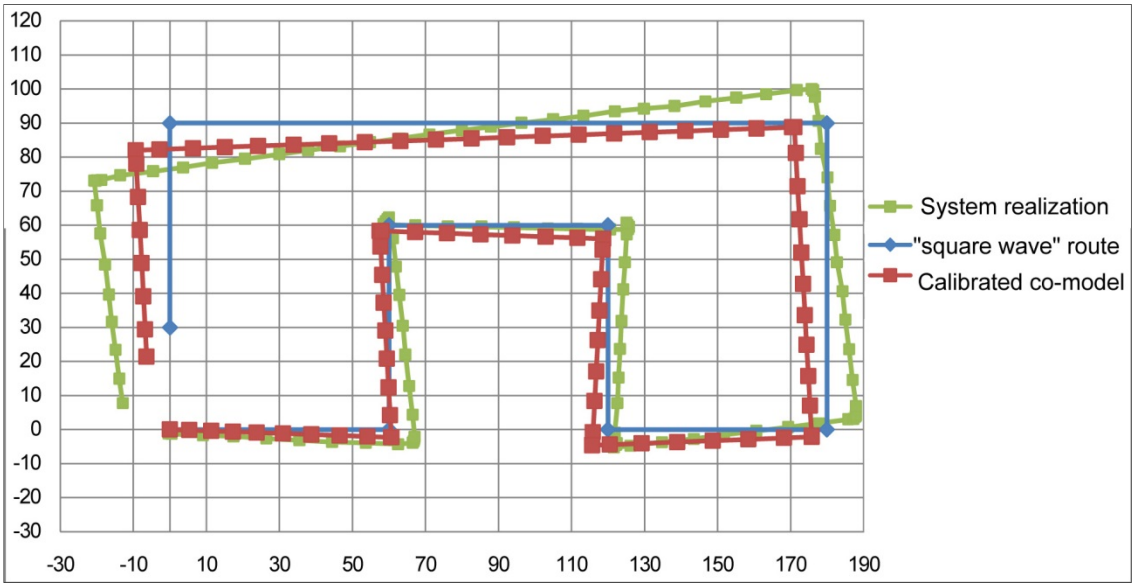
EVALUATION OF DEVELOPMENT PROCESS AND METHODOLOGY FOR CO-MODELS

Electrical and Computer Engineering
Technical Report ECE-TR-12



AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

DATA SHEET

Title: Evaluation of Development Process and Methodology for Co-Models.

Subtitle: Electrical and Computer Engineering
Series title and no.: Technical report ECE-TR-12

Author: Peter Würtz Vinther Jørgensen
Department of Engineering – Electrical and Computer Engineering, Aarhus University

Internet version: The report is available in electronic format (pdf) at the Department of Engineering website http://www.eng.au.dk.

Publisher: Aarhus University©
URL: http://www.eng.au.dk

Year of publication: 2012, Pages: 103
Editing completed: April 2012

Abstract: An embedded control system often requires a tight association between computational and physical system components. In such cases, embedded system development is difficult, as it requires the collaboration among stakeholders with different backgrounds (software engineers, mechanical engineers, managers etc.). With the constant increase in design complexity, caused by advances in implementation technologies, new ways of approaching embedded system development are needed.

This thesis presents an evaluation of a tool-oriented development process and methodology, supporting embedded system development. The philosophy of the development process and methodology, is that design complexity can be managed through collaborative work and multi-disciplinary modeling. To obtain input for the evaluation work, the development process is applied during a case study, involving the development of a route following robot and a model of this. To demonstrate the value of this model, it is simulated to predict route completion times for the physical robot.

The evaluation work identifies possibilities and challenges of the development process and methodology, with respect to traditional physical prototyping. This will support developers in choosing the most optimal way of approaching development. In addition to this, suggestions for extensions to the methodology are provided. These intend to increase the value the development process and methodology may bring the development work.

Keywords: development processes, methodologies, co-models, discrete event, continuous time, DESTECS, SysML, iterative development, embedded systems.

Supervisor: Peter Gorm Larsen and Sune Wolff.

Please cite as: Peter Würtz Vinther Jørgensen, 2013. Evaluation of Development Process and Methodology for Co-Models. Department of Engineering, Aarhus University. Denmark. 103 pp. - Technical report ECE-TR-12

Cover image: Peter Würtz Vinther Jørgensen.

ISSN: 2245-2087

EVALUATION OF DEVELOPMENT PROCESS AND METHODOLOGY FOR CO-MODELS

Peter Würtz Vinther Jørgensen
Aarhus University, Department of Engineering

Abstract

An embedded control system often requires a tight association between computational and physical system components. In such cases, embedded system development is difficult, as it requires the collaboration among stakeholders with different backgrounds (software engineers, mechanical engineers, managers etc.). With the constant increase in design complexity, caused by advances in implementation technologies, new ways of approaching embedded system development are needed. This thesis presents an evaluation of a tool-oriented development process and methodology, supporting embedded system development. The philosophy of the development process and methodology, is that design complexity can be managed through collaborative work and multi-disciplinary modeling. To obtain input for the evaluation work, the development process is applied during a case study, involving the development of a route following robot and a model of this. To demonstrate the value of this model, it is simulated to predict route completion times for the physical robot.

The evaluation work identifies possibilities and challenges of the development process and methodology, with respect to traditional physical prototyping. This will support developers in choosing the most optimal way of approaching development. In addition to this, suggestions for extensions to the methodology are provided. These intend to increase the value the development process and methodology may bring the development work.

Acknowledgements

I would like to thank my academic supervisor, Peter Gorm Larsen, for advice and attention during this thesis and Master's degree. I would also like to thank my co-supervisor, Sune Wolff, for valuable feedback on the content and structure of this thesis work.

Additional thanks go to Martin Peter Christiansen, who provided guidance for the software tools used during this thesis, and participated in some of the early development activities.

Finally, I would like to thank my friends and family for their patience and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the background and motivation underlying the work of this thesis. To set the context of the subsequent chapters, the thesis goals are presented along with the work to be carried out, and the approach taken. Chapter 2 continues with an introduction to the theory and applied technologies, necessary for understanding the work following. Finally, the thesis goals are revisited for discussion in chapter 7.

1.1. Overview

An embedded control system consists of a digital *controller* running software responsible for making control decisions, based on stimuli from external sources such as users and the physical *environment*. Instructed by the control decisions, the controller responds to the environment by interacting with for instance mechanical or electrical system components. A well-known example of an embedded system is the cruise controller in a car. The controller maintains a steady speed by regulating the throttle actuator, based on inputs from sensors monitoring the current velocity and acceleration of the car. For the cruise control to provide a pleasant ride, the control decisions need to take the association between mass, force, velocity and acceleration into account. Therefore, the technical knowledge required to successfully develop such a system, spans more engineering disciplines. This causes a high degree of design complexity, and requires the collaboration among various types of stakeholders with different skills. Traditionally the inherent engineering disciplines have been treated separately by different engineering teams. As a consequence, consistency checks get postponed to late stages of the development, where the cost of corrections are high.
This chapter provides the background and motivation of this thesis in sections 1.2 and 1.3. This is followed by a description of the thesis goals and the case study supporting their achievement in sections 1.4 and 1.5. Finally, this chapter ends with a reading guide and a presentation of the overall chapter structure of this thesis in sections 1.6 and 1.7.

1.2. Background

One way of addressing the design complexity of embedded systems, is to use abstract multi-disciplinary models, for reasoning about the desired system-level properties. What motivates the

use of models in particular, is their ability to focus on the important challenges, while neglecting things of less significance, by applying a higher level of abstraction. As an example, investigation of different controller behaviors is only further complicated, if the underlying hardware platform needs to be taken into account. By omitting these details, the full attention can be turned to the controller, which provides a stronger basis for the analysis work. Hence evidence or insight obtained from the model analysis, supports the design decision activities already at the system-level. This is a major advantage, as it reduces the amount of expensive rework, often caused by the feedback of late design phases.

1.2.1 Multi-disciplinary challenges

By treating development as a collaborative task with the participation of the different types of stakeholders, it is possible to minimize the chances of misunderstandings. These misunderstandings are often caused by the fact that engineers do not fully understand the impact of changes across the different *domains*. In this context, a domain is encapsulating one or more of the represented engineering disciplines. As an example, consider an embedded system where software is responsible for controlling the speed of a motor based on sensory inputs. If a hardware failure leading to wrong sensor outputs is not dealt with in the control algorithm, it can lead to controller behavior harmful to the physical components of the system. For instance caused by the instruction to perform a sudden increase in motor speed. For these situations an appropriate level of knowledge spanning multiple domains is needed, in order to achieve an overall better system solution. Holistic and heterogeneous views may serve useful for this task [Henzinger&07], as they facilitate knowledge sharing among the different domain representatives.

1.2.2 Managing complexity using collaborative modeling

In collaborative modeling, or *co-modeling*, the behavioral logic of the controller is described using a Discrete Event (DE) formalism, while the physical concepts of the environment are based on Continuous Time (CT) models, expressed as differential equations. The combined model will subsequently be referred to as a *co-model*, and the simulation of it as a *co-simulation*. A *system* is either a co-model or a *realization* of an embedded system. In this context, a system realization may refer to either a physical system prototype or a physical copy of the final system. A system includes everything which needs to be developed, but also the definitions (e.g. units of measure) and the assumptions made (e.g. constant temperature).

The hypothesis is that co-modeling will help engineers in the process of obtaining the sufficient insight needed, for contributing to a globally optimal solution. The intention is to raise awareness of the cross-disciplinary consequences of introducing changes in a single domain. This is further supported by the early feedback obtained, by simulating the multi-disciplinary models. These models enable visualization of design choices, without relying on a system realization. If such multi-disciplinary models are constructed with sufficient fidelity, they enable prediction of properties concerning the system realization, which is the focus of section 5.4. This potentially reduces the number of physical machine-built iterations, due to the opportunity of doing virtual system exploration [Heemels&07].

1.3. Motivation

The need for tools supporting co-modeling and co-simulation of multi-disciplinary systems, is well-recognized by the research community, and several noteworthy contributions already exist (see section 2.7). Still, the challenge of *how* and *when* to apply these tools to bring the most value to the development work, remains a topic lacking further research.
This has motivated a proposal for a *development process* [Wolff12] addressing the multi-disciplinary concerns of embedded systems using co-modeling, while adhering to a four phased iterative spiral structure [Boehm88]. Wolff, who suggested this development process, was lecturing one of the classes in the *Modeling of Mission Critical Systems* course prior to the writing of this thesis. During this course, he proposed a project involving the evaluation of his work. The thesis author's personal interest in formal modeling, and the novelty of development processes using co-modeling, are the main reasons for this thesis choice.
The development process is described in more detail in chapter 3, and a brief introduction to the four phases will suffice for now. In the first phase, *model purpose and requirements*, activities are performed to establish a common understanding and objective among the represented stakeholders. Next, the system is broken down into more manageable parts in the *system decomposition* phase using multi-disciplinary system descriptions. This is followed by the co-modeling activities in the *system modeling* phase. In the final phase *validation and verification*, the co-model is analyzed and validated using 3D animations and domain specific tools. Furthermore, the fidelity of the co-model is increased, so it can be used for making predictions concerning the system realization.
The development process is supported by a *methodology*, providing guidance to the development team during the suggested activities. More specifically, the methodology provides inputs on *how* and *when* to apply the suggested modeling notations. It describes what steps to take, in order to obtain the necessary evidence or insight needed, before transitioning to the realization phase. The methodology is captured in the form of step-wise *methodological guidelines*, expressing the essence of the complete methodology. The purpose of these guidelines is to enforce sufficient structure to the development work, to ensure that the right questions are being asked in a timely manner, to avoid substantial rework. A *process* is regarded as being a sequence of activities resulting in a number of coherent artifacts such as system descriptions, models and a realization of the system.
Throughout this thesis, performing the activities of the development process, while following the methodology (or guidelines), will be referred to as *"applying the development process"*. In addition to this, the phrasing *"applying the methodology"* is used for emphasizing that the focus is on the advice given by the methodology, rather than the activities being performed. The same holds for the phrasing *"applying the methodological guidelines"*.

1.4. Thesis goals, scope and approach

The goals of this thesis are:

G1: To evaluate the development process and methodology [Wolff12] for co-models suggested by Wolff, through case study development. The result of this work will provide inputs for potential changes to the development process and methodology.

G2: To extend and suggest new ways of using the current tools and techniques for collaborative modeling, supporting the application of the development process.

G3: To improve personal skills concerning researching a scientific topic and disseminating the findings.

1.4.1 Evaluation categories

To determine the overall value of the development process and the supporting methodology, a description of the categories forming the basis of the evaluation is given below. These categories will be re-addressed in chapter 6, where the final evaluation is performed.

Advantage: When does it make sense to apply the development process? And what value may it bring the development work, which is impossible or difficult to obtain using traditional approaches?

Relevance: The point of this category is to investigate, whether the activities suggested by the development process apply to different types of embedded systems. As an example, the failure of *mission-critical* systems will lead to loss of human lives or huge economics costs. These systems demand extra attention during the early project stages to ensure a high degree of confidence in the design, before transitioning to the realization phase. This often requires a comprehensive test coverage and resilience to failure, which may be of less significance for other systems. Is the approach taken by the development process suited for both types of systems?

Effectiveness: To meet the demands of the highly competitive market, a development process must be supported by enabling tools and techniques, to increase the productivity of the development work. How well the development process exploits these enabling tools and techniques for co-modeling and co-simulation, is the concern of this final category.

As a final remark of this sub-section, it should be noted that these categories are conflicting, in that the effectiveness will influence the advantage and so forth.

1.4.2 Scope

The point of applying the development process is to gain sufficient experience to be able to perform an evaluation of the development process and methodology. Thus developing high-fidelity multi-disciplinary models and well-working system realizations during the case study, is out of the scope of this thesis. Instead the focus is set on trying to cover as many aspects of the development process as possible, within the time-frame of this thesis. This is done by applying the development process to a concrete case study, while taking other cases representing systems with different characteristics, into account as well.

1.4.3 Approach

The approach taken to achieve the thesis goals is a four-step iterative process, given in the description below. These steps should *not* be confused with the phases of the development process to be evaluated.

Analysis: Involves the acquisition of new knowledge for the writing of this thesis. During early iterations, this step was amounting for most of the time spent on the thesis work, as the learning of new tools and modeling techniques was needed.

Modeling: Comprises the modeling activities of the development process. The multi-disciplinary system descriptions and models are built with the purpose of performing system-level reasoning and co-simulation.

Realization: Involves the gradual realization of the case study, using the artifacts resulting from the activities of the previous step.

Evaluation: Combines the results of the literature study with the experiences gained from the co-modeling and realization activities, in order to evaluate the development process and methodology.

1.5. Case study

To obtain inputs for the evaluation of the development process and methodology, a part of this thesis concerns the development of the *Autonomous Robot System*. To broaden the perspective of the evaluation, other cases representing systems with different characteristics, will be taken into account, when discussing the development process and methodology. The hope is that this will increase the value of the resulting evaluation work, by appealing to a wider range of systems.

1.5.1 The Autonomous Robot System case study

The Autonomous Robot System is a small four-wheeled robot capable of following a known route of two-dimensional way points in the x-y plane. It performs this task only aided by sensors for measuring the distance covered, and the current orientation. In order to complete the route, the robot must visit all the way points in the correct order. Both a co-model and a realization of this system will be developed. The co-model will be used for predicting route completion times for the system realization.
This inherently multi-disciplinary system is suited for collaboration among engineers having different backgrounds (mechanical, software etc.). The case study faces the developers with the controller/environment partitioning challenge, which makes it a proper candidate for the application of the development process. The details of the case study work are covered in chapters 4 and 5.

1.5.2 The tool-chain used during the case study

The development process relies on software tools for co-modeling and co-simulation, but does not dictate the use of a particular tool-chain. More specifically, the development process and methodology should be general enough, to be applicable across different tool-chains capable of simulating the multi-disciplinary models. The Design Support and Tooling for Embedded Control Software (DESTECS) project [Broenink&10] focuses on developing tools and methods aiding the development of embedded systems. The approach taken is to combine the DE and CT models through co-simulation. The tool-chain provided by the DESTECS project is particularly important with respect to this thesis, as it is being used during the case study work, for reasons given in section 2.8. Insight into the terminology and tools of the DESTECS project is a prerequisite to be able to fully understand the work following, and therefore this will be treated at the necessary level of detail in chapter 2. The use of the DESTECS tool-chain required the learning of a new tool-chain and CT formalism, as the author only had experience using the supported DE formalism.

1.5.3 Development settings

In the attempt of creating a realistic development setting, the Autonomous Robot System has been developed by a software engineer (the author of this thesis) with little assistance of a robotics engineer. The development process activities have been carried out in the form of interactive white-board aided sessions. During these sessions, the two engineers were collaborating on describing the system, by following the suggestions of the methodological guidelines. That said, the software engineer has been responsible for modeling of both the controller and the environment. In case problems occurred or advice was needed, the robotics engineer would provide guidance.

1.5.4 Other DESTECS cases

Throughout this thesis, other examples will be referred to during the discussion of the development process and methodology. Descriptions of these systems are given below.

Flare Dispensing System: This system is presented in [Wolff12], and used as a case study by the author of the development process, for demonstrating the application of it. The system is capable of detecting incoming missiles, and responds by dispensing flares made of hot burning materials. The bright infrared image created by the flares, is used for distracting the infrared guided missiles away from the air-borne system in which the flare dispenser is installed.

