-to-zero relation for force, additional MechNode blocks must be modeled.

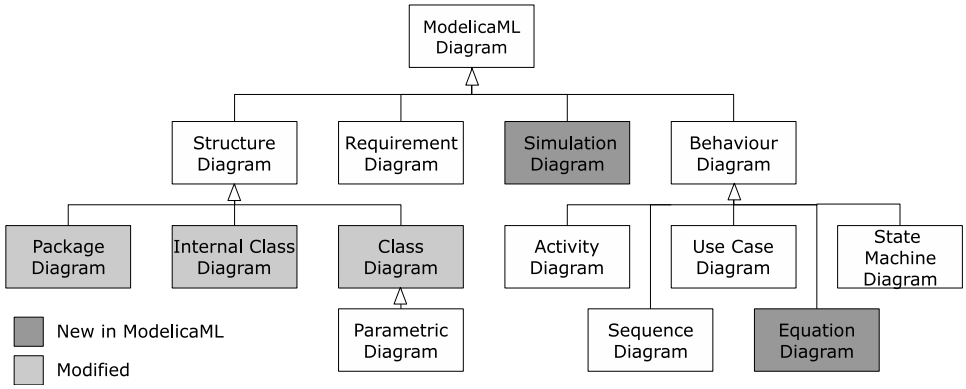

Figure 4.7: The ModelicaML profile for SysML [72].

to be modeled in parametric diagrams, and not structural entities. Due to technical issues (it is difficult to comprehend if there is a limitation of the tooling environment, or the model), the parametric diagrams for Modelica components are abandoned in favor of internal block diagrams. Consequently, the end result is similar to our representation in Figure 4.4.

4.3.3 ModelicaML - Modelica effort to integrate Modelica in SysML

Another effort to combine use of SysML and Modelica is the ModelicaML language, described in [72]. It is a complete implementation of Modelica in SysML, and hence it supports modeling with all Modelica constructs and properties. It is "partly based" on SysML, and the ModelicaML meta-model is consistent with SysML to ensure SysML-to-ModelicaML conversion compatibility.

As seen in Figure 4.7, the SysML block diagram and internal block diagram have been modified to *class diagram* and *internal class diagram* respectively (as described in section 3.6, these two diagrams are in turn derived from the UML class diagram and composite structure diagram). The internal class diagram provides a physical modeling view of the system, so it resembles Figure 4.4. Package diagrams are also extended, to support Modelica specific features. Instead of using SysML constraint blocks to model equations (as in Figure 4.5) an *equation diagram* of ModelicaML is used, where e.g. the TwoPin class is modeled.

In a follow-up paper [75], it is suggested that ModelicaML should abandon using UML/SysML as a meta-meta model to describe the language, and instead use MOF (see [63] and section 2.2.1) in form of Eclipse Modeling Framework [23]. Reasons for this include:

- **UML is semantically unsound**

By this it is meant that there is no way to define what constitutes a valid UML model, or a test suite to show UML compatibility.

- **UML's profiles are a problem**

One problem with UML profiles is that the users often desire to change UML concepts, rather than constraining them.

- **UML's sublanguages are a problem**

UML contains both the *Object Constraint Language* (OCL) and *Action Language*, which will be inherited in UML profiles.

4.3.4 Conclusions of SysML to Modelica integration

There have been several efforts to combine Modelica and SysML. In fact, it is not a new idea, see e.g. [7] for an early reference on UML/Modelica integration. Today, there is still no unified, verified way to relate these languages. There are however some conclusions that can be drawn from this comparison:

- The SysML internal block diagram seems like a good candidate to use to represent interconnected Modelica components, although they are not used directly by ModelicaML.
- The parametric diagram does not enable physical modeling, although it is acausal. This is a reason to separate physical modeling and acausal model abstractions.
- The sum-to-zero relation for flow-variables has proven difficult to realize in SysML using parametric diagrams. One solution to this would be to introduce a sum-to-zero connector in the parametric diagram.
- The conclusion from [75] is that *domain specific* modeling techniques (mentioned in section 2.2.1) are favorable compared with SysML, if the motive is to describe the Modelica language in a concise way.

Further discussions between Modelica and SysML communities on how to proceed are ongoing.

4.3.5 Describing Simulink behavior using activity diagrams

In Paper D, Simulink behavior and structure are mapped to UML/SysML. The structure of Simulink models, i.e. blocks and subsystems, are mapped to composite structure diagrams. Behavior is mapped to different interacting activity diagrams. A key factor of describing the behavior of Simulink is the execution order of the

blocks. When a user is presented to a signal-flow model like in Figure 3.4, it is intuitive to think that the order starts with the input, and then executes the blocks to calculate the output. However, as the blocks of a Simulink model describing differential equations are in one of many loops, it is not trivial to decide where to begin or end computations.

Simulink uses two rules for defining the order of the blocks [53][3]:

1. If a block drives the direct-feedthrough port of another block, it must appear in the sorted order ahead of the block it drives. Direct feedthrough means that the output is controlled directly by the value of an input port.

2. Blocks that do not have direct-feedthrough inputs can appear anywhere in the sorted order as long as they precede any direct-feedthrough blocks which they drive.

The actual ordering of the blocks can be displayed by the *slist* MATLAB command. This ordering can also be altered by assigning priorities to blocks. So, it can be seen that Simulink only has partial order, which in SysML can be described using activity diagrams. This activity diagram can be derived using the following method:

1. Create a start node and then a fork to all blocks with non-direct feedthrough.

2. Blocks with multiple inputs will have join nodes in front of them, and signals going to many blocks will have fork nodes after.

3. Insert all activities according to the precedence, as described above.

4. Finally, create a join node, and propagate all activities here. After the join node, the end node is inserted.

There are two output activity diagrams: *Outputs.minor*, and *Outputs.major*, of which the first is used for internal steps in e.g. the Runge-Kutta solver. Thus, outputs (e.g. *Scope* or *To Workspace* blocks) are excluded from the *Outputs.minor* diagram. The resulting *Outputs.minor* activity diagram is shown in Figure 4.8. In Paper D, Figure 7 and Figure 8 represent the other activities that together describe the Simulink execution loop.

Conclusions

Using the *Outputs.minor* diagram in Figure 4.8 is a means to understand in what order Simulink blocks execute, and why the execution order can differ between identical models. It also visualizes how the calculations could be parallelized. Modeling the solver in UML could also be useful and opens up future possibilities such as:

- The simulation could be run using a UML activity diagram virtual machine.