Water Tank: Another example described in [Fitzgerald&10a] represents a model of a water tank, supplied by a continuous inwards flow of water. The tank is equipped with sensors for measuring the current water level in the tank, which must be kept within some minimum and maximum thresholds. A valve at the bottom of the tank can be opened and closed, to control the current water level. If the water level exceeds the maximum threshold, the valve will be opened to enable water flowing out of the tank. On the other hand, if the water falls below the minimum threshold, the valve is closed to stop the water flowing out of the tank.

Paper Pinch: This system was originally presented in [Heemels&07] and studied in relation to DESTECS in [Pierce&11]. A *paper pinch* consists of two rollers sitting one above the other. The bottom roller is motor driven to enable the transferring of motion to a piece of paper, whereas the top roller is spring loaded to produce a downwards pressure. This enables the passing of a piece of paper between the two rollers through friction. A real printer may consist of several paper pinches conveying the piece of paper during the manipulation process, before it reaches the output tray.

Line Followining Robot: A final example is the Line Following Robot described in [Pierce&12a]. This example should not be confused with the main case study of this thesis. The Line Following Robot is able to follow a line of known color, differentiated from the background color of the floor. Thus the line can be detected using light sensors for measuring the light and dark areas on the floor. The robot is equipped with two wheels, each controlled by a motor. These motors enable smooth directional control, when following the angled corners and steep curves exhibited by the line.

1.6. Reading guide

This section introduces the conventions used throughout this thesis.

Emphasis:
Words or terms of special relevance are written in *italic*, e.g. *environment*.

References:
References for written work appear in square brackets, e.g. [Larsen&10a]. "Larsen" refers to the surname of the main author, while "10" refers to the year of writing. The "a" indicates that the author is responsible for several publications within the year of writing. When more authors have been contributing to the work referenced, a "&" is placed in between the surname of the main author and the year of writing. Thus [Kruchten95] is a reference for the work of a single author.

Guidelines:
The methodological guidelines supporting the development process are introduced as shown below.

Guideline 1: Document all assumptions made of the system.

Keywords:
DESTECS related keywords are written in a **`boldface typewriter font`**, e.g. **`real`**. Model elements are written in a `plain typewriter font`, e.g. `SensorReal`.

VDM model listings:
VDM model listings are presented as shown below.

```
1 private coordinates : seq of Coordinate := [mk_Coordinate(0,0)];
2 inv coordinates <> [] and
3         coordinates(1) = mk_Coordinate(0.0,0.0) and
4         forall i in set {1,...,len coordinates -1} &
5                 coordinates(i) <> coordinates(i+1);
```

Quotations:
Quoted material are put within double quotation marks and written in *italic*, e.g. *"this is an example of a quotation"*.

Equations:
Equations are listed as shown below.

$$x = \int v\,dt + x_0 \tag{1.1}$$

1.7. Structure

This thesis consists of seven chapters and two appendices as visualized in figure 1.1. The solid arrows indicate the suggested orders of reading. The extensions suggested for the methodology in

chapter 6, can be understood without reading about the case study work, which explains the arrow from chapter 3 to chapter 6. Appendix material should be referred to as necessary, and thus the dashed arrow is used for indicating an association between chapter 6 and appendix A. Similarly, some of the case study related information has been moved to appendix B.

Chapter 2: Introduces the theory and technologies used for achieving the goal of evaluating the development process and methodology. A brief introduction to the System Modeling Language (SysML) is given. In addition to this, the DESTECS tool-chain used during the case study is presented and related to other relevant tool contributions.

Chapter 3: Presents the development process and methodology to be evaluated. These are related to other work contributions within the field of development processes and methodologies addressing multi-disciplinary concerns.

Chapter 4: Demonstrates the application of the methodological guidelines supporting the first three phases of the development process.

Chapter 5: Demonstrates the application of the methodological guidelines supporting the final phase of the development process.

Chapter 6: Provides suggestions for extensions to the methodology of the development process, based on the findings of the case study work and the literature study of this thesis. In addition to this, chapter 6 provides the final evaluation of the development process and methodology.

Chapter 7: Presents the achieved results, the future work and personal learning outcomes, resulting from the work of this thesis.

Appendix A: Presents the details of a co-modeling related tool extension developed during this thesis, only briefly covered in chapter 6.

Appendix B: Presents the details of the case study work omitted from chapters 4 and 5.

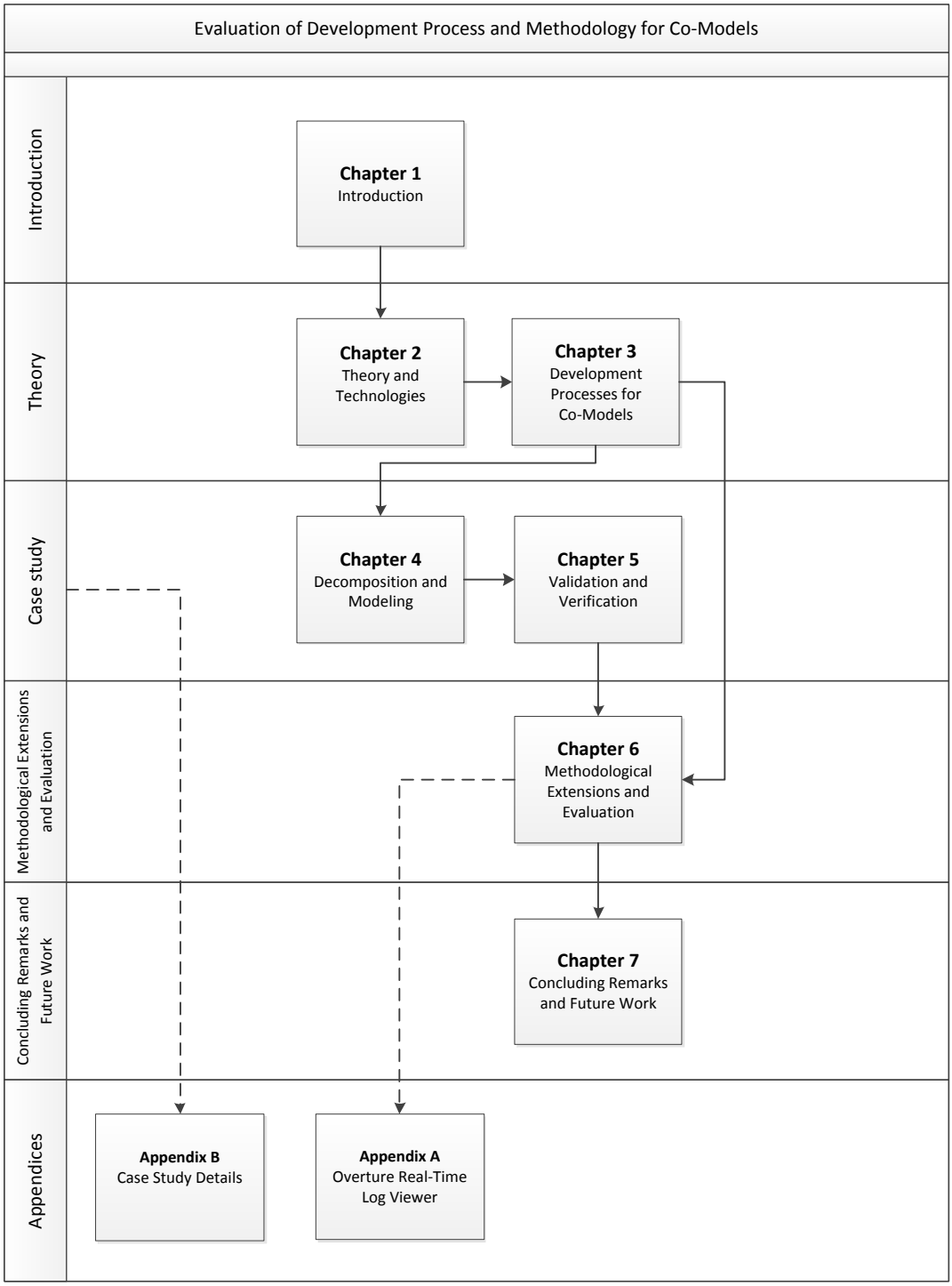


Figure 1.1: The structure of the thesis

Chapter 2

Theory and Technologies

In chapter 1 the goal of evaluating a development process and methodology using the Autonomous Robot System case study was presented. The development process and methodology are described in chapter 3, and related to other work contributions within the field of development processes and methodologies addressing multi-disciplinary concerns. The topic of this chapter is to introduce the theory and technologies, which will be necessary for understanding the development process, the methodology and the evaluation of them.

2.1. Introduction

To be able to understand the application of the development process, it is necessary to acquire some insight into the modeling notations used during the early development activities. In addition to this, knowledge about the tool-chain chosen for the case study work, is assumed throughout the subsequent chapters. Since the DESTECS tool-chain will be used during the case study work, this chapter will provide the sufficient insight into the corresponding terminology, tools and formalisms.
First, the System Modeling Language is described in section 2.2. This is followed by an introduction to the DESTECS terminology and formalisms belonging to the corresponding tool-chain in sections 2.3 and 2.4. Next, Automated Co-model Analysis and fault modeling are described in sections 2.5 and 2.6. This is followed by an introduction to other tool contributions relevant to DESTECS in section 2.7. Finally, it is discussed why DESTECS was chosen for the case study work in section 2.8.

2.2. The System Modeling Language

The activities of the development process to be evaluated, make use of holistic views for reasoning about system-level properties. Therefore, the development process rely on a notation commonly used by system engineers. Such a notation is found in the System Modeling Language (SysML) [Friedenthal08], which provides an extension to the Unified Modeling Language (UML) [Fowler&03]. SysML allows modeling constructs of different domains in the same system description. In embedded system development, this can be used for describing the interaction among software and physical components. The following description provides an overview of the

different SysML constructs.

Requirements: *Use case diagrams* support identification of the important *actors* and major functions of the system. System functionality is captured using *use cases*, which describe high-level interaction between a system actor, and the system under consideration. From these diagrams, the system requirements can be derived. Modeling of requirements is supported by the *requirements diagram*. The *verified by* relation makes it possible to associate requirements and test cases, in order to promote traceability. Similarly, system components may *satisfy* requirements.

Structure: SysML introduces the concept of a *block*, which constitutes the basic unit of structure in SysML. A block may represent any modular unit of the system such as a software class, a hardware component or even a function. By arranging these blocks in a *Block Definition Diagram* (BDD), the hierarchical structure of the system can be described. If a block is composed of other blocks (such as sub-systems), its part properties can be captured using the *composite association*. The internal structure of a block, can be specified using the *Internal Block Diagram* (IBD), while interaction among the internal parts is described using *ports*. *Flow ports* are typed and specify what flows in and out of a block (liquids, data etc.). *Standard ports* define interactions using required and provided services (e.g. operations), which is often used in software engineering.

Behavior: SysML offers three types of diagrams for describing behavior. *State charts* describe the states of a system, and the transitioning among them. *Sequence diagrams* describe message-based behavior among blocks. Finally, *activity diagrams* describe the work-flow of stepwise activities.

Constraints: *Constraint blocks* arranged in a BDD are used for capturing equations such as physical laws. Using these blocks, the *parametric diagrams* can be used for describing the constraints on system properties (performance, physical properties etc.).

2.3. The DESTECS terminology

DESTECS has introduced its own terminology, in order to establish a common vocabulary among the different engineering disciplines (mechanical, electrical etc.). The hope is that this will help minimize ambiguity, but also reduce the chances of misunderstandings among stakeholders. As an example, the term “parameter” may refer to a variable within the software engineering domain, whereas a control engineer may regard it as being a constant value in an analysis. This section presents the terminology used by DESTECS. These terms will be used frequently throughout the subsequent chapters.

- Model: An abstract description of a system, or a part of it, according to a specific purpose.
- Discrete Event (DE) model: Describes the controller of the system using a discrete event formalism. The controller is discrete in the sense that it only interacts with the environment at specific discrete points in time.
- Continuous Time (CT) model: Represents the physical environment (or physical dynamics) of the system, expressed using a CT formalism.

- Co-model: Formed by the DE and CT models and the contract defining the communication between them.
- Co-simulation: Simulation of a co-model, where a specific scenario is chosen as input.
- Design parameter: Describes a property of a model, which remains constant during a co-simulation, but may change between the different runs.
- Shared variable: Variables shared between the domain models (the DE and CT models) of a co-model. Shared variables may change during a co-simulation, and can be classified as either *controlled* or *monitored*. Controlled variables are changed by the DE model and read by the CT model, whereas monitored variables are changed by the CT model and read by the DE model.
- Event: Events raised in the CT model, may result in a response in the DE model.
- Contract: Defines communication between the DE and CT models using design parameters, shared variables and events.

2.4. DESTECS formalisms and main tools

The approach chosen by DESTECS is to combine existing tools, in order to enable the creation of co-models. The *Overture* tool [Larsen&10a] is built on top of the Eclipse platform[1], and supports DE modeling using the Vienna Development Method (VDM) formalism (see section 2.4.1). The Overture interpreter is capable of performing DE model simulations, which enable simple model validation. Subsequently, the simulation or run of a model, will be referred to as *model execution*. For modeling of the physical environment (the CT model) DESTECS relies on the *20-sim*[2] tool, which uses the bond graph formalism (see section 2.4.3). A co-simulation is managed by the co-simulation engine, which is responsible for handling the coordination between the DE and CT models, being simulated in their respective simulators. Details on co-simulation can be found in [Fitzgerald&11].

2.4.1 VDM

VDM is one of the most well-established formal methods for describing computer-based systems [Larsen&10b]. VDM was invented during the 1970s at IBM's Vienna Laboratory, and has evolved into several dialects. The VDM Specification Language (VDM-SL) is ISO standardized, and used for writing specifications for sequential systems. Later VDM-SL has been extended into VDM++, including ordinary Object-Oriented (OO) concepts and concurrency [Fitzgerald&05]. Functionality is expressed using operations and functions. Operations are allowed to manipulate the state of objects in which they reside, whereas functions are only allowed to access the input specified in the signature. An extension to VDM++, called VDM-RT, supports modeling of real-time and distributed systems [Larsen&09]. VDM-RT is the dialect used by the DESTECS tool-chain for DE modeling.
VDM supports several abstraction and consistency mechanisms. *pre* and *post* conditions are used for specifying properties to hold before and after operations or functions have been invoked. *Invariants* can be used for specifying predicates, which must always hold for instance variables and

[1] Eclipse: http://www.eclipse.org
[2] 20-sim: http://www.20sim.com

user defined types. The interpreter, executing the model, will do the consistency checks automatically, and alert the modeler in case any violation occurs.

2.4.1.1 Concurrency and distribution modeling

VDM-RT allows modeling of distribution using special class constructs for CPUs and buses, which are characterized by capacity and scheduling policy. Everything concerning the distribution of the model, is handled in a special **`system`** class. Buses connect the CPUs, which are allocated by objects of active classes. These instances have their own periodic thread of control, specified using the **`periodic`** keyword in the **`thread`** section of the particular class. Operations are synchronous by default, but can be made asynchronous using the **`async`** keyword. The **`duration`** and **`cycles`** keywords, can be used for specifying an execution delay as a fixed-time estimate, or relative to the capacity of a processor, respectively. This may be useful, if characteristics of the system components are already available. The notion of global time is also included in VDM-RT, and may be referred to using the **`time`** keyword. In addition to this, VDM-RT enables early Design Space Exploration (DSE) of different electronic system architectures, by executing the model in a virtual environment [Lausdahl&11].
VDM-RT supports modeling of access to shared resources using **`mutex`** constraints and permission predicates, where the latter must be true before execution is allowed. False predicates will therefore block the calling thread. Besides referring to instance variables, permission predicates may refer to special *history counters*. These are self-contained variables, which count the number of times each operation of an object has been requested, activated or completed [Fitzgerald&05].

2.4.2 Validation techniques in a VDM setting

The VDMUnit testing framework [Fitzgerald&05], allows the modeler to build test cases very similar to what can be obtained using the JUnit testing framework[3]. *Proof Obligations* are logical assertions or checks, which can be automatically generated from the model. These checks can be useful for avoiding potential run-time errors, as the proof obligation generator will detect all places, where they may be violated. Proof obligations can be proven formally or dealt with by manual model inspection [Fitzgerald&07a]. Typically they are used by the modeler, for detecting any potential misapplication of partial operators. As an example, a common pitfall is to do a *sequence application* outside the valid index boundaries. A sequence is similar to an array, present in many programming languages.
The *showtrace* plugin [Larsen&09], included in the Overture tool, enables validation of strict deadline requirements, before any implementation has taken place. When the model has been executed by the interpreter, the resulting *trace file* can be analyzed graphically. This can be used for obtaining insight into the timing and ordering of operation invocations, the message exchange and the thread scheduling, resulting from the chosen distribution architecture. Therefore, the showtrace plugin can be used for analyzing the DE model, after a co-simulation has taken place.

2.4.3 Bond Graphs

20-sim uses bond graphs [Amerongen10] as the underlying formalism for modeling of the physical dynamics. The bond graph notation is a domain-independent way of modeling the ideal transfer of energy in physical systems. Bond graphs are directed graphs, where the vertices are *sub-models* exchanging energy through edges called *bonds*. For example, the `R`-element (or sub-model) may represent both electrical resistance or mechanical friction. Energy exchange is characterized by

[3] JUnit framework: http://www.junit.org

flow and *effort* variables, denoted f and e. The meaning of these variables depend on the particular domain. In the electrical domain, flow and effort refer to current and voltage, whereas the analogy in the mechanical domain is force and velocity, respectively. The energy exchanged through a port is the product of the flow and the effort variable. Fundamentally, the physical concepts remain the same across the different domains, which makes the bond graph notation useful for modeling of systems, where several physical domains are included. Two types of junctions exist for connecting sub-models. The **1**-junction represents a common flow, similar to a series circuit in the electrical domain, while the **0**-junction represents an effort, where the algebraic sum of all the flows sum to zero. Hence, the two junctions meet more general cases of Kirchhoff's voltage and current laws, respectively.

2.4.4 20-sim validation techniques

20-sim also offers graphical validation techniques, which makes it particularly suited for visualizing the impact of design decisions, before the system has been realized. 2D visualization (graph plotting) is supported by a variety of features, such as the possibility of doing multiple runs automatically. This may be useful for performing parameter sweeping, optimization, curve fitting and so on. For example, parameter sweeping is a technique used for investigating ranges of values for certain parameters automatically, which is particularly useful during a fine-tuning process. In addition to this, 20-sim also makes it possible to construct 3D animations for visualizing the underlying equations of a model. This may be useful for many dynamical systems, where the impact of design decisions may be difficult to understand using 2D visualization only.

2.5. Automated Co-model Analysis

Automated Co-model Analysis (ACA) is a tool-supported technique provided by the DESTECS tool-chain, for exploring the design space. Using this technique, the modeler does not have to deal with the manual work of setting up each design solution configuration, and performing the corresponding co-simulation. ACA enables running multiple different co-simulations (with minimal user intervention) using Shared Design Parameter (SDP) sweeping. Shared in this context simply means that the design parameter is relevant to both the DE model and the CT model, e.g. the position of a sensor. SDP sweeping allows the modeler to specify ranges to be applied for the different design parameters, along with the increment in between the co-simulation runs. It is possible to use several SDPs when performing SDP sweeping, one must just be aware of the exponential growth of the design space. In addition to this, logging may be used for determining the design solution, leading to the best result of the ACA execution, based on some well-defined criterion. For example, the fastest completion time of a route, followed by a robot.

2.6. Fault modeling

When modeling faulty behavior, it should be considered good practice to keep a clear separation between faulty/realistic and ideal behavior, to avoid model *pollution* [Broenink&12b]. A component exhibits *ideal* behavior, if no disturbances and noise exist. However, often it will be necessary to account for these things, to make the model reflect a real component operating in a real environment. This is referred to as modeling of *realistic* behavior. Finally, the behavior of a component

after a fault (cause of an error) has been activated, is called *faulty* behavior.
Much effort has been spent within the DESTECS project on finding proper strategies, for modeling of realistic and faulty behavior. The result of this work is a series of suggestions and patterns, for dealing with commonly observed situations within the field of fault modeling, both for the DE and CT models. As an example, the voter pattern [Broenink&12b] enables modeling of tolerance to faults. It works by comparing several different sources of the same information, to obtain greater confidence in the true value of it. Another example is the kernel pattern [Broenink&12b], which can be used in the DE model, for protecting against unsafe controller behavior. Such behavior could be caused by a controller, responding to inputs from malfunctioning sensors.
The DESTECS methodology [Broenink&12b], specifically suggests using the OO inheritance of VDM, for modeling of faulty behavior. As an example, a class exhibiting realistic sensor behavior, might serve as a superclass for other classes modeling faulty behavior. In this way, the faulty behavior could be introduced, by overwriting functionality of the superclass. Hence, inheritance enables simple switching between the realistic and faulty sensor behaviors, by instantiation of the appropriate object. Since 20-sim does not support the concept of inheritance, other approaches to fault modeling must be taken. As an example, the DESTECS methodology points out that 20-sim sub-models allow multiple implementations. Using this feature, implementations for both ideal, realistic and faulty behavior may be incorporated into the same sub-model. However, this approach to fault modeling suffers from the drawback that connections between sub-models in 20-sim are *perfect*, and do not allow separate implementations.

2.7. Related work

Several suggestions for tools and technologies have been made within the field of multi-disciplinary system modeling and simulation. This section will briefly mention some of the more notable contributions related to DESTECS.

2.7.1 Ptolemy

Ptolemy [Eker&03] is an open-source framework, which addresses the challenge of the inherent design complexity of embedded control systems. However, unlike DESTECS, Ptolemy does not rely on the coordination between separate simulators. Instead everything is simulated in a single environment. Ptolemy uses an actor-oriented approach, where models are built of hierarchically interconnected actors or sub-models, which interact through ports. Interaction may be either flow of data or control. Domains (in the sense of Ptolemy) define the order of communication and the semantics between the different actors, which may execute concurrently or sequentially. Examples of domains are Communicating Sequential Processes (CSP), Continuous Time (CT), Discrete Event (DE), Process Networks (PN) and Synchronous Dataflow (SDF). Hence, Ptolemy enables modeling using a single formalism, but at the cost of complexity. This is different from the DESTECS approach, which has a clear distinction between the different formalisms of the DE and CT models, where interaction is specified in the contract.

2.7.2 SystemC

It is very common for embedded control systems to include the development of both hardware and software. Traditionally, these disciplines have been treated separately in a sequential fashion. The problem with this approach, is that the software can only be tested in a proper setting, when

the hardware has been manufactured. Instead SystemC [Black&04] suggests considering the development of hardware and software as a collaborative task, using a single language. This makes it possible to postpone the decisions of what system parts goes into hardware and software, to a later stage in the development. The corresponding activity is referred to as *hardware/software partitioning*. SystemC is a C++ class library, which introduces concurrency mechanisms, data types and structures for modeling of both hardware and software, at different abstraction levels. SystemC supports model analysis using simulation, ordinary unit testing and trace files analysis.

2.7.3 CODIS

COntinuous/DIscrete Systems simulation (CODIS) [Nicolescu&06] is a framework, where the DE and CT simulators are OSCI SystemC[4] and Simulink[5], respectively. The approach taken by this framework, is to have all communication between the models, passing through a (co-simulation) bus. The co-model, bus and co-simulation interfaces are generated automatically from the SystemC and Simulink models by the framework, using the CODIS library. Therefore, CODIS makes it possible to use a single modeling environment to generate C-code from models, which can be imported into other simulation environments. Results obtained from experiments, indicate a synchronization overhead of less than 30% in simulation time.

2.7.4 Functional Mock-up Language

Instead of relying on proprietary model interfaces, the approach taken by the Functional Mock-up Interface (FMI)[6], is to define a tool independent standard for model exchange and co-simulation. FMI enables connection of several simulation tools, communicating with each other at discrete points in time, with the exchange of data and synchronization among the simulators, being handled by a master algorithm. For the time between the exchange of data and synchronization, the models are solved individually by the respective simulators.
A Component implementing the FMI, is referred to as a Functional Mock-up Unit (FMU). It is distributed as a file, so it can be imported into different simulation environments. For co-simulation, C-functions are provided for exchange of data, calculation of synchronization points and initialization of communication between the simulators. Another way of achieving tool independence compared to FMI, is to use a tool independent language like Modelica[7], which enables easy exchange of models between different tools. However, obtaining tool-support for a particular language, requires a lot of effort.

2.8. DESTECS for the case study

The benefits of using a specific tool-chain or formalism are not only dependent on the purpose of the model, but also the characteristics of the system to be developed. As an example, when developing mission-critical systems, it is important that the critical system aspects get the required amount of attention. Therefore, abstract modeling is often used during development, to neglect things of less importance. This helps staying focused on the critical system aspects.

[4]OSCI SystemC: http://sourceforge.net/projects/systemc
[5]Simulink: http://www.mathworks.se/products/simulink
[6]FMI: http://www.modelisar.com/
[7]Modelica: http://en.wikipedia.org/wiki/Modelica

2.8.1 Predictability of the co-model

As mentioned in section 1.5.1, the co-model will be used for predicting route completion times for the system realization. Therefore, techniques for adjusting the parameters of the co-model will be needed, in order to make the co-model behave more like the system realization. The DESTECS tool-chain provides comprehensive support for this, in the form of ACA and the parameter sweeping feature of 20-sim.

2.8.2 Abstract modeling and validation

The case study will consider the controller/environment partitioning, not develop new hardware or investigate the hardware/software partitioning possibilities. Although this makes a tool-chain which uses SystemC less attractive, it would still be highly suited for developing control-algorithms for the controller of the robot. SystemC is nevertheless a superset of C++. However, what is needed is a modeling technique, which helps ensuring the consistency of the model at a higher level of abstraction, than what can be obtained using a programming language. The abstraction and consistency mechanisms mentioned in section 2.4.1, along with the validation techniques in section 2.4.2, will be useful for this.
Graphical 3D visualization would be ideal for route following, as it could be used for obtaining rapid feedback from changes made to the control algorithms. For more detailed insight into the inner workings of the robot, 2D graph visualization for analyzing sensor and actuator output, may be useful. The need for visualization during the development of the Autonomous Robot System favors 20-sim, which supports both 3D and 2D visualization. Although Ptolemy also offers 3D and 2D visualization support, it is not as sophisticated and user-friendly, compared to that of 20-sim.
There is no need for extending the co-simulation to include more tools, which more or less obviates the use of FMI. Furthermore, CODIS does not provide the desired level of abstraction for DE-modeling. Nor is there a need for the feature for importing the model into another simulation environment. Based on the above discussion, and the author's previous experience with VDM, the DESTECS tool-chain is chosen for the Autonomous Robot System case study.

Chapter 3

Development Processes for Co-Models

The main focus of this chapter is to outline the structure, and describe the core activities of the development process to be applied during the Autonomous Robot System case study. The details on the case study development, is the topic of chapters 4 and 5. The findings of the case study serve as input for the methodological extensions, and the final evaluation of the development process and methodology in chapter 6.

3.1. Introduction

The development process applied during the Autonomous Robot System case study, will be referred to as *"the work of Wolff"* throughout this chapter. Hence, it can be distinguished from the other work contributions mentioned. The specifics on this development process can be found in [Wolff12], which constitutes work in progress. Therefore it may change, for instance due to the evaluation work of this thesis. The definitions used for describing the development process are consistent with the ones of DESTECS, presented in section 2.3. In addition to this, the definition of *system* matches the one introduced in section 1.2.2.

The development process suggested by Wolff is divided into four phases, each supported by a set of methodological guidelines. These guidelines are captured in the form of small statements, for easy reference throughout development. As an example, *"document all assumptions made of the system"*, represents the suggestion of the first guideline out of 18. The specifics on the methodological guidelines will be gradually introduced, as they are applied during the case study work, presented in chapters 4 and 5. However, this chapter will point out the essence of them.

This chapter starts out by explaining the need for development processes and methodologies, for managing the complexity of a multi-disciplinary system in section 3.2. Next, the development process is presented and related to other work contributions in sections 3.3 and 3.4.

3.2. Addressing multi-disciplinary concerns

There is a tendency to bridge the design gap among the different domains, by reasoning about the system at a higher level of abstraction, using multi-disciplinary models. By analyzing these models, the hope is to obtain useful insight into the system under consideration. Naturally these multi-disciplinary models demand for a notation capable of accommodating all of the involved

domains. Such a notation is found in the System Modeling Language (SysML).

3.2.1 Model maturity

The idea is to consider modeling as a collaborative task, and postpone domain specific activities (software and control engineering etc.) to later project stages, until the multi-disciplinary models have reached sufficient *maturity*. In this context, a mature model fulfills its purpose, i.e. it has brought the necessary insight into the system being developed. The risk of starting domain specific activities early on, and treating the involved disciplines in isolation, is that domain interfaces may be vaguely specified. Therefore, the impact of changes made in one domain, are likely to be poorly understood. This may cause unforeseen problems in a domain, when changes are made in another. This is certainly undesirable, since compatibility between the different domains is required, before the overall system objectives can be fulfilled.

3.2.2 Expectations for development processes and methodologies

A proper methodology should provide guidance on how to construct, analyze and evaluate the produced multi-disciplinary models [Verhoef08]. Specifically by pointing out how supporting tools may be efficiently used for this during development. Since most development is subject to tight budget constraints, proper guidance on how to manage other resources such as time and money, may also be needed. For instance, deciding on when sufficient time has been spent on analyzing the multi-disciplinary models. Another important questions is how to actually transfer the work obtained during early modeling to the domain specific activities, where realization of the system begins. Automation tools such as code-generators, may be helpful for this.

To support the development teams, a development process should define activities, and suggest a specific order in which to perform them, to make sure that the focus is kept on the actual development work. The quality of the artifacts resulting from these activities, can be assessed through review or inspection. In addition to this, the time spent on producing them, may be used for monitoring the progress [Verhoef08]. Although the consequences of enforcing structure should be considered carefully, it may still be useful for making sure that the critical questions get asked in time. Otherwise there is a risk that substantial amounts of rework has to be done. On the other hand, enforcing structure also tends to hamper creativity, as pointed out in [Wolff12].

3.3. The applied development process

The development process to be applied during the Autonomous Robot System case study, is divided into the four phases *model purpose and requirements*, *system decomposition*, *system modeling* and *validation and verification*. Figure 3.1 outlines the corresponding structure, showing these phases in a spiral structure [Boehm88] to emphasize the iterative work-flow. This structure merely serves as a reference, for putting the methodological guidelines into context.

The development process and the supporting methodology have been developed based on experiences with the DESTECS project. Additionally, the capabilities and constraints of the corresponding tool-chain have been taken into account. Still, the development process should be general enough to be applied using other tools-chains as well. Specifically, the model purpose and requirements and system decomposition phases would remain exactly the same, since no tool-chain is required until the system modeling phase. The following sub-sections cover the four phases of the development process, by introducing the essence of the corresponding methodological guide-

lines.

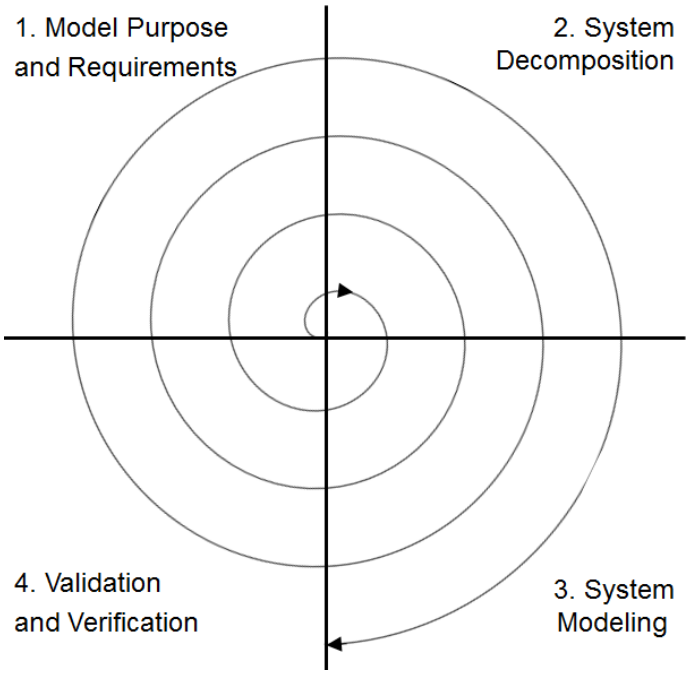


Figure 3.1: The structure of the development process suggested by Wolff

3.3.1 Model purpose and requirements

During the model purpose and requirements phase, the system assumptions and definitions are explicitly documented. This is done to establish a common ground, and minimize the chances of misunderstandings among the representatives of the different domains. Assumptions might state that certain components such as sensors will behave perfectly, i.e. sensor noise is abstracted out of the co-model. Definitions are typically used for agreeing on parameter units (e.g. distance is measured i meters). After the assumptions and definitions have been specified, the purpose of the co-model is described. Using the resulting work, the major system actors and functions are then identified and described using the SysML use case diagram. Since the purpose of the co-model is described in natural language, it ensures that all stakeholders can participate under the same conditions. Put differently, the notation is understandable, whether the participant has an engineering background, or is a non-technical stakeholder (customer or manager etc.). Therefore, this technique is valuable for agreeing upon the direction of the work following. Finally, the system requirements can be derived from the use case diagram, and documented using the SysML requirements diagram.

3.3.2 System decomposition

After the purpose of the co-model has been agreed upon, the SysML Block Definition Diagram (BDD) is used for identifying the hierarchy of composite blocks, making up the system. These blocks are referred to as *main blocks*. The previous work should be used as a check-list for making sure that all the important system actors and parts, have been taken into account. This BDD serves as input for the partitioning, where it must be determined for each of the blocks, whether it belongs to the DE or CT domain. This sub-task will subsequently be referred to as the *DE/CT partitioning*,

and the corresponding decisions, should rely on the judgment of the domain experts. Blocks for which the physical dynamics must be expressed, typically are placed in the CT domain, whereas software controllers belong to the DE domain. Finally, the DE and CT constructs need to be described, and as a final step of the system decomposition, the co-simulation contract needs to be established. These sub-tasks are described below.

3.3.2.1 CT constructs

It may be necessary to obtain a deeper into insight into some of the main blocks identified during the BDD modeling. The methodology suggests using the SysML Internal Block Diagram (IBD) for this. The interaction points of child blocks are described using SysML ports. The *atomic* and *non-atomic* ports are used for indicating uni- and bi-directional flow, respectively. The former may be signaling pulses, directed at a software controller, whereas the latter may be forces acting between two physical objects.
The methodology suggests capturing equations such as physical laws in an additional BDD, using constraint blocks. SysML parametric diagrams can be used for describing these constraints within the context of an owning block. This is done by associating the parameters of the constraint blocks with other parameters and value properties. For instance a value property describing the mass of a car may be bounded to the parameter m of a constraint block, expressing a physical law involving the car mass.