[3]These rules are slightly rewritten in this version of the Simulink user manual (R2008b), compared with Paper D (R2006a)

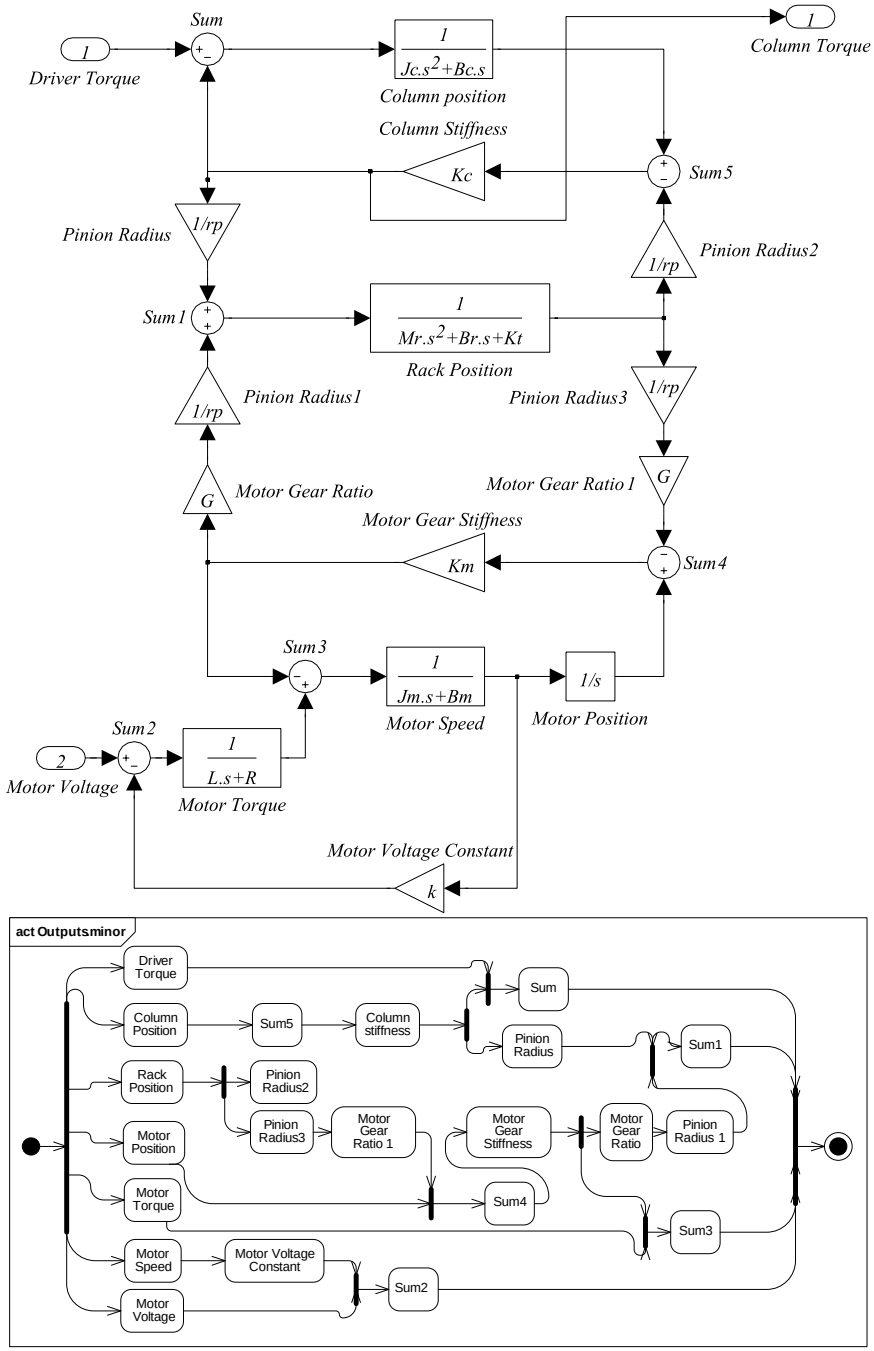

Figure 4.8: Activity diagram describing the execution order of the Simulink model above. The *Column Torque* block is not included; it is however included in the *Outputs.major* diagram which otherwise is identical to this diagram.

- Implementation code could be generated from the activity diagrams.
- The functions can be further developed, assigned to processes and hardware, scheduled etc. in UML.

Chapter 5

Methodological Concerns When Modeling Continuous-Time Systems

This chapter presents the various conclusions that can be drawn from the survey of modeling languages, i.e. the EPAS study in Chapter 3, and from the case studies presented in Chapter 3.

Section 5.1 discusses interfaces between components, section 5.2 is devoted to the abstraction levels of the different tools. Section 5.3 is focused on transformation from the physical modeling realization level to lower levels, and section 5.4 addresses which tool to choose for what purpose.

5.1 Effort-flow vs potential-flow vs across-through

A central characteristic when modeling lumped systems, is the conjugate variables in Table 5.1, named *effort-flow* (bond graphs), *potential-flow* (Isermann, Modelica) or *across-through* (Isermann, Simscape), which are used to define interfaces between components.

- According to Isermann's definition, across-variables are variables that can be measured between two terminals, also named two-point variables or trans-variables [41]. Through-variables can be measured at one terminal, and are thus called one-point variables, or per-variables. In the electrical and the fluid domains they are the same, but in the translational and rotational mechanical domains they are switched. This is referred to as the *dualism* issue [41].
- Effort-flow is promoted by bond-graph theory, although it is pointed out that what is defined as effort and flow respectively can be interchanged [50].
- Modelica labels its conjugate variables as potential-flow, defining potential quantities as quantities connected to the same port as being equal, and flow

quantities as obeying laws analogous to Kirschhoff's current law, summing all flows into a specific port to zero. Confusingly enough, this is more close to the across-through viewpoint.

- In bond-graph theory, if the conjugate variables are multiplied, the result will have the unit power (Watts), and hence the interface is named *power-port.* If the product is not power, they are referred to as pseudo-bond graphs [45]. This viewpoint is also shared by Hirz [37].
- It is notable Modelica uses displacements instead of velocities in mechanical translational and rotational one-dimensional systems. One reason for this could be that the actual displacement is a key characteristic when modeling mechanical systems, due to spatial constraints. One side-effect is that for rotating machines the absolute rotating angle can get very large, which can lead to numerical problems. This issue is addressed in the 3.0 version of the Modelica Standard Library, using a relative angle and displacement per default.
- The previous issue highlights that it can be difficult to know which part of Modelica to address: Is it a language issue, a tooling issue or an issue with the standard library? The Modelica Language specification does not prescribe that any particular conjugate variables should be used, so in principle a new mechanical library with power-port variables (i.e. force/torque and velocity) could be designed. However, it is not certain that this library will work in current tools, and it will not be compatible with current models in the standard library.
- As mentioned earlier, Simscape uses the across-through classification. For the foundation library, power-port variables are used (see Table 3.2 on page 36), but since the documentation also lists another table of across-through variables (Table 3.1 on page 36) that can be used, Simscape is not limited to power-port variables for user-defined interfaces and components.

So, it can be seen that if we want to describe a physical port or connector (e.g. in SysML), there is a naming confusion in academia. If the potential-flow nomenclature is used, it will refer to different variables in bond graphs and Modelica for mechanical systems. We can not name such a connector a power-port either, since the product will not be of the dimension power in all cases. Using the across-through classification is more general, and fits best with the physical modeling view.

5.2 Five realization levels of physical systems

Models of continuous-time systems can, just like an embedded system, have different levels of abstraction. Since the expression *abstraction level* is reserved in EAST-ADL2, they are called *realization levels* throughout this thesis. These realization levels were created to provide answers to such questions as:

Table 5.1: A comparison of different ways to describe the conjugate variables.

	Bond graphs [45]		Modelica [30]		Functional modeling [37]		Potential - Flow [41]		Across - Through [41]	
System	***Effort***	***Flow***	***Potential***	***Flow***	***Effort***	***Flow***	***Potential Diff.***	***Flow***	***Across***	***Through***
Electrical	Voltage	Current	Voltage	Electric current	Electro-motive force	Current	Electric voltage	Electric current	Electric voltage	Electric current
Mechanical Transla-tion	Force	Velocity	Position	Force	Force	Linear velocity	Power (N)	Speed	Speed	Power (N)
Mechanical Rotation	Torque	Angular velocity	Angle	Torque	Torque	Angular velocity	Torque	Angular speed	Angular speed	Torque
Flow, hydraulic	Pressure	Volume flow rate	Pressure	Volume flow rate	Pressure	Volume-tric flow	Pressure diff.	Volume flow	Pressure diff.	Volume flow
Magnetic	Magneto-motive force	Magnetic flux	Magnetic potential	Magnetic flux rate	Magneto-motive force	Magnetic flux rate	-	-	-	-
Heat	Temp-erature	Heat flow	Tempe-rature	Heat flow rate	Temp-erature	Heat rate	Temp-erature	Heat flow	-	-
Chemical	Chemical potential	Mole flow rate	Chemical potential	Particle flow rate	Affinity	Reaction rate			-	-
Chemical	Enthalpy	Mass flow rate	-	-	-	-	Enthalpy differ-ence	Mass flow	-	-
Mechanical 3D	-	-	Position (3D) and Rotation angle (3D)	Force (3D) and Torque (3D)	-	-	-	-	-	-
Thermo-dynamic	-	-	-	-	-	-	Entropy flow	Tempe-rature	Entropy flow	Temp-erature

- What is the behavior of Simulink and Modelica models?
- How is it possible to connect model x and model y?
- How is it possible to convert model x into model y?

The answer to the first question is that the behavior of simulation models depends on which realization level is being regarded. Some of these languages specify behavior only at certain realization levels, while other languages specify transformations into lower levels of abstractions, more or less explicitly, see Table 5.2. The short answer to the next two questions is that models of the same realization level are, at least in theory, interchangeable.

5.2.1 Physical modeling models

The physical modeling realization level is a topological schematic view of how components are interconnected, with physical connectors. The term "physical modeling" is used by [30], [88] and [52] for these kinds of models, it seems like academia and industry has found a name for this type of models. The name is however not ideal, it could be mistaken for physical mock-ups of the system, and models at this realization level will be "physical modeling models", which is a tautology. In VDI 2206 [93], the word topological model is used, which maybe is a better word to describe the characteristics of these models.

5.2.2 Constraint models

A constraint model is a model where the behavior is represented as a set of constraints, which could be equations. Equations can be acausal, in the sense that different variables can be used as input. The word *acausal* could be misleading, since it is a negative definition (i.e. models not being causal). Constraints can have a specified cause and effect, for example:

```
y = if v > limit then limit else v;
```

Here, `y` must be an output from the input variable `v`, since an inverse function is not defined. The statement above is not a mathematical equation, so the word constraint is a better word than *equation models*, *mathematical models* or similar.

In the Modelica Standard Library, it is the conjugate variables that are acausal. E.g. for a standard resistor, the resistance R is given as a parameter, and either the current I or voltage U is calculated from the formula:

$$U = RI \tag{5.1}$$

So, it can be seen that in Modelica, as it is implemented in tools and model libraries, some variables are parameters and thus must be used as input.

5.2.3 Continuous causal models

Continuous causal models are used by control or signal processing engineers to describe a system and its controller/signal transformation. This abstraction level corresponds to how continuous-time systems are modeled using Simulink block diagrams. There is no one-to-one mapping of an equation to a continuous causal representation; the mapping depends on which variables are used as input and output, but also on whether integral or differential causality is chosen. The bond graph is also a continuous causal formalism, with more information added.

5.2.4 Discretized models with solver

Discretized models typically have update and output functions, as a result of inputs and a time step. The selection of solver is crucial to get a valid simulation result, including consideration of stiff systems[1], selection of time-step etc. When hybrid models are simulated, the solver will also need to take into account zero-crossing effects. This level of abstraction can be described by the same means as a computer program, e.g. UML activity diagrams, state machines, C-code.

5.2.5 Discretized models with solver and platform implementation

Especially for real-time Hardware-in-the-Loop simulations, it is crucial that the simulation can be run in real-time. The platform can have a limited numerical resolution, have memory constraints etc. In a real-time system, the calculation time needs to be taken into account, and possibly scheduled.

5.2.6 Comparison with other approaches

These realization levels have been developed throughout the work; evidence of this can be found in Paper C and Paper D, where they have different names. A comparison with similar approaches follows here.

Sen/Vangheluwe approach

In [80], an approach to transform physical models into bond graph models using meta-modeling and graph rewriting is presented. This is interesting, since it is a missing link in Table 5.2. As seen in section 3.1, there are methods to derive a bond graph from e.g. a mechanical system, and these methods has been implemented using graph rewriting. The model formalisms used includes [80]:

[1] A stiff system has such mathematical properties that makes it difficult to analyze numerically; special solvers must be used.