3.3.2.2 DE constructs

The methodology suggests describing the DE constructs of the co-model using traditional software engineering approaches. Class diagrams are used for describing the structure of the software, while behavioral diagrams such as the sequence diagram, the state chart and the activity diagram are used for describing the software dynamics. Note that all of these diagrams are part of SysML, since it provides an extension to the Unified Modeling Language (UML).

3.3.2.3 Co-simulation contract

Establishing the co-simulation contract involves identifying the Shared Design Parameters (SDPs), events and shared variables. The interaction between blocks belonging to different domains, have already been described in the earlier decomposition efforts using SysML ports. This will serve as a good place for identification of shared variables. Events may be revealed by studying the sequence diagrams, and SDPs are likely to be derived as value properties of blocks.

3.3.3 System modeling

When the system decomposition has reached sufficient maturity (the insight needed has been obtained) the modeling approach must be chosen, before the construction of the co-model can begin. The following sub-sections present the different modeling approaches suggested by the methodology, along with a description of the subsequent co-modeling activities.

3.3.3.1 Modeling approaches

The methodology suggests three different ways of approaching the system modeling. Namely *CT-first*, *DE-first* and *contract-first*. The first two approaches suggest building the initial model solely using the corresponding formalism. For the CT-first approach, this means that DE constructs

must be described using a CT formalism initially. Naturally, this may lead to oversimplification of controller behavior, where CT stubs can be used for performing sanity checks of the model, such as basic movements. Likewise for the DE-first approach, where a DE formalism must be used for describing the potentially overly simplified CT constructs. The contract-first approach suggests describing the interface first, and then start working on the DE and CT models in parallel. However, initially stubs may still be used for checking the basic behavior of the two domain models.
The methodology also provides suggestions for how to choose between the different modeling approaches. If the complexity relies within the controller behavior, and discretized CT constructs will suffice initially, then using a DE-first approach may be preferable. The CT-first approach may be a the better choice, if the complexity relies within the physical dynamics, and the CT-formalism will suffice for modeling of the initial DE behavior. If the complexity is evenly distributed over the DE and CT domains, the contract-first approach may also be used. Additionally, choosing the contract-first approach may be easily justified, if a DE and CT team is available for working on the two domain models in parallel. Besides looking at the complexity of the two domains, the experience within a development team, may also influence the choice of modeling approach. As an example, a person with a software engineering background, will most likely find it easier start the co-modeling using the DE-first approach.

3.3.3.2 Co-modeling

When the modeling approach has been agreed upon, it is time to get started with the co-modeling. To keep things simple, the CT-first approach is assumed in the following description. However, the steps for the other approaches will be similar. The methodology suggests using the system decomposition work, for construction of an initial CT model. More specifically, an empty sub-model should be created in 20-sim for each of the IBDs. The Input/Output (IO) specifications of the sub-models should follow the corresponding IBDs. The inner workings of these sub-models can be expressed using the descriptions of the physical laws and dynamics, obtained from the constraint blocks and the parametric diagrams. The methodology suggests starting simple and adding the functionality in small increments. This is mainly due to how Ordinary Differential Equation (ODE) solvers work: Problems become harder to track down, the more differential equations the model is composed of. Hence, this suggestion is likely to minimize the time spend on debugging. Additionally, isolating any issues in sub-models, may also enable easier debugging, since fewer differential equations have to be evaluated.
Initially the methodology suggests neglecting fault scenarios and disturbances such as absence of sensor readings and noise, to avoid complicating the model in the early modeling phases. Instead the focus should be getting the basics working. In the beginning, ideal parameters will suffice, since the focus is not to provide exact values. The important thing is to identify which of the parameters have the greater impact on the co-simulation results, when they are being changed. The identification of these *significant parameters* is supported by the Automated Co-model Analysis (ACA) and parameter sweeping features of the DESTECS tool-chain and 20-sim, respectively. It is recommended that these significant parameters are added to the co-simulation contract, to enable use of ACA, which is useful for later fine-tuning of the co-model.
The methodology introduces the notion of *micro steps* and *macro steps*. These concepts help raising awareness of the fact that many multi-disciplinary issues, are caused by frequent interface changes, not communicated properly to all the domain representatives. A change may be regarded as a micro step, if only it affects one of the domains, while a macro step requires interface updates. As an example, when functionality is introduced in the DE model, which require additional monitored variables in the CT model, the change is a macro step. Using this common vocabulary,

representatives of the different domains can classify likely changes to the co-model, and reason about the impact of introducing them.

3.3.4 Validation and verification

During each iteration, the co-model needs to be evaluated with respect to its purpose. Part of this job can be done using tools specific to each of the two domains. Hence DE and CT model validation can be performed in isolation. Tools specific to the CT domain may include 2D and 3D visualization as provided by 20-sim. Examples of DE domain specific tools are VDMUnit, the proof obligation generator and the showtrace plugin. These validation techniques have already been covered in sections 2.4.2 and 2.4.4.
The methodology specifically advocates the use of *visual validation*. This is not only useful for analyzing basic behavior, it also enables communication of the co-model to non-technical stakeholders. Moreover, 3D visualization may also be useful for understanding the impact of changes to the co-model, which are not easily understood from 2D visualizations such as graph plotting.
Demands for co-model fidelity may increase along with the iterations. If data is available, it may be possible to obtain more realistic parameters values, to make the co-model approach more realistic behavior. The methodology suggests doing this in an iterative fashion, by analyzing measurements made on the system realization.

3.4. Related work

The work within the field of development processes and methodologies addressing multi-disciplinary concerns, is not as comprehensive as the related tool-chain contributions discussed in sections 2.4 and 2.7. Still, a notable contribution is found in the BODERC project [Heemels&07]. It suggests a methodology addressing the challenges of multi-disciplinary systems in a very different way, compared to the work of Wolff. The following sub-sections start out by introducing the BODERC methodology, and the core activities supporting it. Finally, this work is compared to the methodology suggested by Wolff.

3.4.1 BODERC project

The advances in implementation technologies and market demands, naturally cause an increase in design related information. To manage and organize this information, the BODERC project suggests using the CAFCR methodology [Muller04], which enables reasoning at the system-level. This methodology uses a system architectural description, divided into the five view structure given below, for structuring of the design related information.

Customer objectives view: Specifies what the customer wants to achieve.

Application view: Determines how these customer objectives will be achieved.

Functionality view: Identifies the functional properties needed of the product.

Conceptual view: Determines how these functional properties will be achieved.

Realization view: Determines how these concepts will be implemented.

The CA views drive the identification of system functionality in the F view. Altogether the CAF views motivate the design and implementation related work in the CR views. Hence, the customer related information justifies the subsequent decision making. If a functional property, or the design/realization of it, cannot be traced back to a customer objective, there is a problem: Either the first C view must be incomplete, or the functionality is not really supporting anything, the customer wants to achieve. Therefore, the CAFCR methodology clarifies the connection between the customer (CA views) and the product (FCR views).
The BODERC design methodology suggests an iterative approach consisting of the three steps: *preparation*, *selection* and *evaluation*. A description of these steps is given below.

Preparation: Involves things such as domain analysis as well as identification of realization concerns and system requirements. All stakeholders should be present during this step.

Selection: Focuses on identifying the most critical design issues such as trade-off points using qualitative assessment, for estimating the associated priorities and risks. This is to avoid spending too much time on less critical concerns. As an example, performance and cost are often in conflict.

Evaluation: Uses light-weight models or measurements from the system realization, to obtain quantitative data concerning the most critical design issues, identified during the selection step. This data may impact the qualitative assessment made during the preparation step, or even identify new issues or requirements. When using light-weight models, the design methodology advocates the importance of finding the level of abstraction, providing the sufficient data fidelity. Models should at most require a few days of construction and evaluation work.

The BODERC methodology is supported by the three core activities named *the key driver method*, *threads of reasoning* and *budget-based design*. The purpose of having these activities in place, is to document the rationale of the design decisions, and raise awareness of resource limitation. Semantically well-defined notations, supporting these core activities, enable communication of activity output highlights for quick insight. The following description briefly covers these core activities.

The key driver method: Focuses on explicitly capturing the relationship between the important customer objectives and system requirements, from the point of view of a specific stakeholder. This is done by associating the key drivers of the first C view with the application drivers of the A view, which are finally related to the system requirements of the F view. As an example, the key driver "waiting time" could be associated with the application driver "speed", which in turn would be associated with relevant system requirements.

Threads of reasoning: Identifies potential design driver conflicts. Conflicts are design choices, which will influence some design drivers positively, and others negatively. For instance, increasing the number of components may lead to better performance, but at the cost of price. Together a thread is formed by these design drivers, the design choices and the corresponding impact they might have. Hence, this activity ensures that critical design choices are explicitly documented and made traceable, which may be useful for later reference (maybe in a future project). Threads of reasoning is an iterative activity, which may use information from all of the CAFCR views.

Budget-based design: Identifies the critical resources and assigns them a budget, indicating the amount available. Parts of the budget can then be assigned to different functions or compo-

nents of the system. Then the designer of this particular function or component, is responsible for staying within the assigned budget part. As an example, a budget may be used for managing the available power of an electrical system such as a printer.

3.4.2 BODERC and Wolff methodologies differences

The BODERC and Wolff methodologies choose completely different approaches to system-level reasoning. The work of Wolff introduces a light-weight tool-oriented approach, tightly associated with the DESTECS tool-chain, and where the SysML notation is used for initial system modeling. The BODERC methodology is more concerned with the management of possibly vast amounts of design related information, using the CAFCR methodology. The supporting core activities of the BODERC methodology, assist the user in capturing the relationship between the customer and the product. For instance using the key driver method. The threads of reasoning activity helps identifying conflicts and documenting design choice rationale, using qualitative/quantitative assessment and risk analysis. Finally, budget-based design raises awareness of resource limitation. The BODERC methodology adds more overhead to the development work, compared to the methodology suggested by Wolff. This is mainly due to decision rationale and budget documentation. Documenting the rationale may be useful for companies, where similar design decisions are likely to be made in future projects. This information may contribute positively to the productivity of a company in the long run. However, for small one-time projects, this extra effort may not be rewarded. Similarly for risk analysis, which may be useful for identifying threats and weaknesses, which may jeopardize the success of the project, if not dealt with properly. Furthermore, the evaluation activity of the BODERC project suggests using models requiring at most two days of construction and evaluation work. However, this time-framing may impact the fidelity of a model negatively. This differs from the methodology suggested by Wolff, which does not explicitly state the concerns of resource limitation. Hence, the methodologies suggested by the BODERC project and Wolff, may be regarded as commercial and scientific approaches, respectively.

Chapter 4

Decomposition and Modeling

In chapter 3 the development process to be applied during the Autonomous Robot System case study was presented along with a description of the supporting methodological guidelines. This chapter describes and discusses the application of these guidelines according to the observations and experiences made during the first three phases of the development process "model purpose and requirements", "system decomposition" and "system modeling". The application of the guidelines supporting the final phase "validation and verification", is the topic of chapter 5.

4.1. Introduction

The presentation of the development process in chapter 3 covered the activities of the four phases, but omitted the details of the guidelines. This chapter will gradually introduce these guidelines, as they are applied. More specifically, each guideline will be presented right before the application of it. The point of this chapter is not to provide a comprehensive description of the Autonomous Robot System, but instead to demonstrate how the guidelines have been applied. The details of the case study work can be found in appendix B.
This chapter starts out by covering the model purpose and requirements phase in section 4.2. This is followed by the system decomposition phase in sections 4.3 through 4.6. Finally, the system modeling phase is covered in section 4.7.

4.2. Model purpose and requirements

The guidelines presented in the methodology advocate the importance of defining the model purpose and specifying the system requirements. This activity favors informal descriptions, in order to facilitate communication among stakeholders.

4.2.1 System assumptions

Guideline 1: Document all assumptions made of the system.

The following assumptions have been made for the Autonomous Robot System. These are fixed properties of the system itself, or the surroundings in which the robot is performing route follow-

ing.

A1: The robot is performing route following on an even and level surface.

A2: The robot has four wheels.

A3: The Pirate-4WD Mobile Platform[1] will be used as robot body frame.

A4: Adding/removing sensors will not change the robot mass.

A5: All system components (sensors, motors, external systems etc.) are assumed to behave perfectly (i.e. no errors) unless faulty behavior is explicitly stated as being modeled.

A6: All motors share the exact same properties (i.e. they have the same performance characteristics).

A7: Motors attached on the same side (left or right) run at the same speed and direction.

A8: The robot knows its initial position and orientation before starting route following (see the system definitions in section 4.2.2).

4.2.2 System definitions

Besides the system assumptions, it is also appropriate to specify the system definitions, to avoid any misunderstandings. This split between the assumptions and definitions is not directly suggested by the guidelines. Still, it is recommended by the author of this thesis, as it is being considered good practice. The following definitions have been made:

D1: All units will be SI-units, unless explicitly stated otherwise.

D2: A route is an ordered sequence of two-dimensional way points.

D2.1: Subsequent way points are required to be different.

D3: The starting position of the robot is defined as $(0, 0)$.

D4: The robot position is measured from the starting position, to the center of rotation of the robot.

D4.1: Initially the robot will have a forward direction following the positive y-axis.

D5: Orientation or rotation is measured in radians, and always considered as being around the vertical z-axis in the three-dimensional space.

D5.1: The positive direction of orientation or rotation is defined as clockwise.

D5.2: The initial orientation of the robot is zero radians.

[1] The Pirate-4WD Mobile Platform: http://www.robotshop.com/dfrobot-4wd-arduino-mobile-platform-3.html

4.2.3 Purpose of the co-model

Guideline 2: Define the purpose of the model by identifying all actors and use cases of the system.

The main purpose of the co-model is to analyze autonomous route following behavior of a robot in given surroundings, while providing it with a route and inputs from different types of sensors. The final version of the co-model will be used for predicting route completion times for the system realization.

4.2.4 Use case modeling

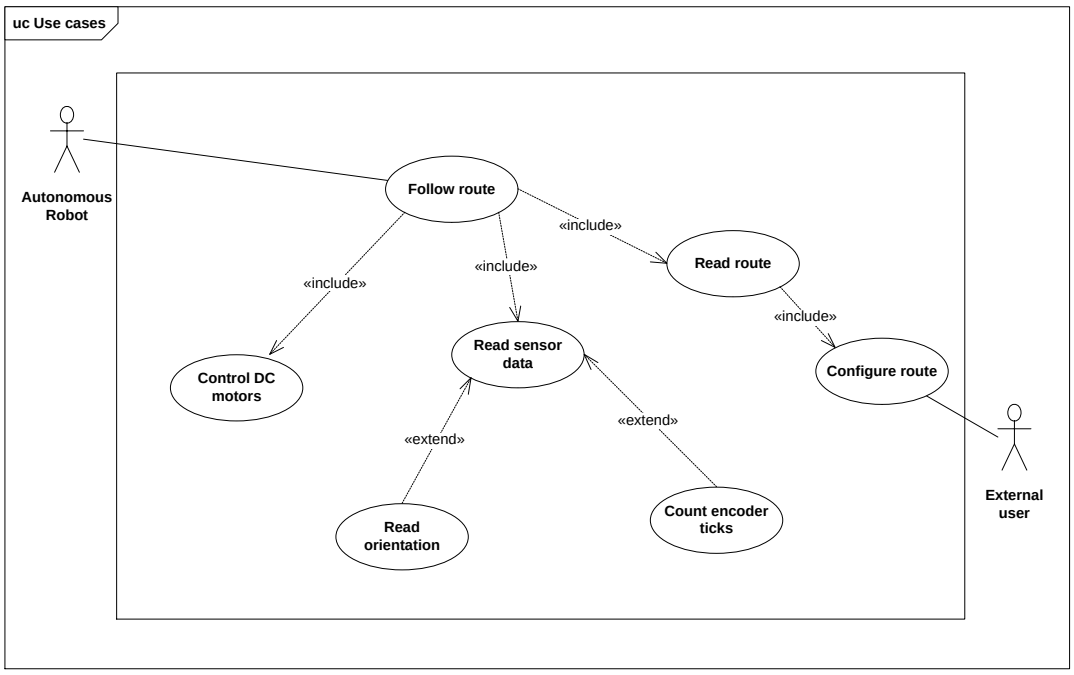


Figure 4.1: The use case diagram

Figure 4.1 shows the use cases identified for the system. The main task of the robot is to follow a route autonomously in given surroundings, by having a controller making the driving decisions. This controller will subsequently be referred to as the *decision controller*. Ultimately, robot control is realized by controlling the DC motors, which turn the wheels. In order to perform this task, the robot is provided with a route, configured by an external user and read by the robot. The robot is equipped with sensors, in order to improve its route following capabilities. A gyroscope is used for determining the orientation of the robot, while encoders can be used for measuring the rotation of a wheel, by counting the ticks generated by it. Thus, encoders can be used for determining the traveled distance, given that the wheel diameter is known. However, when estimating the distance traveled and the orientation of the robot, errors are accumulated over time. This may result in route following deviating increasingly from the route, as the time passes.

4.2.5 Requirements modeling

Guideline 3: Derive requirements of the system from the use cases defined.

The requirements identified for the Autonomous Robot System have been organized using a System Modeling Language (SysML) requirements diagram, as suggested by the methodological guidelines. The identification of these requirements is based on the use case modeling in section 4.2.4. The resulting requirements diagram is shown in figure 4.2. Although this diagram only shows four requirements, this way of grouping and structuring requirements, could be more valuable for a larger system.

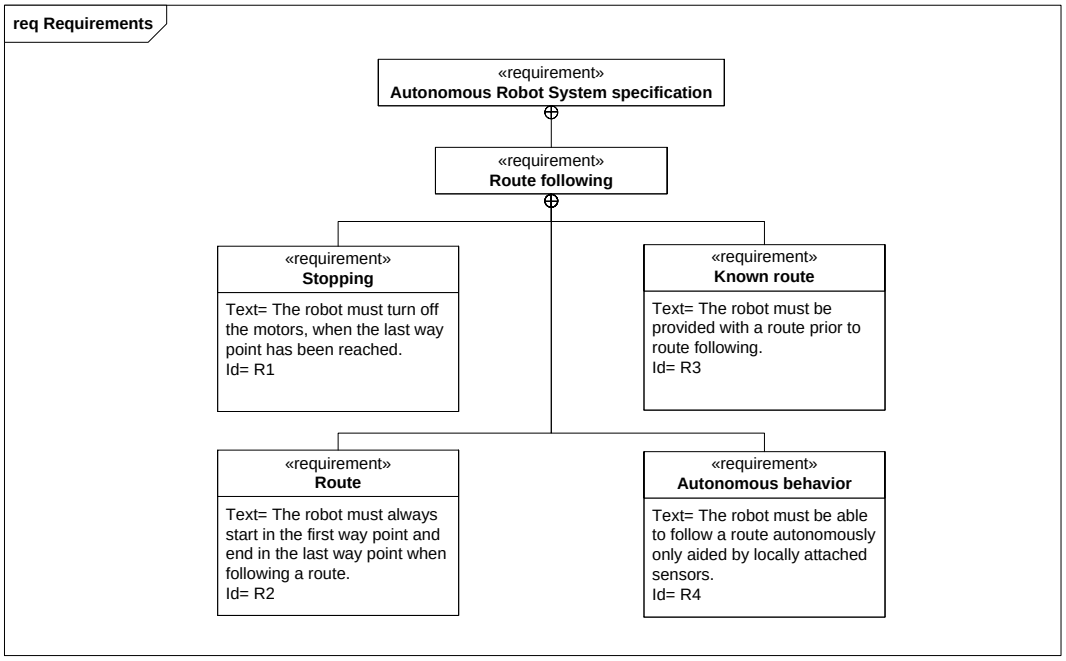


Figure 4.2: The system requirements listed in a requirements diagram

4.2.6 Model purpose and requirements retrospective

Defining the model purpose and specifying the system requirements as the first things worked well, because it ensures that the work following, is taking a proper direction. Since the guidelines only encourage use of notations which can be understood without any special prerequisites, it was useful in order to obtain a common understanding of the Autonomous Robot System. However, the textual descriptions suffer from the ambiguity inherent in a natural language. Although the consequences of ambiguity are always undesirable, they will be far more severe for mission-critical systems, where people's lives or huge financial costs are at stake. Finally, it would also have been appropriate for the guidelines to have suggested the split between the system definitions and assumptions, as mentioned in section 4.2.2.

4.3. System decomposition: Structure and partitioning

For system decomposition, the guidelines suggest using the SysML Block Definition Diagram (BDD), which enables modeling of the system without yet making decisions on which domain (DE or CT) to place the different system constructs. The output of the BDD modeling is then used for the DE/CT partitioning, which must be done eventually.

4.3.1 Block Definition Diagram modeling

Guideline 4: Define main parts of the system in a *block definition diagram* and determine which domain each individual block belong to.

To provide better insight into the hierarchical structure of the Autonomous Robot System, a SysML BDD is shown in figure 4.3. The knowledge acquired from the previous use case diagram modeling was used as a check-list, to make sure every major actor and function of the system were covered.
In this diagram the root block, the specification of the Autonomous Robot System, has been further decomposed into two sub-systems. The first represents the robot, while the second represents the surroundings, being formed by the surface on which the robot is driving.

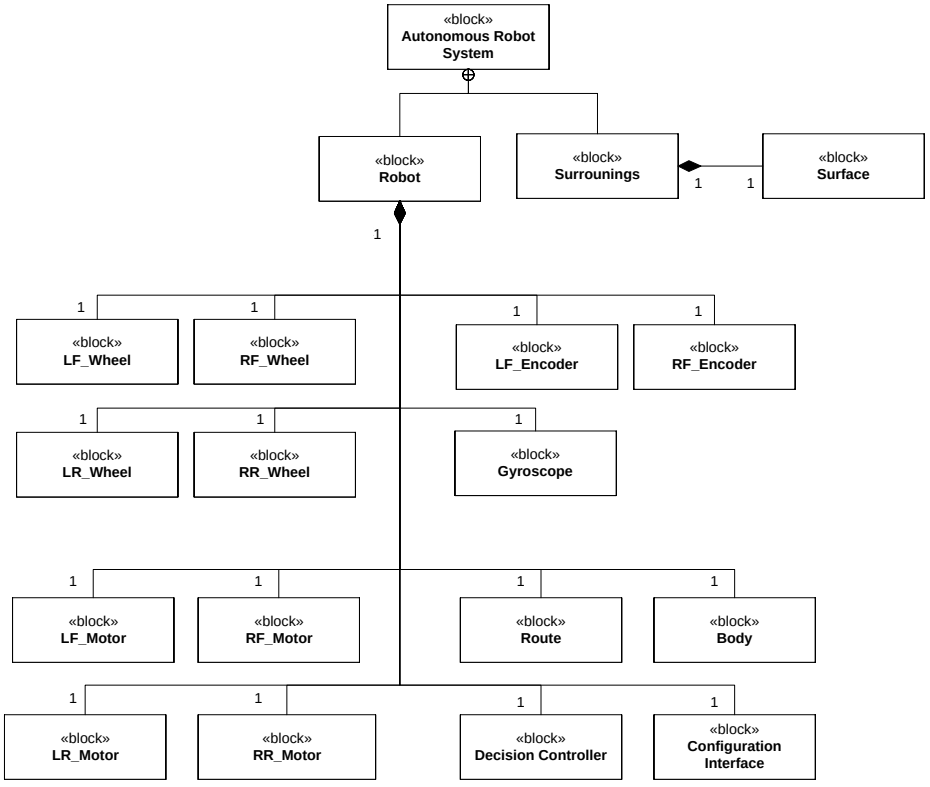


Figure 4.3: The BDD of the Autonomous Robot System

Besides the body frame, the robot is composed of wheels, motors and sensors. The motors are

prefixed by LF (left front), RF (right front), LR (left rear) and RR (right rear) in order to distinguish between them. The same naming convention is used for wheels and encoders. The intelligence of the robot is the decision controller, which controls the motors. The decision controller is able to make the robot drive, by sending control signals to a motor actuator (H-bridge [Scherz&00]), which controls the motors and hence the wheels. To avoid too much detail, the motor actuator is not shown in any of the diagrams in this chapter.

4.3.2 DE/CT partitioning

The decision controller of the system will need complex control logic, in order to describe the driving decisions, and therefore the corresponding block in figure 4.3, is put into the DE domain. The route and configuration interface are tightly connected with the decision controller. The former will serve as input for determining the driving decisions, and the latter will be used for loading the route. Therefore these blocks are also placed in the DE domain.

The motors, encoders, gyroscope, wheels and robot body frame are all put in the CT domain, since their physical dynamics must be described. For example, sensors (encoders and gyroscope) have to directly interact with the physical environment. To obtain input for route following, these sensors can be read using Input/Output (IO) interfaces in the DE model. In addition to this, the `Surroundings` block consisting of the `Surface` block, relate to the physics of the CT model, and therefore these blocks belong to the CT domain as well.

4.3.3 Structure and partitioning retrospective

Decomposing the system using a BDD worked well, and the blocks in the diagram were used as a check-list, when doing the (DE/CT) partitioning. The BDD is a modified Unified Modeling Language (UML) class diagram, and therefore the notation it relies on, may be more easily understood by a person representing the DE domain. The precise semantics of the notation, such as that of the composite associations, are not easily understood without any prior experience with SysML or UML. However, the required effort to be able to understand the notation for this particular diagram, should be fairly small. The robotics engineer had some experience with UML, and the BDD worked well as an enabling tool during the system decomposition phase.

The choice of which blocks go into the DE and CT domains may be difficult to make. In addition to this, partitioning decisions may also depend on the tools and formalisms being used. As pointed out in [Wolff12], 20-sim provides strong support for modeling of Proportional-Integral-Derivative (PID) controllers [Goodwin&01]. However, ultimately a PID controller will most likely be implemented in software. The partitioning problem gets difficult, when the blocks can be put in both domains. For such situations, it is important to rely on the judgment of domain experts, in order to increase the chances of making proper partitioning decisions. During the case study work, it was unclear whether the sensor blocks should be placed in the DE domain or the CT domain. However, by discussing the partitioning of each of the blocks in figure 4.3, confidence in the decision was gained.

4.4. System decomposition: CT constructs

The guidelines of this section suggest that the CT constructs identified during partitioning in section 4.3.2, need to be described in more detail, before the CT model is being constructed. The methodology suggests describing the main CT blocks using the SysML Internal Block Diagram

(IBD), while using constraint blocks and parametric diagrams for describing the physical laws and dependencies.

4.4.1 Internal robot structure

Guideline 5: Define the internal composition of the main blocks in *internal block diagrams*. The interface between child-blocks is defined using ports.

Figure 4.4 highlights a small section of the complete internal structure of the robot, to demonstrate how IBDs have been used during the case study, for obtaining a deeper insight into the CT constructs. A more detailed description can be found in section B.2.1.

The motor-wheel configuration is completely symmetric for every wheel. As a consequence of this, figure 4.4 only shows the right front wheel (along with motor and encoder).

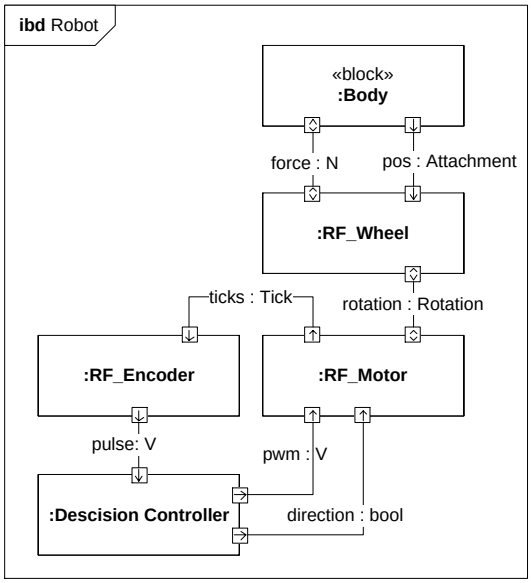


Figure 4.4: The internal structure of the robot

It is important to note that the driving capabilities of the robot will be influenced by how and where the wheels are connected. Not only the coordinates of an attachment are important, the orientation must also be taken into account. Therefore this attachment is described using the `Attachment` type.

For motor control, a Pulse-Width Modulation (PWM) signal [Horowitz&89] and a directional signal are needed. The direction can be either forward or backward, and hence indicated by a Boolean value. The encoder is a spinning disc with a mass that is very small compared to that of the wheel. Hence the energy required for rotating the disc is very small compared to what is needed for making the wheel turn. This explains why the rotational influence of the encoder disc is not shown in figure 4.4.

4.4.2 Physical dynamics descriptions

Guideline 6: Using a combination of the constraint blocks and the parametric diagram is a valuable tool for documenting not only the differential equations of the system, but also the causality between these.

It was found to be difficult to express the physical laws and dependencies of the wheel forces in a useful manner, when using the parametric diagram in particular. This difficulty was caused by the circular dependencies among the different wheel forces (see section B.2.3 for details). Therefore, the resulting diagram is hard to read, due to the many interrelations, which produce a lot of overlapping lines. Instead the physical descriptions have been made using informal sketches, combined with classical mathematical notation. This choice is re-addressed in section 4.4.3 for discussion.
This sub-section covers the physical dynamics of the system, focusing on the movement of the robot. The point is not to provide an exhaustive description of the CT model, but to demonstrate how informal sketches and classical mathematical notation have been used for describing the physical dynamics. More elaborate details of the wheel forces can be found in sections B.2.2 and B.2.3. The robot orientation is the topic of section B.2.3.1.

4.4.2.1 The robot position

The movement of the robot is described by the resulting force acting on it. This force is equal to the sum of the resulting wheel forces in the x-y plane, generated by the respective DC motors. More formally this can be written as:

$$F_{tot} = F_{LF} + F_{RF} + F_{LR} + F_{RR} \tag{4.1}$$

However, equation (4.1) does not take the robot orientation into account. In figure 4.5, the rotation of the forces is expressed using a rotation matrix.

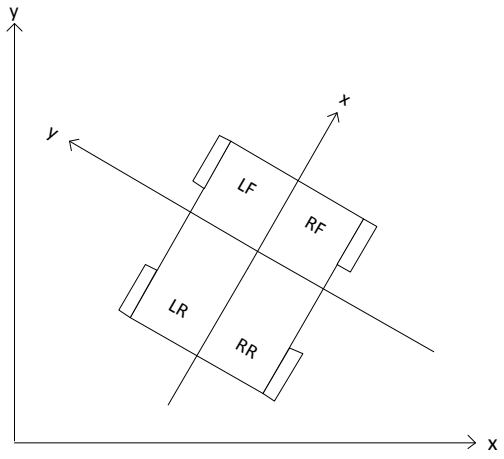


Figure 4.5: The global and local robot coordinate systems

Let *R(θ)* denote the rotation matrix. Now the rotation of F_{tot} by the angle θ, F_{rot}, is given by:

$$F_{rot} = R(\theta)F_{tot} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} F_{tot} \tag{4.2}$$

The velocity of the robot can be found by integrating this force and dividing the result by the mass of the robot m.

$$v = \frac{1}{m} \int F_{rot}\, dt \tag{4.3}$$

Finally the position vector x can be obtained by integrating the velocity and adding the initial robot position.

$$x = \int v\, dt + x_0 \tag{4.4}$$

4.4.3 CT constructs retrospective

The IBDs of the robot provided a strong basis for further discussions on the physical dynamics, which was very useful for reaching a common understanding among the representatives of the DE and CT domains: Mostly the software engineer would ask questions, and the robotics engineer would answer them. As an example, the robotics engineer explained the workings of the encoders and the gyroscope to the software engineer. In addition to this, the diagrams were also useful for understanding the number and types of control signals between the decision controller and the motor actuators.
The first real issue encountered when applying the methodological guidelines, occurred when constraint blocks and the parametric diagram were to be used for describing the physical laws and dependencies. The way of specifying physical dependencies using a parametric diagram, was not as easily understood as the notations for the other types of SysML diagrams, previously used during the case study. Instead of using the parametric diagram, informal sketches of the robot and the physical forces were drawn on a white board, serving as a basis for further discussions. From these sketches, the equations describing the physical dynamics could be derived and described using a classical mathematical notation. However, in some cases applying the parametric diagram may be useful. As an example, in [Wolff12] it was successfully applied for describing the dependencies among the constraints of a flare for the Flare Dispensing System (see section 1.5.4).

4.5. System decomposition: DE constructs

The guidelines of this section suggest using ordinary UML diagrams for describing the structure and behavior of the DE constructs identified in section 4.3.2. The resulting diagrams will be used for alleviating the process of transitioning to the VDM modeling.

4.5.1 Decision controller modeling

Guideline 7: Use UML class diagrams to define the structure of the DE controller.

The structure of the decision controller is shown in the class diagram in figure 4.6. The upper part of the diagram deals with the IO interface classes for the sensors and actuators of the system. The abstract class `AbstractSensorReal` is concretized as `SensorReal`, from which the IO objects for the gyroscope and the two encoders will be instantiated. The reason for using the `SensorReal` class for encoders, is that the co-simulation contract in the DESTECS tool-chain, only is able to cope with types **`real`**, **`bool`**, **`array`** and **`matrix`**.

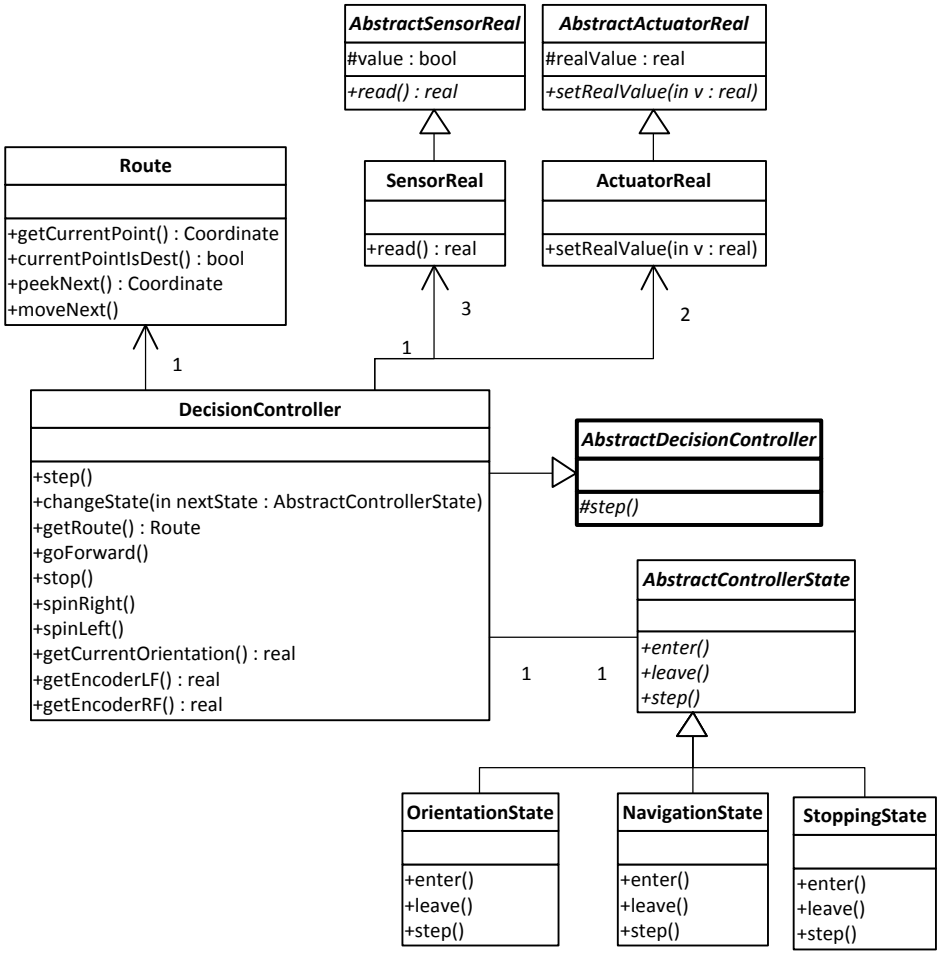


Figure 4.6: The class diagram of the decision controller

The two motor signals are represented by IO objects, instantiated from the `ActuatorReal` class. For the system realization, there will be a difference in orientation between motors attached on the left side of the robot, and motors attached on the right side of the robot. This fact has been abstracted out of the model. As a consequence, a positive signal will mean "forward" for all motors. Similarly, a negative signal will mean "backward".
The `DecisionController` is a specialization of the `AbstractDecisionController` class. One thing which is important to note here, is that the `DecisionController` is an active class, as indicated by the bold face border of its superclass. The periodic `step` operation of the `DecisionController`, is responsible for invoking the other `step` operations of the controller states (classes inheriting from `AbstractControllerState`). The state pattern [Gamma&95] has been used for incorporating the Autonomous Robot System state machine

into the design as an inheritance hierarchy.