	Bond graphs	Ptolemy II CT	Simulink	Modelica	Simscape	SysML
Physcial modeling	-	-	Using Simscape	Using Standard library	Yes	Available through profiles
Trans-formation ↓	-	-	Using Simscape	Using the language	Hidden in tool	-
Constraint models	-	-	Using Simscape	Native	Simscape language	Parametric diagrams
Trans-formation ↓	From physical to causal	-	-	-	-	-
Causal models	Native	Native	Native	Native	Using Simulink	Using activity diagrams
Trans-formation ↓	Possible, using algorithms	Explicit, using a director	Using simulation engine, described in docu-mentation	Tool (e.g.) Dymola transforms causal and acausal models	Using Simulink simulation engine	Using our approach in Paper D
Discretized models with solver	-	In principle, not explicit	Native	Using Dymola	Using Real-time workshop	Using our approach in Paper D
Trans-formation ↓	-	-	Not explicit	-	Not explicit	-
Discrete models with solver and execution platform	-	-	Using Real-time workshop, Targetlink	-	Using Real-time workshop	-

Table 5.2: Coverage of the langugages and tools on different realization levels, and how transformations into lower levels are defined.

- *Real World Visual Modeling Formalism*: This is a higher realization level than physical modeling. In the EPAS case this would correspond to Figure 3.9 on page 39.
- *Idealized Physical Modeling Formalism*: This has also been labelled as physical modeling in this thesis.
- *Acausal and Causal Bond Graph Formalism*: In our approach there is no such thing as an acausal bond graph, although it could be seen as an intermediate step in the generation of a bond graph. Causal bond graphs and continuous causal models are both on the same realization level (see section 3.1).
- *Trajectory Formalism*: This is a formalism of the simulation results, basically to describe a graph of the results.

The approach uses the Modelica Bond Graph Library [18] and Dymola as a simulation engine. Modeling bond graphs in Modelica could be seen as *abstraction inversion*, i.e. implementing lower levels of abstractions using higher level of abstractions, which is undesirable. To a certain extent the concept of using Modelica is to get rid of causal representations. However, for educational purposes this could be useful. Bond graphs are also suggested to be used inside Modelica models for multi dimensional mechanical models, see [95].

VDI 2206

In VDI 2206 [93], the following levels are used:

- *Topological model*: The topology of the system to be simulated is modeled as the "arrangement and interlinking of function-performing elements".
- *Physical model*, describing system-adapted variables, such as mass, length, resistances, number of connections etc. It is pointed out that this is a domain-specific view.
- *Mathematical model*: The physical model is transferred into an abstract representation of the model, using mathematical descriptions. The modeler can choose the depth of the modeling, using more or less detailed models of e.g. friction, applying linearization of non-linear properties etc.
- *Numerical model*: The mathematical model is prepared in such a way that it can be simulated, i.e. it is discretized and a solver is chosen. The model is also parameterized, i.e. concrete number values are assigned to variables.

The combination of topological and physical models of VDI 2206 corresponds to our realization level physical modeling. Mathematical models represent constraint models, and numerical models represent our two lower realization levels.

Platform-based design

Sangiovanni-Vincentelli [76] presents a concept of using abstraction levels. Here the application is low-level design of electrical circuits, *system-on-chip* (SoC). A *platform* is a layer in the design flow for which the underlying design-flow steps are abstracted. By defining suitable platform layers, and associated transitions between these layers, the design process can be improved. In a platform enough information transpires from lower levels of abstraction to allow design space exploration with sufficiently accurate prediction of the final implementation.

Using the True-Time tool [19], real-time kernels and communication networks can be modeled using Simulink blocks. This way, the Simulink model can be seen as a platform, where combined development of control algorithms and their implementation can be performed.

Leaky abstractions

Leaky Abstractions is a term that originates from software engineering [84]. The core meaning is that when using a simple high-level abstraction of something much more complicated, issues from this underlying implementation *leak* unintentionally to the high-level abstraction. An example is a two-dimensional array that could be faster to browse in one direction than the other, depending on the underlying implementation. For the user, not knowing about the underlying implementation, this can be rather confusing, for example when error messages from the lower levels of abstractions are shown on the higher abstraction.

From this point of view, one could say that it is the implementation that is leaky, and not the abstraction itself. However, [84] claims that "All non-trivial abstractions, to some degree, are leaky" and that there is no such thing as a perfect implementation of an abstraction. An example of a leaky abstraction would be that the signal-flow model allows algebraic loops to be modeled. However, good modeling practice is to avoid algebraic loops if possible, due to poor simulation performance.

5.3 Different approaches for transforming from physical modeling to constraints

A common problem is to convert a physical model of interconnected components, like the EPAS system, into equations. In section 3.1, we presented the bond graph approach of converting a physical model into a bond graph, which in turn can be converted to equations. This method has been implemented using meta-modeling and graph rewriting in [80], as mentioned in previous section. In this section, other approaches are discussed.

5.3.1 Modelica method

Modelica uses the *connect* statement to connect components, and equations can be associated with the connector. In the connector, flow variables are defined. When two ports are connected, potential variables are set to equal, and flow variables are summed to zero.

The system of equations that is generated by this method is large, containing many equations with duplicate variables. In our EPAS example, 113 equations containing 113 variables were generated from the native Modelica descriptions. From equations 3.1 - 3.4, we know that this system can be described by four differential equations, containing 11 variables if we count all derivatives. The electrical system alone consists of 47 variables, 13 of them representing the electric current. In this particular example, there is only one current, as seen in equation 3.4. The reason for the large number of current variables is that the *OnePort*-class (see Figure 4.5 on page 55) defines three currents for each component: one component current (i), one current for the positive port ($p.i$ which is equal to i) and one for the negative port ($n.i$) that equals the negative value of the current. The implication of this is that even for small systems, the native Modelica descriptions are inconvenient to solve manually and a tool is needed.

5.3.2 Network equations

For electrical networks, there are many techniques to generate the governing equations and perform computer simulation using matrix methods, see e.g. [94]. Since electrical networks consist of across-through variables, these methods should be possible to extend to include other domains. In [85], it is claimed that the approach of using across-through variables in a network with nodes and edges, can be used for different application domains. This approach seems promising, but has not been investigated as a part of this thesis.

5.3.3 Higher-order functions

In [12], a concept is presented on how to create an acausal modeling language by using the concept of higher-order functions, used in functional programming languages. This would imply that the transformations are a part of the language. The results are preliminary, and it would change Modelica in a fundamental way, so here the concepts are presented using a completely new language. One interesting thing to note is that nodes (here named wires) are used in this approach, just like the network methods mentioned above [85]. In Modelica, nodes are somehow implicit; ports are connected directly to each other. Another interesting thing is that this method would allow models to be created and modified at run time, which would be an enabling technology for e.g. a chamber model of displacement machines, as described in section 4.1.