The state objects control the robot during route following, by invoking the relevant operations in the `DecisionController` class. As an example, `OrientationState` will invoke the `spinLeft` and `spinRight` operations. These operations are used for making the robot *spin* around its center of rotation, when searching for the direction, which will set it towards the next way point. Subsequently, the activity of spinning while searching for a target direction, is referred to as *turning*. Similarly, `StoppingState` will invoke `stop`, while `NavigationState` will determine the traveled distance using the `getEncoderLF` and `getEncoderRF` operations. In addition to this, a `Route` class helps keeping track of the current way point and the following way points to be visited. The details on route following will be covered in section 5.4.1.

The final VDM model will contain `World` and **`system`** classes, which are responsible for setting up the entire system and handling the distribution, respectively. These classes have been omitted from figure 4.6 for simplicity.

Guideline 8: Use UML behavioural diagrams to define the intended behaviour of the DE controller.

Additional diagrams have been provided in appendix B for expressing the behavior of the Autonomous Robot System. The state machine is described using a state chart in section B.3.1, along with a sequence diagram expressing the dynamics of it in section B.3.2.

4.5.2 DE constructs retrospective

Using UML diagrams for describing the DE constructs worked well for several reasons. First, the notation is specifically targeting software engineering. Secondly, the notation was well-known in the development team. Similar to the IBDs used for describing the CT constructs in section 4.4, the UML diagrams could be used for obtaining a common understanding of the DE model. The only difference is that the roles were more or less reversed: Mainly the robotics engineer would ask the questions and the software engineer would answer them. This is a likely situation for other projects applying these methodological guidelines, if the persons representing the CT domain are working on the edge of software development (electrical or robotics engineering for instance).

State machines are very useful for describing embedded control systems, as the transformation into VDM-RT can be done by following the diagram-to-code mapping rules of the state pattern. This principle is general enough to be applied for most embedded control systems, and therefore knowing about this technique, may improve the productivity of the development work.

4.6. System decomposition: Co-simulation contract

The guidelines of this section suggest defining the co-simulation contract using the work, resulting from the application of the methodological guidelines presented in the previous sections. This corresponds to identifying the Shared Design Parameters (SDPs), events and shared variables.

4.6.1 The co-simulation contract

Guideline 9: For the co-simulation contract, derive the monitored and controlled *shared variables* from the interface between DE and CT blocks in the *internal block diagrams*; the *shared design parameters* from the *parametric diagrams*; and the *events* from *sequence diagrams*.

The SDPs, events and shared variables which have been identified are listed in the co-simulation contract shown in table 4.1.

Table 4.1: Co-simulation contract

Co-simulation contract				
	Name	Type	Default	Note
SDP	`initialPos`	**`array`**	[0, 0]	Starting position
	`initialOrient`	**`real`**	0	Starting orientation
	`encoderRes`	**`real`**	22	Encoder resolution (22 ticks per revolution)
	`wheelRadius`	**`real`**	0.033	Wheel radius
Controlled	`motorLeft`	**`real`**	0	Motor signal range: $[-1, 1]$
	`motorRight`	**`real`**	0	
Monitored	`encoderLF`	**`real`**	0	Encoder tick counts
	`encoderRF`	**`real`**	0	
	`currentOrient`	**`real`**	0	Current robot orientation

Some parameters such as mass and rotational inertia of the robot are not included in the contract, because they will not be accessed by the DE model. These "extra" parameters will be addressed in section 4.6.2.
The decision controller changes the motor direction by regulating the sign of the motor signal. Hence, there is no need for additional variables for directional motor control. The two motor signals make up all of the controlled variables. Monitored variables represent the encoder ticks and robot orientation. Encoder ticks are modeled as shared variables, but could also have been represented as events. However, the DE model will only need to read the current state of the encoder counters regularly, not necessarily every time they change. Using two encoders, 22 (the encoder resolution) events per wheel revolution would require a lot of event handling, when the robot is moving around.

4.6.2 Other parameters

Besides the SDPs identified, there may also be other parameters, which the modeler do not want expose through the contract. Perhaps these parameters will not be accessed by both the DE and CT models (they are not really shared). Some of them will have a stronger impact on the co-simulation results than others, and may require extra attention during a fine-tuning process. It may be that these parameters cannot be accurately determined, because information concerning the system realization is not currently available, or that crudely tuned parameters will suffice for the given time. Hence, it makes sense to postpone the fine-tuning (if necessary) of these parameters to a later stage. Parameters to be fine-tuned could be added as SDPs to the co-simulation contract, which would enable Automated Co-model Analysis (ACA) as described in section 2.5. Still, local fine-tuning in 20-sim could be significantly faster. However, it should be considered good practice to explicitly write down these parameters. A list of these parameters can be used as a check-list,

to make sure that the important parameters of the CT model have been taken into account during fine-tuning. Such a list for the Autonomous Robot System has been included in section B.4.1.

4.6.3 Co-simulation contract retrospective

Identifying SDPs, shared variables and events from the SysML descriptions, the equations and the informal sketches, was a straight-forward task when the partitioning was done. However, there may be parameters which are not appropriate to put in the contract. The guidelines did not provide any information about this, so instead these parameters were written down in a separate list, for reasons explained in section 4.6.2. It may be appropriate to group these parameters in the same sub-model in 20-sim for easy reference, instead of having them spread across the model.

To return to the discussion of the parameter fine-tuning mentioned in section 4.6.2, it may not always be obvious which of the parameters have the greater impact on the outcome of the co-simulation. Parameter sweeping may enable identification of these significant parameters (see section 4.7.2).

4.7. System modeling

The methodology presents three different modeling approaches, namely: CT-first, DE-first and contract-first. In addition to this, suggestions are provided for choosing between them. When the modeling approach has been chosen, the guidelines provide inputs to the modeler on how to structure the model and identify the significant parameters.

4.7.1 Modeling approaches

Guideline 10: Determine the correct modeling approach (CT-first, DE-first or contract-first) by analyzing the system decomposition.

The complexity of the physical dynamics of the Autonomous Robot System favors the CT-first modeling approach. Additionally, the complex part of the DE control can be postponed to a later stage, because it is not needed in the beginning of the system modeling. Initially, basic behavior (driving forward, spinning etc.) suffice for sanity checks of the model. This can easily be done using 20-sim alone. Complex behavior will then be added later, when the DE model is introduced.

4.7.2 Co-modeling

Guideline 11: For each of the internal blocks, create an empty sub-model in 20-sim with the IO as specified in the internal block diagrams.

The methodology suggests structuring the CT model according to the IBDs. Instead of strictly following the guideline, it was chosen to divide the CT model into two sub-models: one representing the body of the robot, and another representing the motor-wheel part. The latter sub-model could

then easily be duplicated and used for the other motor-wheel parts.

Guideline 12: When developing CT models start simple and add functionality in small increments — this limits the risk of issues with the Ordinary Differential Equation (ODE) solver and eases debugging.

The sub-models were added sequentially: first the motor-wheel sub-models were created, followed by the more complex robot body. In this way functionality could be gradually introduced into the model and analyzed separately.

Guideline 13: Start by creating an idealised model where all parameters are ideal and any possible fault scenario cases are disregarded.

When modeling the motor-wheel part, ideal parameters were not chosen. This part of the system was available as a legacy sub-model from a previous project, where it was used for a robot with two wheels. Hence, it worked well for that particular project, and therefore it was mistakenly assumed that it would work for the Autonomous Robot System as well. This particular issue will be addressed later in this section.
Simple behavior such as going forward and spinning was tested early on to analyze the model. The details on this will be described in section 5.2. It was clear from the 2D graph plots showing the robot position and orientation from the initial analysis that the robot was not able to spin, which did not conform to the expected behavior. Not even for extreme cases with motors in each side powered asymmetrically.

Guideline 14: Find the significant parameters in the co-model using parameter sweeps or "Automated Co-Analysis".

In order to understand why, the values of the parameters needed to be investigated. Clearly some parameters were more interesting to investigate than others. For instance, the mass of the robot and wheel radius were not of much interest. The focus was turned to the rotational inertia of the robot, the axle widths, and the parameter expressing resistance to sideways movement. By performing parameter sweeping, it was found that the problem relied within the parameter expressing resistance to sideways movement. It became clear that it was accidentally set to a value which would avoid any velocity in the sideways direction. By lowering the value, the robot started to conform to the expected behavior (it was able to spin).

Guideline 15: Isolate issues in sub-models to ease debugging.

Although the methodology suggests isolating issues in sub-models, this was more a problem of integration, when the sub-models were connected.

4.7.3 System modeling retrospective

Most of the guidelines directly associated with system modeling, reflected a natural way of working during the case study work. However, enforcing this kind of structure makes sure that the right questions get asked early on. The system modeling guideline, which turned out to be most interesting for this case study work, was the one regarding parameter sweeping. Identifying the problem with resistance to sideways movement was difficult and took some time. When sweeping a single parameter, the values of other parameters may also suppress the effect seen from the sweeping: What if the rotational inertia of the robot was set for a very high value, would the robot still be able to spin, when sweeping other parameters? One simple solution could be to sweep several parameters at once, but for many parameters this solution clearly is not scalable. If a lot of parameters are to be swept simultaneously (the design space is very large), smarter ways

of solving these problems are needed. Another approach could be fixing certain parameters at values, which do not suppress the parameters being swept. For instance keeping the rotational inertia of the robot artificially low, when sweeping the parameter associated with the resistance to sideways movement, in order to make the robot spin. Another fundamental problem is actually expressing what to look for, when performing parameter sweeping. For the example from the case study, it was a significant change in orientation. The lesson learned is that the best way of solving these problems, is to obtain a good understanding of the underlying physics, instead of simply sweeping through every parameter blindly. Hence, a highly qualified domain expert will always be invaluable for such cases.

Chapter 5

Validation and Verification

In chapter 4 the methodological guidelines supporting the first three phases of the development process to be evaluated, were applied during the Autonomous Robot System case study. This chapter continues with the fourth and final phase "validation and verification", by describing how the remaining guidelines have contributed to the development work. The case study related findings serve as input for chapter 6, where extensions for the methodology are suggested, and the final evaluation of the development process and methodology is presented.

5.1. Introduction

This chapter gradually introduces the guidelines right before their application is demonstrated and discussed. For the reader to be able to sufficiently understand the application of the development process, this chapter describes the most important aspects of the validation and verification related development work. This is done by giving concrete examples of how the techniques suggested by the methodology have been applied. Additional details relevant to this phase of the development process can be found in appendix B.
First, the use of 3D animations for obtaining rapid feedback during the initial construction of the co-model is described in section 5.2. Next, the use of domain specific tools for validating and analyzing the domain models separately is covered in section 5.3. Finally, the co-model is compared to the system realization in section 5.4.

5.2. Validation and verification: Rapid feedback

5.2.1 Use of a 3D animation

Guideline 16: Use 3D animations early in the validation process to ensure correct overall behavior of the co-model.

As explained in section 4.7.1, the CT-first approach was chosen for system modeling. Initially a very simple 3D animation was built with the purpose of ensuring the correct behavior of the underlying CT model. It was decided that the 3D animation of the Autonomous Robot System (a screen shot of this is shown in figure 5.1) should visualize the robot movement and orientation. In

additional to this, it should also reflect the rotational speed of the wheels during route following.

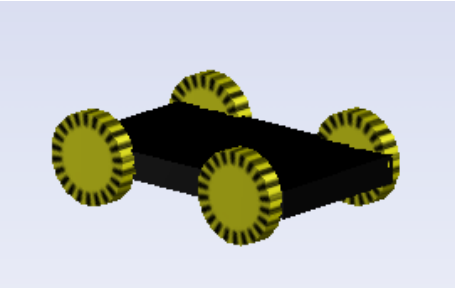


Figure 5.1: Screen shot of the Autonomous Robot System 3D animation

To be able to analyze the CT model, very simple sanity checks were established for making the robot:

Move forward: by signaling "forward" to each of the motors.

Drive left or right: by signaling asymmetric forward speeds to the motors of each side of the robot.

Spin: by signaling "forward" to the motors of one side and "backward" to the motors of the other side. To make the robot spin around its center of rotation, the same speed was being signaled for both sides.

These checks were encapsulated in a single 20-sim sub-model as separate implementations (in the sense of the 20-sim term). This would allow easy switching between them, without polluting the model with analysis related formation. Carrying out the initial analysis in 20-sim was appropriate, as it was based on very simple behavior. This resulted in significantly faster simulations, compared to using the DESTECS tool-chain.

5.2.2 Rapid feedback retrospective

The Autonomous Robot System 3D animation was built in a few hours without any prior experience with 3D animating in 20-sim. This was time well spent, as it turned out to be very valuable during the process of analyzing and constructing the co-model. As an example, the 3D animation helped revealing problems with the underlying calculations concerning searching for target directions when spinning. These problems would immediately appear in the 3D animation as wrongly chosen target directions, not simply caused by small deviations.

Besides using the 3D animation for obtaining rapid feedback when analyzing the behavior of the CT model and co-model, it has additional advantages worth mentioning. A stakeholder who does not understand the applied DE and CT formalisms, will have a hard time validating the system. This potentially results in expensive rework, if the system realization turns out to deviate from the true needs of the user. 3D animations, enabling communication of the system functionality, may serve a useful tool for coping with such problems.

5.3. Validation and verification: Domain specific tools

Guideline 17: Use domain specific tools on isolated sub-models to validate the functionality disconnected from the rest of the system.

5.3.1 DE specific tools

During the construction of the DE model, a class was created for representing the route to be followed by the robot. As stated in the system definitions in section 4.2.2 *"a route is an ordered sequence of two-dimensional way points"*. Furthermore, subsequent way points are required to be different, since it would not make sense for the robot to move to a point, in which it already is. Therefore, the definition of a route has been changed to require this. This was caused by the feedback of the VDM modeling, which naturally enforces stronger definitions due to its mathematical notation. In general this technique works well for revealing ambiguity in the system specification. Thus, using a formal specification technique causes a higher degree of precision, but requires technical prerequisites. To demonstrate its use during the case study, the VDM example below formalizes the definition of a route. Line 1 in the example initializes the coordinates of the route to the starting point. Lines 2-5 state an invariant, which must hold for the `coordinates` variable at all times during execution. It demands that the coordinates of the route are different from the empty sequence, and that the starting point is indeed $(0, 0)$ (lines 2-3). Finally, the quantified expression **`forall`** is used for ensuring that subsequent points are different (lines 4-5).

```
1 private coordinates : seq of Coordinate := [mk_Coordinate(0,0)];
2 inv coordinates <> [] and
3         coordinates(1) = mk_Coordinate(0.0,0.0) and
4         forall i in set {1,...,len coordinates -1} &
5                 coordinates(i) <> coordinates(i+1);
```

Besides the use of invariants for ensuring the consistency of the route definition, other validation techniques specific to the DE domain, have been applied during the case study. As an example, the VDMUnit testing framework has been used for creating unit tests for validating utility functions used by the robot for route following. This technique has been very valuable due to the support for regression testing.

5.3.2 CT specific tools

Besides the use of a 3D animation, 20-sim 2D graph plots also served useful for validating and understanding the inner workings of the CT-model. As an example, this technique was used for analyzing the behavior of the encoder model shown in figure 5.2.

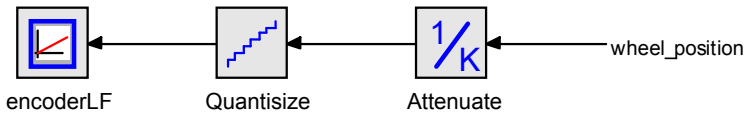


Figure 5.2: The CT model of an encoder

This model takes as input an absolute angular position of a wheel, to output the corresponding encoder ticks. To reflect real encoders, the quantization block makes sure that only the whole

ticks are crossing the co-simulation interface. This particular encoder model will increase or decrease the tick count, depending on whether the wheel is trying to move the robot in a forward or backward direction, respectively. This does not reflect the real encoders, which simply generate pulses independently of the robot direction. However, choosing abstractions which deviate from the nature of the system realization is not bad practice, as long as it does not violate the purpose of the model. This particular topic will be treated in greater detail in section 6.2.4.1.
The encoder model was analyzed by comparing the wheel position to the corresponding encoder value using the 2D graph plotting feature of 20-sim. A plot visualizing these variables for the robot moving in a forward direction, is shown in figure 5.3. Using the *numerical analysis* feature of 20-sim, it can be confirmed that an encoder value of 22 (the number of ticks per wheel revolution) indeed corresponds to a wheel position of 2π radians etc. Although this technique does not serve as a proof of correctness, it is a useful technique for gaining confidence in sub-models.