5.3.4 MapleSim

The relatively new toolbox for Maple, MapleSim, uses the physical modeling view, and it is claimed to use symbolic manipulation to derive simplified equations. As mentioned earlier, MapleSim uses Modelica models "behind the scenes", and the models are derived from the standard Modelica library [52]. This tool was not part of the survey, since it was not possible to get hold of a license.

5.4 Choosing realization level and tool

The choice of simulation language or tool is of course dependent on the task. Other soft parameters include software licenses, legacy models and knowledge. The user starts out on one of the three highest realization levels, here it is discussed what level to choose.

5.4.1 Signal-flow or constraint models

A review of the statements given in the Ptolemy II documentation [14](see section 3.2.1 on page 31) is made when choosing between constraint models and signal-flow (i.e. causal) modeling.

- **The signal-flow model is more abstract**

The signal-flow mode does abstract away things such as units, physical domains. It can even abstract away if functions are implemented in hardware or software. Using a classic example from control theory where a mass-damper system is PID-controlled by a force actuator, is that in the signal-flow model equivalent to add mass, as to increase the derivative constant of the PID controller. If a functional-modeling approach is used, the signal-flow model is indeed more abstract. This is an interesting feature of the realization levels, making them stand out from abstraction levels: The final product is at a higher realization level for the plant (when selecting actuators), but a at lower abstraction level for the embedded system (when implementing control functions in microcontrollers)

- **The signal-flow model is more flexible and extensible**

As shown, adding a component to a signal-flow model, such as adding a damper in the EPAS case study, can cause the causality to change, which in turn implies that parts of the model needs to be re-modeled. From this point of view, the models are neither flexible nor extensible.

- **The signal-flow model is consistent with other models of computation in Ptolemy II**

This can be extended to claim that signal-flow modeling (i.e. causal modeling) is more straight-forward for deriving the behavior, as compared to constraint models. In section 4.3.5, we have developed a behavioral transformation from Simulink to SysML activity diagrams, and in section 5.3 we elaborate various methods to derive the behavior of physical modeling models.

- **The signal-flow model is compatible with the conservation law model**

One can combine acausal and causal modeling in the same model, so this is true.

5.4.2 Discussion

To conclude, here are some opinions from the author:

- Given a physical system, physical modeling should be used. One reason is that the cause-effect derivation is abstracted away from the user, and model rewriting and tearing can be used to formulate the problem to avoid algebraic loops. Another reason is that there are many models available, e.g. from the Modelica Standard Library [57], so the modeler should consider reusing available models, rather than modeling from scratch. For example, large parts of the NFCCPP model (Paper B) could be implemented directly from the Modelica Standard Library.
- The off-the-shelf solvers included in Dymola or Simulink include many integration algorithms with advanced zero-crossing mechanisms. Open source alternatives such as Scicos and Ptolemy II are also advanced and stable enough to use. Writing your own solvers is error-prone and something that should be avoided.
- The bond-graph approach is good for teaching, to learn about causality and the coupling of the conjugate variables. As a practical modeling tool, bond graphs do not scale well, especially when parameters are dependent on other variables than effort and flow. Hybrid systems and non-linear systems (when there is no direct coupling between effort and flow) are not directly covered very well by the bond-graph method either.

Chapter 6

Conclusions and Future Work

6.1 Research questions review

To conclude the results of this research, the research questions, formulated in section 1.5, are reviewed:

- **How can one create a component-based simulation environment, using MATLAB/Simulink?**

The answer to this question is the NFCCPP model, where such a simulation environment has been created. The main issue was to define proper generic interfaces between components. The sub question was "How should generic interfaces be designed in different physical domains, especially for fluid systems?", and this question is answered in Paper B with more detail in [83].

- **How can one relate current modeling and simulation environments for physical systems to the EAST-ADL2 modeling language? How can one relate Modelica and Simulink and SysML/UML?**

Many different alternatives of relating SysML with Modelica and Simulink have been investigated and evaluated. We have developed our own mappings of Modelica to SysML (Paper C), and from Simulink to UML (Paper D). Related approaches of Modelica and SysML integration are presented in section 4.3 on page 53 and related approaches for integrating Simulink and UML/SysML are presented in Paper D.

- **What are suitable abstraction levels of a physical system, and how can they be related?**

In two case studies, some of the difficulties in choosing abstraction level are highlighted. In the compressor case presented in section 4.1 on page 47, Simulink was an inappropriate abstraction; using an object-oriented language like C++ could have

been more convenient. For the fuel-cell environment, described in section 4.2 on page 50 and Paper B, Simulink maybe was at a too low realization level; it would have been more convenient to model using e.g. Modelica.

- **How can the behavior of continuous-time system be specified?**

An attempt is made to describe the modeling of physical systems using five realization levels. The approach, described in section 5.2 on page 62, is inspired by abstraction levels used in embedded systems.

6.2 Future work

The work for model-based design will continue with improved versions of modeling languages and tools. Although one may find flaws, and missing pieces in SysML today, it is a standard that many tool-vendor and companies are willing to support, and develop further. Sanford Friedenthal, one of the key persons behind SysML, admits that the standard is incomplete in some aspects, but wants to make a comparison to the more mature CAD-systems for geometrical design of mechanical products, which were also immature and incomplete not so long ago, but are now widespread [28]. Another observation is that CAD has not only improved the development process, it has radically changed it, and as a result both cost and time-to-market has been reduced [34].

6.2.1 Integration of Simscape and Modelica

A comparison between the Simscape Language and Modelica is of course inevitable.

- Simscape hides away component definitions for the Simscape foundation library in .p-files, which is MATLAB's format for code that can be read but not modified. This can be compared with the large Modelica Standard Library that is available. Only a handful of Simscape language example files are provided in the Simscape package, making it difficult to develop new components.

- Section 5.3 shows how the Modelica language generates many redundant variables and equations, which are later removed by the solver for the BLT[1] form. In Simscape it is not shown how many equations that are generated, but most probably it is fewer, given the documentation on how the equations are set up.

- Equations have well-defined semantics, although the syntax varies between languages. The strict component model of Modelica should make it possible to convert Modelica models into Simscape blocks. In the same way, exporting

[1] Block Lower Triangular, see [30] for more details.

Simscape blocks to Modelica should be possible, maybe by defining new connectors for various across-through variables. However, since most Simscape libraries are hidden in .p-files, there are today not too many components or models to convert to Modelica.