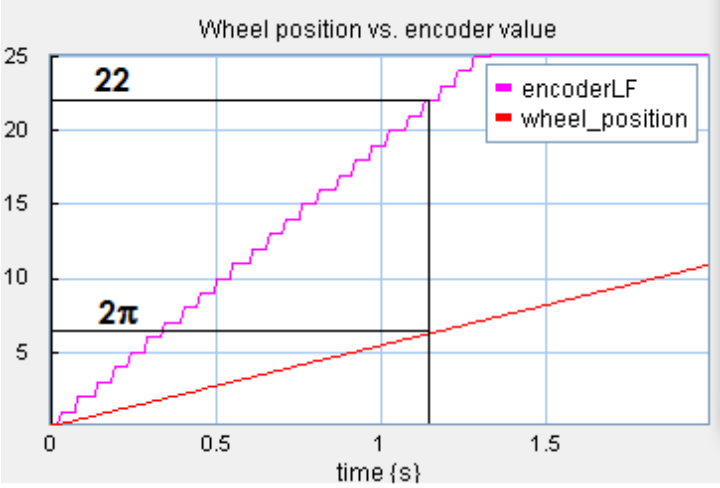


Figure 5.3: Plot showing the wheel position and the encoder value

5.3.3 Domain specific tools retrospective

The domain specific tools, optimized for a particular domain model not only perform well, they also support analysis at a wide range of different abstraction levels. Besides being used for validating the co-model, the feedback of abstract modeling can be used for strengthening the system specification. During the case study the use of VDM lead to a stronger route definition as shown in section 5.3.1. More specifically, this technique was used for revealing ambiguity in the initial route definition. As another example, the DE specific tools complemented the 3D animation well. As described in section 5.2.2, the 3D animation worked well for detecting problems with the initial CT model. As a next step, the DE specific tools like VDMUnit could be used for precisely locating these problems, so they could be dealt with.

5.4. Validation and verification: System realization

The Autonomous Robot System was gradually realized as the co-model evolved. After the system realization and co-model reached their final version, they were instructed to follow certain routes. Data were collected and used for adjusting the significant parameters, in order to make the co-

model behave more realistic with respect to route completion times. Subsequently, this process will be referred to as *co-model calibration*, and the state of the co-model before and after as being *initial* and *calibrated*, respectively.
What motivates the calibration work of the case study, is the opportunity of using a co-model for making predictions concerning the system realization. Put differently, a co-model of sufficient fidelity can be used for making critical design decisions virtually. This does not only increase the productivity of the development work, it also significantly reduces the time and money spent on physical prototyping (see section 6.2.2). In this section the Autonomous Robot System co-model is calibrated to investigate if predictions can be made concerning the route completion times of the system realization.

5.4.1 Route following

The routine used by the robot during route following (covered in detail in section B.3.1) toggles between the *orientation state* and the *navigation state*, until the final way point has been reached. A description of these states are given below.

Orientation state: The initial state. The robot spins around its center of rotation until the target orientation, which will set it towards the next way point, has been reached. In section 4.5.1, this was referred to as *turning*. Furthermore, the robot chooses the direction that will require the least amount of turning, in order to reach the target orientation as fast as possible. The speed at which the robot is spinning/turning is referred to as the *rotational speed*.

Navigation state: The robot follows a straight line until the distance between the current and the target way point has been covered. The speed at which the robot does this is referred to as the *forward speed*.

The co-model and system realization were instructed to follow the three routes shown in figure 5.4 using this routine. These routes will subsequently be referred to as the "square wave" route (left), the "box" route (center) and the "arrow" route (right).

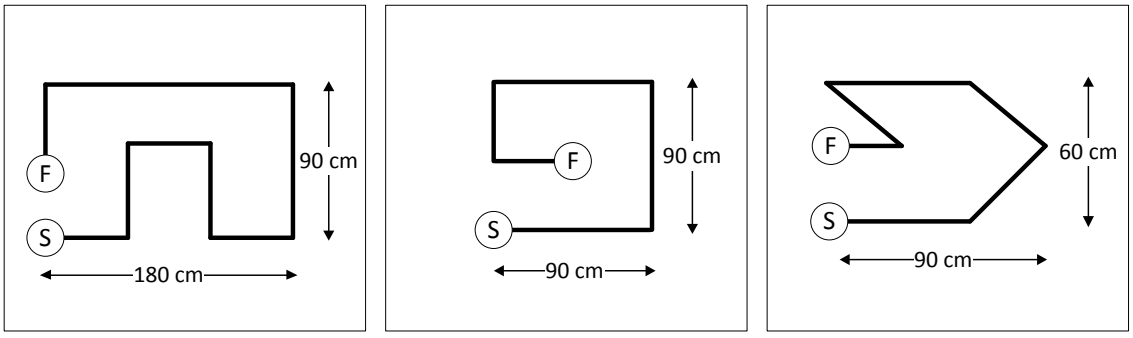


Figure 5.4: An overview of the three routes to be followed. Left-most: the "square wave" route. Center: the "box" route. Right-most: the "arrow" route. The way points of each route to be visited by the robot include all corner points and "F" (finish)

The "square wave" route represents the longest route, where the robot will be taking both left and right turns during the route following. The "box" route is shorter than the "square wave" route, but requires only left turns. The "arrow" route is the shortest route, but it differs from the other routes in that sharper turns are required. These routes have been chosen to investigate the robot behavior under different circumstances.

5.4.2 Measuring conventions

The distance covered by the system realization or co-model, D_{sys}, is compared to that of the route, D_{ideal}, in terms of absolute and relative error, denoted ΔD_{abs} and ΔD_{rel}, respectively. Finally, letting $dist(s_{i-1}, s_i)$ denote the distance between the sampled positions s_{i-1} and s_i, and n be the total number of samples, leads to the definitions below.

$$D_{sys} = \sum_{i=2}^{n} dist(s_{i-1}, s_i) \tag{5.1}$$

$$\Delta D_{abs} = |D_{sys} - D_{ideal}| \tag{5.2}$$

$$\Delta D_{rel} = \frac{\Delta D_{abs}}{D_{ideal}} \tag{5.3}$$

To avoid confusion, a lower case d will be used for indicating that deviations are calculated with respect to the system realization. In addition to this, t_{sys} represents the measure of route completion time for any of the two system representation. When referring to the route completion time of the co-model or system realization in particular, t_{real} and t_{co} will be used instead. Finally, co-model deviations with respect to the system realization, are calculated in the same way as equations (5.2) and (5.3).

5.4.3 Materials and methods

A lot of effort was put into constructing the measuring setup used for obtaining data from system realization tests (shown in figure 5.5).

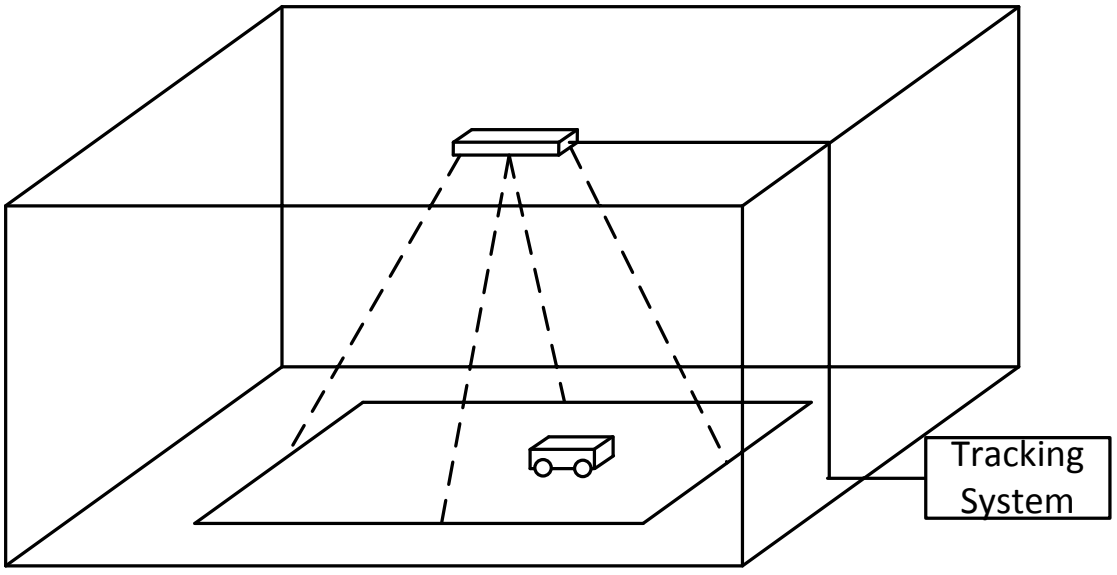


Figure 5.5: An illustration of the setup used for obtaining data from the system realization

During the following of a route, a camera mounted in the ceiling of the testing room would continuously capture images (320 x 240 pixels) of the robot, as it was moving around. These images were sent to a tracking system, where the current robot position was derived using a tracking algorithm, implemented as part of the case study work. When the robot finished route following, the

position data were written to a text file, which would undergo further processing before the final presentation of the results given in this thesis.
Tasks like physically installing the measuring equipment and implementing the software systems used for collecting the data measurements, were amounting for almost half of the time spent on the development work. Still, some data measurements have been obtained using primitive equipment. As an example, a stopwatch and a measuring tape were used for determining the rotational speed and the forward speed of the robot (see section B.5.1). This is acceptable, since the aim of the case study is not to obtain data or create a co-model of high fidelity, but rather try to cover as many aspects of the development process as possible, within the time-frame of the thesis.

5.4.4 Approach

The system realization was first instructed to follow the "square wave" route to obtain data for co-model calibration. To make the co-model behave more realistic with respect to route completion times, the rotational speed and the forward speed of the co-model were adjusted using additional measurements from the system realization. To investigate the effect of the calibration process, new routes were chosen, and the co-model was instructed to follow them. Before repeating the following of these routes using the system realization, an attempt in predicting the route completion times was made, based on results of the co-simulation. Finally, the predictions were evaluated to judge the effect of the calibration process.
The presentation of the results of following a single route will be done in two steps. More specifically, it comes down to the following comparisons: *1) Co-model vs. system realization* and *2) Systems vs. route*. The concrete steps taken during the co-model calibration are listed below.

- **Step 1:** Initial test of the systems
  - The initial co-model and system realization were instructed to follow the "square wave" route.
- **Step 2:** Calibrating the co-model.
  - The rotational speed and the forward speed of the system realization were determined.
  - The rotational speed and the forward speed of the co-model were calibrated using the data obtained from the system realization.
- **Step 3:** Test of the calibrated co-model.
  - The co-model was instructed to follow the "box" and "arrow" routes.
  - Predictions were made concerning the system realization following these routes.
  - The system realization was instructed to follow the "box" and "arrow" routes.
  - The predictions were evaluated.

5.4.5 Step 1: Initial test of the systems

The initial version of the co-model was first compared to the system realization, to determine what parameters needed to be changed, to make the co-model appear more realistic. For easy comparison, the movement of the co-model and system realization are visualized together in figure 5.6 for the "square wave" route.
Judging from the movement alone, the co-model and system realization tend to behave similarly, in that they both turn more than actually requested. Moreover, when the system realization is

following a straight line, the distance covered is a little longer, than what is actually calculated (an explanation follows in this sub-section).

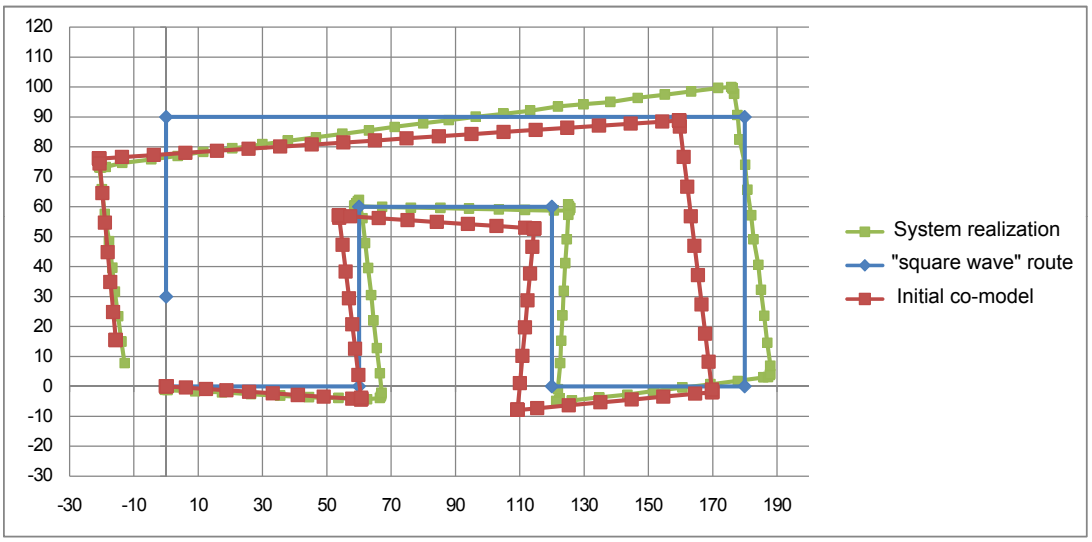


Figure 5.6: Initial co-model and system realization “square wave” route following

In table 5.1 the results of the co-model are compared to the system realization. The results show that the distance covered by the co-model deviates by 9,09% with respect to the system realization. A smaller deviation would be expected considering the tracking plot of figure 5.6 alone (an explanation follows in this sub-section). Moreover, the co-model deviates by 34,77% in terms of route completion time.

Table 5.1: Comparing the initial co-model to the system realization before calibration using the “square wave” route

“square wave” route						
Description	D_{sys} [cm]	Δd_{abs} [cm]	Δd_{rel}	t_{sys} [s]	Δt_{abs} [s]	Δt_{rel}
System realization	702,24	-	-	58,01	-	-
Initial co-model	638,43	63,81	9,09%	37,84	20,17	34,77%

In table 5.2 the system realization and co-model are compared to the route in terms of distance covered. The system realization is measured to drive 11,47% longer than the distance of the route. There are several reasons explaining this observation. First, the system realization tends to drive too long, when following a straight line. This is caused by the period at which the robot checks if the current distance is at least that of the target distance. A similar argument can be used for explaining the rotational overshoot. Secondly, the position of the robot is also measured, when the robot is turning. The robot tends to drift a little when turning, as the center of rotation is not symmetrically placed in the center of the robot body frame, as assumed by the co-model. This is mainly caused by a shift in weight, due to how the hardware such as battery holders have been attached to the robot body frame. Another factor is the drifting caused by the immediate change from rotation to driving straight, and vice versa. During tracking the low resolution images of the camera also introduced inaccuracy. Thus, there are many factors affecting the outcome of the

results.

Table 5.2: Comparing the initial data results to the "square wave" route

"square wave" route, $D_{ideal} = 630$ cm			
Description	D_{sys} [cm]	ΔD_{abs} [cm]	ΔD_{rel}
System realization	702,24	72,24	11,47%
Initial co-model	638,43	8,43	1,34%

The co-model represents a very abstract version of the system realization, which only covers some of the system parameters and variables. As an example, describing the physics concerning resistance to forward movement in detail, would be extremely complex. For the Autonomous Robot System case study, it was modeled very simplistically to start with. More specifically, the co-model was assumed to have full friction with no wheel slip, when the wheels were being turned by the DC motors. In section 5.4.6, it will be explained how a very simple model for expressing resistance to forward movement has been introduced, to make the co-model behave more realistic. Although the resistance to sideways movement has been considered in the CT model, it is only modeled to be constant, which is an example of another approximation.

5.4.6 Step 2: Calibrating the co-model

Guideline 18: Make the ideal parameters more realistic in an iterative process until the co-model and the real system behave similarly.

In section 5.4.5, it was found that the initial co-model behaved similar to the system realization: Both system representations would overshoot in terms of turning and driving straight. On the other hand, the results of table 5.1 showed a co-model deviation of 9,09% and 34,77% in terms of distance covered and route completion time, compared to the system realization.
Instead of trying to calibrate all aspects of the co-model, to make it behave more similar to the system realization, the methodological guidelines suggest doing it in an iterative process. Hence the impact of changing a single parameter can be treated in isolation to start with, to get a better idea about the corresponding impact. When treating more parameters together, it may be difficult to determine, which of them have the greatest impact with regards to a certain *quality criteria* (completion time, robot speed etc.). Although parameters are likely to be re-calibrated during the next iteration, the suggested technique allows managing the complexity of the calibration process gradually.
For the Autonomous Robot System, the focus was directed at increasing the fidelity of the route completion times of the co-model, by adjusting its speed. Due to the way the robot moves (see section B.3.1), this process may be split into two, by adjusting the forward speed and the rotational speed separately. To regulate the former, another parameter has been introduced for changing the resistance to forward movement, whereas the latter can be adjusted using the existing parameter, expressing resistance to sideways movement. An important thing to note here, is that the resistance to forward movement will also influence the rotational speed. However, the resistance to sideways movement will not influence the forward speed. Thus the resistance to forward movement should be adjusted before the resistance to sideways movement.
The calibration was performed using two separate simulations in 20-sim, one where the robot would follow a straight line, and another where the robot would spin left. Having determined the forward speed of the system realization (see section B.5.1), the expected distance to be covered by the model during the simulation run was calculated. The value was then set as target for the

distance covered by the robot during the simulation. Next, the parameter expressing resistance to forward movement was investigated using the parameter sweeping feature of 20-sim. Initially, very crude parameter sweeps were performed to narrow down the design space, making it possible to perform Design Space Exploration (DSE) at a finer granularity. By repeating this process, an approximated value of the parameter was found. The same procedure was repeated for determining the rotational speed of the robot. A final thing to note, is that the parameter sweeping could have been performed equally well, using the Automated Co-model Analysis (ACA) feature of the DESTECS tool-chain. The reason for not doing this, is that basic behavior such as moving straight forward and spinning, is easily expressed in 20-sim alone. This also results in faster simulation times, since no information has to be exchanged via the co-simulation interface.