6.2.2 Further integration of SysML and Modelica

One conclusion from this thesis is the recommendation to separate acausal modeling from physical modeling. In Modelica, acausal connectors enable physical modeling, but in the case of SysML, the parametric diagram enables acausal modeling, but not physical modeling (see section 3.6). Extending SysML with a physical connector and physical ports would enable physical modeling, and a possibility to convert SysML blocks to Modelica or Simscape. These physical connectors and ports could have across-through variables declared, as discussed in section 5.1. There are ongoing activities on how to relate the languages, from both the Modelica and SysML communities.

6.2.3 Higher realization levels

When the physical modeling level is established, the follow-up question is what a higher abstraction level would mean, and look like. In the Sen/Vangheluwe approach [80] (described in section 5.2.6) the *Real World Visual Modeling Formalism* is set higher, but it is pointed out that there are no fixed rules to transform it into a physical modeling representation, and in general human input will be required for this.

One forecast is that tools will perform more engineering work, such as symbolic manipulation, unit check, sanity check, and be more user friendly. Another vision is that tools will be able to generate solution concepts from system constraints, or at least choose between different concepts.

For embedded systems, the vision is to synthesize the lower abstraction levels from higher, or to find suitable platforms (as defined by [76]) to work on. Here, embedded systems have a high potential, since the end product is at the lowest level of abstraction, as compared with simulation models, which per definition are models of the product.

6.3 Validity of research results

- **Two simulation models have been developed: one of a twin-screw compressor and the other of a fuel-cell simulation environment.**

As mentioned earlier, the twin-screw compressor is used in this thesis as a case study in exploring problems when performing modeling. The individual component models of the fuel-cell simulation environment have been validated for industrial use.

- **A method of protecting the intellectual property (I.P.) of simulation components has been developed.**

A security mechanism is never completely secure, it can only be secure enough. We have tried to consider many different aspects, including clamping, which is often overseen.

- **A survey of modeling languages for continuous-time systems has been carried out. An example model of an electric power assisted steering system has been implemented in these languages.**

The EPAS models presented in Chapter 3 have been tested in all the respective tools[2], to verify that they generate similar results. No tools were available for bond graphs, so this model is not verified. For SysML, there is always a possibility that the specifications has been misinterpreted, or used in a non-intended way.

- **Transformations between SysML and Simulink as well as between SysML and Modelica have been investigated.**

The transformations are based on thorough studies of SysML, Simulink and Modelica. The transformations have only been investigated for the behaviors and aspects covered in the thesis. Experiments have been carried out using small examples. As described in Paper D, a prototype implementation using ATL has been developed for the Simulink to SysML transformation.

- **Based on experience from the case studies, methodological concerns when modeling continuous-time systems have been identified and elaborated.**

In Chapter 5, and also in section 2.2.3 these methodological concerns are presented. The realization levels are verified in the sense that they are compared to related work and evaluated by applying them to studied modeling languages.

6.4 Concluding remarks

To enable model-based development of mechatronics, there is a need for languages that capture the important aspects of the system at the right level of detail.

This thesis has made an inventory of tools and languages that are available today, and evaluated how they perform modeling of continuous-time systems. The potential of model transformations has been shown in the form of transformation between SysML and Simulink and SysML and Modelica. Today there is trend of physical modeling, with many new languages emerging. Examples include Modelica

[2]MATLAB R2008b, Dymola 5.3b, Ptolemy II 7.0.1.

3.0, the Simscape language, MapleSim, SystemC-AMS and VHDL-AMS. Comparing Figure 3.6 on page 35, Figure 3.8 on page 38 and Figure 3.13 on page 44, the syntax of the languages are similar, so model transformations between the languages, or maybe from another source should be possible. This source could be modeled in an architectural description language, such as EAST-ADL2, or SysML. These languages should then be equipped with constructs so that this could be made easily.

The two case studies on modeling and simulation have shown that the signal-flow Simulink abstraction level has many drawbacks when modeling physical systems, with respect to e.g. modularity (NFCCPP case) and expressiveness (twin-screw compressor case). A major argument for why the signal-flow model is used today is that it is easy to transform into computations. If equation-based models were equally easy to transform they would be used to an even higher extent. In section 5.3, we show that the Modelica way of handling networked system may not be ideal. In addition, results from the Modelica community (section 5.3.3) point to a new language, with transformations defined directly in the language.

Bibliography

[1] 20-SIM WEBSITE. http://www.20sim.com, 2009-05-01.

[2] AADL WEBSITE. http://www.aadl.info, 2009-05-01.

[3] ATESST AND ATESST2 WEBSITE. http://www.atesst.org, 2009.

[4] ATL PROJECT WEBSITE. www.eclipse.org/m2m/atl/, 2008.

[5] AUSLANDER, D. What is mechatronics? *IEEE/ASME Transactions on Mechatronics 1*, 1 (Mar. 1996), 5–9.

[6] AUTOSAR DEVELOPMENT PARTNERSHIP. http://www.autosar.org, 2008-02-21.

[7] AXELSSON, J. Real-world modeling in UML. In *In Proc. 13th International Conference on Software and Systems Engineering and their Applications* (2000).

[8] BLESSING, L. T. Comparison of design models proposed in prescriptive literature. In *Proceedings of the COST A3/COST A4 International Research Workshop on "The role of design in the shaping of technology"* (1996), J. Perrin and D. Vinck, Eds., vol. 5 of *Social Sciences Series*, pp. 187–212.

[9] BOCK, C. UML 2 activity model support for systems engineering functional flow diagrams. *Systems Engineering 6*, 4 (2003), 249–265.

[10] BOCK, C. SysML and UML 2 support for activity modeling. *Systems Engineering 9*, 2 (2006), 160.

[11] BORUTZKY, W. Bond graph modelling and simulation of multidisciplinary systems - an introduction. *Simulation Modelling Practice and Theory 17*, 1 (2009), 3 – 21. Bond Graph Modelling.

[12] BROMAN, D., AND FRITZSON, P. Higher-order acausal models. In *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools* (Paphos, Cyprus, 2008), LIU Electronic Press, pp. 59–69.

[13] Brooks, C., Lee, E. A., Liu, X., Neuendorffer, S., Zhao, Y., and Zheng, H. Heterogeneous concurrent modeling and design in Java (volume 1: Introduction to Ptolemy II). Tech. Rep. UCB/EECS-2008-28, EECS Department, University of California, Berkeley, Apr. 2008.

[14] Brooks, C., Lee, E. A., Liu, X., Neuendorffer, S., Zhao, Y., and Zheng, H. Heterogeneous concurrent modeling and design in Java (volume 3: Ptolemy II domains). Tech. Rep. UCB/EECS-2008-37, EECS Department, University of California, Berkeley, Apr. 2008.

[15] Burns, A., Ed. *ARTIST Survey of Programming Languages*. ARTIST Network of Excellence on Embedded Systems Design, 2008.

[16] Burton, A. W. Innovation drivers for electric power-assisted steering. *Control Systems Magazine, IEEE 23*, 6 (2003), 30–39.

[17] Carloni, L. P., Passerone, R., Pinto, A., and Sangiovanni-Vincentelli, A. L. Languages and tools for hybrid systems design. *Foundations and Trends in Electronic Design Automation 1*, 1/2 (2006), 1–193.

[18] Cellier, F., and Nebot, A. The Modelica bond graph library. In *Proc. 4th International Modelica Conference* (Hamburg, Germany, 2005), vol. 1, pp. 57–65.

[19] Cervin, A., Ohlin, M., and Henriksson, D. Simulation of networked control systems using TrueTime. In *Proc. 3rd International Workshop on Networked Control Systems: Tolerant to Faults* (Nancy, France, June 2007). Invited talk.

[20] Cuenot, P., Chen, D., Gérard, S., Lönn, H., Reiser, M.-O., Servat, D., Tavakoli Kolagari, R., Sjöstedt, C.-J., Törngren, M., and Weber, M. Managing complexity of automotive electronics using the EAST-ADL. In *ICECCS* (2007), IEEE Computer Society, pp. 353–358.

[21] Daenzer, W. F., and Huber, F. *Systems engineering - Methoden und Praxis*, eighth ed. Verlag Industrielle Organisation, Zürich, 1994.

[22] Dynasim AB. `http://www.dynasim.se`, 2008-09-25.

[23] Eclipse Modeling Framework Procejt (EMF). `www.eclipse.org/modeling/emf/`, 2007.

[24] Elmqvist, H., and Otter, M. Methods for tearing systems of equations in object-oriented modeling. In *Proceedings of the European Simulation Muliticonference (ESM'94)* (Barcelona, Spain, June 1995), SCS, Society for Computer Simulation, pp. 326–334.

[25] Emission test cycles. http://www.dieselnet.com/standards/cycles/, 2009-04-25.

[26] Eriksson, B., and Sundström, A. Många miljöbilar ett hot mot svensk trafiksäkerhet. *Dagens Nyheter*. Published 2007-06-18.

[27] ETAS web. http://www.etas.com, 2009-04-27.

[28] Friedenthal, S. SysML: Current challenges and future need. Speech at ICOSE seminar, 2008-11-11.

[29] Friedenthal, S., Moore, A., and Steiner, R. *A Practical Guide to SysML*. Morgan Kaufmann, 2008.

[30] Fritzson, P. *Principles of Object-oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.

[31] Fritzson, P., and Engelson, V. Modelica - a unified object-oriented language for system modeling and simulation. *LECTURE NOTES IN COMPUTER SCIENCE* (1998), 67–90.

[32] Gray, J., Tolvanen, J., Kelly, S., Gokhale, A., Neema, S., and Sprinkle, J. Domain-specific modeling. In *Handbook of Dynamic System Modeling*, P. A. Fishwick, Ed. Chapman & Hall/CRC, 2007, ch. 7.

[33] Grimheden, M. *Mechatronics Engineering Education*. PhD thesis, Dept. of Machine Design, The Royal Institute of Technology, Stockholm, Sweden, Nov. 2006.

[34] Hammer, M., and Champy, J. *Reengineering the Corporation : a manifesto for business revolution*, revised ed. Brealey, London, 1995.

[35] Harashima, F., Tomizuka, M., and Fukuda, T. Mechatronics - "what is it, why, and how". *IEEE/ASME Transactions on Mechatronics 1* (1996), 1–4.

[36] Heinecke, H., Bielefeld, J., Schnelle, K., Maldener, N., Fennel, H., Weis, O., Weber, T., Ruh, J., Lundh, L., Sandén, T., et al. AUTOSAR–current results and preparations for exploitation. In *7th EUROFORUM conference 'Software in the vehicle'* (May 2006).

[37] Hirtz, J., Stone, R. B., McAdams, D. A., Szykman, S., and Wood, K. L. A functional basis for engineering design: Reconciling and evolving previous efforts. In *Research in Engineering Design* (Mar. 2002), vol. 13, Springer-Verlag, pp. 65–82.

[38] Home of Bond Graphs - The System Modeling World. http://www.bondgraph.info, 2009-05-01.

[39] IEEE DASC 1076.1 Working Group. VHDL-A design objective document, version 2.3, Sept. 1995.

[40] IEEE-SA Standards Board. IEEE recommended practice for architectural description of software-intensive systems, Sept. 2000. IEEE Standard 1471-2000.

[41] Isermann, R. *Mechatronic Systems : Fundamentals*. Springer, London, 2005.

[42] Jantsch, A. *Modeling Embedded Systems and SoC's*. Morgan Kaufmann, San Diego, 2004.

[43] Johnson, T. A. Integrating models and simulations of continuous dynamic system behavior into SysML. Master's thesis, Georgia Institute of Technololgy, May 2008.

[44] Karnopp, D. C., Margolis, D. L., and Rosenberg, R. C. *System dynamics : a unified approach*, second ed. Wiley, New York, 1990.

[45] Karnopp, D. C., Margolis, D. L., and Rosenberg, R. C. *System Dynamics: Modeling and Simulation of Mechatronic Systems*, third ed. John Wiley and Sons, New York, 2000.

[46] Kauder, K., Janicki, M., Rohe, A., Kliem, B., and Temming, J. Thermodynamic simulation of rotary displacement machines. *VDI BERICHTE 1715* (2002), 1–16.

[47] Krus, P. Modeling of mechanical systems using rigid bodies and transmission line joints. *Journal of Dynamic Systems, Measurement, and Control 121*, 4 (1999), 606–611.

[48] Larses, O. *Architecting and Modeling Automotive Embedded Systems*. PhD thesis, Dept. of Machine Design, The Royal Institute of Technology, Stockholm, Sweden, 2005.

[49] Larses, O., Sjöstedt, C.-J., Törngren, M., and Redell, O. Experiences from model supported configuration management and production of automotive embedded software. In *SAE Word Congress & Exhibition* (Apr 2007), no. 2007-01-0510, SAE.

[50] Ljung, L., and Glad, T. *Modellbygge och simulering*. Studentlitteratur, Lund, 1991.

[51] Lynn, C. L. Basic elements of mathematical modeling. In *Handbook of Dynamic System Modeling*, P. A. Fishwick, Ed. Chapman & Hall/CRC, 2007, ch. 5.

[52] Maplesoft web. `http://www.maplesoft.com`, 2009-04-16.

[53] MathWorks. Simulink user manual R2008b, 2008.

[54] Mechanical Simulation Corporation web. http://www.carsim.com, 2009-04-27.

[55] Miller, A. T., and Allen, P. K. Graspit! a versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine 11*, 4 (Dec. 2004), 110–122.

[56] Modelica Association. http://www.modelica.org, 2008-11-20.

[57] Modelica Association. Modelica standard library 3.0, Feb. 2008. available at www.modelica.org.

[58] Moler, C. The growth of MATLAB and The MathWorks over two decades. Reprinted from The MathWorks News and Notes, Jan. 2006.

[59] MSC Software web. http://www.mscsoftware.com, 2009-04-27.

[60] Myrup Andreasen, Mogensand Hein, L. *Integrated Product Development*. IFS, Bedford, 1987.

[61] Navet, N., Song, Y., Simonot-Lion, F., and Wilwert, C. Trends in automotive communication systems. *Proceedings of the IEEE 93*, 6 (2005), 1204–1223.

[62] OMG. OMG systems modeling language specification. Electronic, 2007.

[63] OMG. Meta Object Facility (MOF) Specification 2.0, 2008.

[64] Open SystemC Initiative. SystemC AMS extensions draft 1, Dec. 2008.

[65] Osborne, A. *Osborne 4 & 8-Bit Microprocessor Handbook*. Osborne/McGraw-Hill, Berkeley, 1981.

[66] Pahl, G., Beitz, W., and Wallace, K. *Engineering Design : A Systematic Approach*, second ed. Springer, Berlin, 1996.

[67] Papurus UML website. http://www.papyrusuml.org, 2009-04-29.

[68] Parmar, M., and Hung, J. Y. A sensorless optimal control system for an automotive electric power assist steering system. *IEEE Transactions on Industrial Electronics 51*, 2 (2004), 290–298.

[69] Peak, R. S., Burkhart, R. M., Friedenthal, S. A., Wilson, M. W., Bajaj, M., and Kim, I. Simulation-based design using SysML part 1: A parametrics primer. In *INCOSE International Symposium* (2007).

[70] Persson, J.-G. Heat exchange in liquid injected compressors. Tech. rep., Department of Fluid Technology, Royal Instutute of Technology, Stockholm, Sweden, 1986.

[71] Persson, J.-G., Chen, D.-J., and Sjöstedt, C.-J. Automotive fuel cell system simulation, component and compressor modelling. In *Schraubenmaschinen* (Sept. 2006), VDI Verlag GmbH, pp. 217–235.

[72] Pop, A., Akhviediani, D., and Fritzson, P. Towards Unified System Modeling with the ModelicaML UML Profile. In *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools, Berlin, Germany* (July 2007), P. Fritzson, F. Cellier, C. Nytsch-Geusen, D. Broman, and M. Cebulla, Eds., pp. 13–24.

[73] Roos, F. *Towards a Methodology for Integrated Design of Mechatronic Servo Systems*. PhD thesis, Dept. of Machine Design, The Royal Institute of Technology, Stockholm, Sweden, Jun 2007.

[74] Rust, R. T., Thompson, D. V., and Hamilton, R. W. Defeating feature fatigue. *Harvard business review 84*, 2 (Feb. 2006), 98–107.

[75] Süss, J. G., Fritzson, P., and Pop, A. The impreciseness of UML and implications for ModelicaML. In *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools.* (Aug. 2008), P. Fritzson, F. Cellier, and D. Broman, Eds.

[76] Sangiovanni-Vincentelli, A. Defining platform-based design. *EEDesign of EETimes* (Feb. 2002).

[77] Schilling, M. Toward a general modular systems theory and its application to interfirm product modularity. *Academy of Management Review* (2000), 312–334.

[78] Schneier, B. *Applied Cryptography*, second ed. Wiley, New York, 1996.

[79] Scicos website. `http://www.scicos.org`, 2009-04-22.

[80] Sen, S., and Vangheluwe, H. Multi-domain physical system modeling and control based on meta-modeling and graph rewriting. In *Intelligent Control, 2006 IEEE International Symposium on* (Oct. 2006), pp. 69–75.

[81] Sjöstedt, C.-J. Modelling of displacement compressors using MATLAB/Simulink software. In *Product Development in Changing Environment* (2004), T. Lehtonen, A. Pulkkinen, and A. Riitahuhta, Eds., Tampere University of Technology Product Development Laboratory, pp. 192–200.

[82] Sjöstedt, C.-J. *On the Modular Modelling for Dynamical Simulation with Application to Fluid Systems*. Licentiate thesis, Dept. of Machine Design, The Royal Institute of Technology, Stockholm, Sweden, Dec. 2005.

[83] Sjöstedt, C.-J., and Persson, J.-G. The design of modular dynamical fluid simulation systems. In *Proceedings from OST 05 conference* (Oct. 2005).

[84] Spolsky, J. *Joel on Software*. APress, Berkeley, 2004.

[85] Strang, G. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, Wellesley, Mass., 1986.

[86] Svenska Rotor Maskiner AB website. `http://www.rotor.se`, 2009-02-16.

[87] SysML Forum FAQ. `http://www.sysmlforum.com/FAQ.htm`, 2009-04-29.

[88] The MathWorks web. `http://www.mathworks.com`, 2008-11-12.

[89] Tomizuka, M. Mechatronics: from the 20th to 21st century. *Control Engineering Practice 10*, 8 (2002), 877 – 886.

[90] Tongue, B. Two brains, one car - actively controlled steering. *IEEE Control Systems Magazine 25*, 5 (Oct. 2005), 14–17.

[91] Törngren, M., Henriksson, D., Årzén, K.-E., Cervin, A., and Hanzalek, Z. Tools supporting the co-design of control systems and their real-time implementation; current status and future directions. In *2006 IEEE International Symposium on Computer-Aided Control Systems Design* (Munich, Germany, Oct. 2006).

[92] Ulrich, Karl T.and Eppinger, S. D. *Product Design and Development*, fourth ed. McGraw-Hill/Irwin, 2008.

[93] VDI, The Association of German Engineers. VDI 2206: Design methodology for mechatronic systems, June 2004.

[94] Vlach, J., and Singhal, K. *Computer Methods for Circuit Analysis and Design*. Springer, 1983.

[95] Zimmer, D., and Cellier, F. The Modelica multi-bond graph library. *Simulation News Europe 17*, 3/4 (2007), 5–13.