5.4.7 Step 3: Test of the calibrated co-model

To determine the effect of the calibration process, a re-run of the route was performed using the co-model, in order to obtain the new tracking data shown in figure 5.7. As it can be seen in table 5.3, the calibration has lead to a an improvement of the co-model in terms of route completion time inaccuracy, reducing it from 34,77% to 14,60%. In addition to this, the tracking plot of figure 5.7 and table 5.4 also show that the lower forward and rotational speeds have lead to a more precise co-model with respect to following the route. The co-model not only deviates less in terms of driving straight (albeit very little), the rotational overshoot is also smaller. In fact this makes the co-model behave less like the system realization with respect to this aspect of the system behavior. This particular observation and ways of dealing with it, will be discussed in section 5.4.7.1.

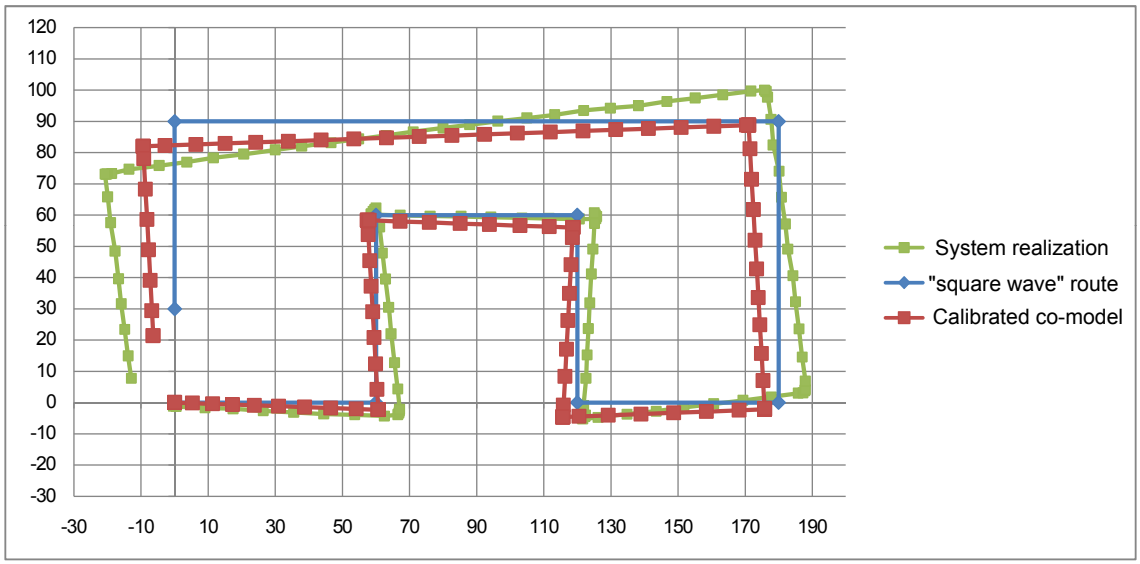


Figure 5.7: Co-model and system realization “square wave” route following after calibration

Although the completion time of the route still deviates by 14,60%, compared to that of the system realization, one should recall that a lot of inaccuracy affects the results (see section 5.4.3).

5.4.7.1 Further co-model calibration

The co-model calibration process made the co-model behave less like the system realization with respect to turning and driving straight. Part of the explanation is found in the way in which the robot periodically checks if the target orientation or distance have been reached. When the robot

Table 5.3: Comparing the co-model to the system realization after calibration using the "square wave" route

"square wave" route						
Description	D_{sys} [cm]	Δd_{abs} [cm]	Δd_{rel}	t_{sys} [s]	Δt_{abs} [s]	Δt_{rel}
System realization	702,24	-	-	58,01	-	-
Calibrated co-model	635,99	66,25	9,43%	66,48	8,47	14,60%

Table 5.4: Comparing the data results to the "square wave" route after calibration

"square wave" route, $D_{ideal} = 630$ cm			
Description	D_{sys} [cm]	ΔD_{abs} [cm]	ΔD_{rel}
System realization	702,24	72,24	11,47%
Calibrated co-model	635,99	5,99	0,95%

is turning at a lower speed, this check is performed at a finer granularity due to the calibration. In addition to this, the lower speed also makes it easier to bring the robot to rest. Bringing back the angular overshoot, in order to make the co-model behave more like the system realization, can be done in two steps: First the overshoot in orientation is estimated for the system realization. Then this estimate is added to the target orientation of the robot. This could be written informally as:

$$target\_orientation = calculated\_orientation + overshoot; \tag{5.4}$$

A similar technique can be used for making the co-model drive longer, when following a straight line between two way points. Another approach is to lower the frequency at which the robot checks if the target orientation or distance have been reached. Other things which could be adjusted is considered in section B.5.4.

5.4.7.2 Predicting route completion times

The data obtained from tracking the system realization and the calibrated co-model following the "square wave" route, indicated a deviation of 14,60% in terms of route completion time. If this is a general tendency exhibited by the co-model across routes, it will be possible to use the co-model for predicting route completion times for the system realization with respect to other routes.
The calibrated co-model was instructed to follow the "box" and "arrow" routes (details are found in sections B.5.5 and B.5.6). Before making the system realization follow these routes as well, the expected route completion times were determined. These predictions were calculated using the route completion time deviation of 14,60% for the "square wave" route, denoted p. Due to the knowledge obtained from initial route following, the co-model was expected to be slower than the system realization. This led to the expected route completion time τ given below.

$$\tau = (1 - p)\Delta t_{co} \tag{5.5}$$

The actual route completion times of the co-model and system realization are listed together with the expected route completion time in table 5.5 (first three columns). In addition to this, the table lists the deviations $\Delta\tau_{abs}$ and $\Delta\tau_{rel}$, calculated with respect to the system realization (columns four and five). The results show route completion time deviations of 5,52% and 1,01% for the "box" and "arrow" routes, respectively. If estimates had been made based on the co-model route

completion times only, they would have deviated by 10,63% and 15,92% (last column). Therefore, understanding how the co-model deviates from the system realization can be another way of making better predictions.

Table 5.5: Calibrated co-model and system realization route completion times comparison using the "box" and "arrow" routes

Calibrated co-model vs. system realization Route completion time predictions						
Description	$t_{co}[s]$	t_{real} [s]	τ [s]	$\Delta\tau_{abs}[s]$	$\Delta\tau_{rel}$	Δt_{rel}
"box" route	38,92	35,18	33,24	1,94	5,52%	10,63%
"arrow" route	35,40	30,54	30,23	0,31	1,01%	15,92%

5.4.8 System realization retrospective

Having covered all the phases of the development process, some of the advantages and disadvantages of co-modeling became clear. Put shortly, co-modeling alleviates the process of transitioning to the domain specific activities, but also brings a higher degree of flexibility into the development work, when performing creative activities like DSE. The former being due to the insight obtained into the system being developed, and the latter being caused by the virtual nature of the co-model. On the other hand, the challenge remains obtaining the necessary level of co-model fidelity, without investing too much effort into co-model calibration.

5.4.8.1 From co-modeling to domain specific activities

Realizing the system was a very time-consuming task compared to co-modeling for several reasons. First, device drivers as well as the tracking and data logging algorithms needed to be developed. Secondly, the case study work included a lot of low-level tasks like assembling the robot and installing the camera physically. However, these time-consuming tasks are required as part of the system realization. One may argue that the camera setup is only used for obtaining data for the co-model, but it might as well be used solely for improving the system realization, if physical prototyping was chosen over co-modeling. These things amounted for most of the time spent on the development work, and could easily be neglected during co-modeling. Being able to simply neglect the most time-consuming tasks during co-modeling makes it particularly attractive in terms of investigating the feasibility of a system (see section 6.2.2.2). It significantly reduces the time being spent on co-modeling, but also makes it easier to focus on more important things without having to care about the details of the target platform. The experience from the case study, is that the time spent on transforming the DE model into target code, is very small compared to that of actually developing the controller design and algorithms. The reason for this is that VDM easily maps into the Object-Oriented (OO) programming language of the target platform. This process may take more time if there is a shift in programming paradigm (the programming language of the target platform is not OO) or the language mapping is not well-understood. Still, code-generation may be helpful for this task [Fitzgerald&07a].

5.4.8.2 Co-modeling and time spent on development

Another interesting question is whether co-modeling and system realization combined, would have taken less time compared to realizing the system directly. Although it cannot be said for

sure, the co-modeling approach is expected to be the most time-consuming on a first application. One reason is that learning a new tool-chain was required, which naturally takes some time during the development work. Still, the experience gained is transferable to future projects, and thus the productivity will increase in the long run when using the same tool-chain. Another explanation is found in the time spent on system decomposition and modeling, as described in chapter 4. Not all the artifacts produced during these phases could be readily reused for the realization work. The System Modeling Language (SysML) DE model design was easily transferable to the OO design used for the target platform. It was a little different with respect to the CT descriptions. Instead, these descriptions helped obtaining insight into the system being developed. As an example, they were used for understanding the workings of encoders, which was necessary for developing the device driver. The time spent on describing and modeling the CT constructs has amounted for most of the time spent on the first three phases of the development process. However, in a development setting where a CT expert would be present all the time, this would most likely not have been the case.

5.4.8.3 Virtual nature of a co-model

As part of the system realization, parameter sweeping was used for increasing the fidelity of the co-model. The co-model calibration enabled prediction of route completion times for the system realization, deviating by 5,52% and 1,01% with respect to the "box" and "arrow" routes. Although these predictions demonstrated a simple example using primitive materials and methods, the point that models can be used for predicting properties concerning the system realization remains intact. Furthermore, these results could be improved by making the system realization approach more ideal behavior, by for instance adding more sensors to it. As an example, it would be possible to provide the system realization with the current position as measured by the tracking system, which could be used for preventing overshoot when following a straight line etc.
Since the co-model is a virtual representation of the system realization, it is possible to explore the design space with a higher degree of flexibility, compared to physical prototyping. Besides being used for increasing the fidelity of the co-model, parameter sweeping also enables DSE, which would otherwise be difficult or time-consuming to do using physical prototyping. In general it applies to parameters expressing physical dimensions such as width, length and height that they are difficult to explore, using the approach of physical prototyping. As an example, it could be interesting to investigate the maneuverability of the Autonomous Robot System, by reducing the wheel dimensions or axle widths. To investigate how such changes impact the route following capabilities of the robot, it would be desirable to visualize the robot following a route using the new parameter values. Such parameters could be investigated quite easily, using ACA due to the virtual nature of the co-model. Using the approach of physical prototyping, investigating such parameters would naturally result in the construction of several system realizations, each having their own axle widths. In case several axle widths needed to be investigated, the development costs and time spent would quickly add up.
In general the value of co-modeling will be influenced by the characteristics of the system to be developed (production scale, types of test be performed etc.). Therefore, the usefulness of co-modeling with respect to a certain system, will determine the value of applying the development process. This motivates the discussion in section 6.2.2, where a comparison of co-modeling and physical prototyping is made.

Chapter 6

Methodological Extensions and Evaluation

In chapter 3, the development process and methodology to be evaluated were presented. Next, the application of the methodological guidelines during the Autonomous Robot System case study was described in chapters 4 and 5. Based on the findings of this thesis, this chapter suggests methodological extensions, and performs the final evaluation of the development process and methodology. Finally, chapter 7 concludes this thesis by presenting the achieved results, the future work and the personal learning outcomes.

6.1. Introduction

The point of this chapter is to provide suggestions for possible improvements to the development process and methodology. Throughout this chapter, the term *methodological extension* refers to a subject, which has not been explicitly included as part of the methodology, but might be useful to know about, when applying it. Some of the methodological extensions of this chapter will be directly related to the case study work, whereas others to a greater degree stem from the literature study of this thesis. In this chapter, these methodological extensions will be referred to as being "case study" and "literature" related, respectively. The purpose of introducing methodological extensions is not to suggest additional guidelines, but rather point out applicable inputs, which could be included as part of the in-depth description of the methodology. In this chapter, the DESTECS cases introduced in section 1.5.4 will be referred to as appropriate. The aim is to clarify certain points and discuss various aspects of the development process and methodology, using the different case characteristics.
This chapter first points out suggestions for methodological extensions in sections 6.2 and 6.3. Next, tool enabling extensions supporting the development process are described in section 6.4. Finally, the development process and methodology are evaluated in section 6.5.

6.2. Case study related methodological extensions

6.2.1 The value of a co-model

When using a co-model, the hope is that it will turn out to be a fruitful investment sooner or later during the development. However, if building a physical prototype of a system is sufficiently

cheap, creating a co-model may not necessarily be justifiable. To be able to benefit the most from a co-model, it is important to understand the strengths and limitations of it. This section discusses this particular topic, not explicitly being addressed by the methodology.

6.2.1.1 Reproducibility of tests

The reproducibility of tests is far from guaranteed, when working with a physical prototype over a co-model. This is a problem, if a re-run will be necessary for obtaining deeper insight into the outcome of a certain test. For example, if the data recordings from the previous run were insufficient. If the test required the physical destruction of some part of the system, then it has to be replaced, before the test can be repeated. If a slight difference in the replaced part influences the outcome of the test, then it may not be possible to reconstruct the results with sufficient fidelity. Reconstructing the test conditions may even be impossible, if the corresponding parameters (say temperature or humidity) are out of the tester's control.

6.2.1.2 Difficulty of fault realization

The difficulty of realizing faulty behavior does not only depend on the type of fault to be realized, but also the circumstances under which it is going to be activated. As an example, the realization of a faulty actuator such as a DC motor turning the wheel of a robot, may be as simple as disconnecting a cable. But what if the fault needs to be activated, when the robot is moving around? Or in the case of a printer, how would one realize a paper jam to test the fault handling of the system? Fault realization may even require destructive modifications to the system. Using the water tank case as an example, a leakage could be realized by creating a hole in the tank, which would then have to be repaired, if the tank was to be used for the analysis of other behaviors later on. Such modifications may be more appropriate to analyze initially at the co-modeling level, in order to save time and money. Still, it cannot be guaranteed that the response of a co-model, will be completely reflected in the behavior of the system realization. However, it may reduce the number of expensive test-runs.

6.2.1.3 Paper Pinch fault modeling

In [Pierce&11] it is shown how modeling of faults and fault tolerance mechanisms can be done using a co-model of the Paper Pinch system. A controller sets the speed of a motor driving the bottom roller of the Paper Pinch using a Pulse-Width Modulation (PWM) signal. In order to do this, the controller needs to know about the current speed of the motor, which is measured using encoders. To analyze the fault tolerance of the system, encoder drift and bit flip is modeled. The former may be caused by missing or adding encoder counts, whereas the latter may result in random jumps in the measured motor speed. The faults are implemented in each their own 20-sim sub-model separated from the remaining CT encoder model, to minimize the model pollution introduced by the fault modeling. Having several implementations of these sub-models, enables easy switching between different drift errors etc. For modeling of fault tolerance, the voter pattern and kernel pattern are used. These patterns have already been described in section 2.6. The kernel pattern is used for guarding against a sudden increase in motor speed, which is deemed unsafe. This mechanism is introduced to protect against bit flips. Additionally, the co-model is extended to include three encoders to demonstrate the application of the voter pattern. This pattern is used for guarding against both bit flips and encoder drifting. These two types of faults may be implemented to occur at specific points in time, and hence they will be fully reproducible between co-simulations. On the other hand, when working with a system realization, it may be a problem

to achieve the required fidelity for a test to be regarded as being reproducible. As an example, how would one ensure that the realization of a bit flip at a specific point in time is reproducible?

6.2.2 Physical prototyping vs. co-modeling

The Paper Pinch was originally introduced in the BODERC project [Heemels&07], where it was modeled using separate DE and CT models. Due to the opportunity of doing virtual system exploration, the modeling process saved many man years of work, and allowed skipping one complete physical machine-build iteration. Hence, co-modeling may also reduce the number of physical prototypes being constructed, without compromising the quality of the final realization of the system. For systems where several expensive physical prototypes need to be constructed it would be desirable, if part of the testing and Design Space Exploration (DSE) could be done at a virtual level, where it is cheaper and faster to do.
Another example is found in the Flare Dispensing System case, where the trajectory of the fares are visualized using a 20-sim 3D animation. Not only would a physical prototype of this system be very resource demanding to construct, the dispensing of flares due to analysis and testing, would only add to the costs. From a practical point of view, preparation for test may also be very time consuming, if equipment needs to be powered up and adjusted etc.

6.2.2.1 From co-modeling to domain specific activities

The question of when to start moving to the domain specific activities such as software and control engineering, arose during the case study development. And when these phases have been entered, what will the role of the co-model be?
First of all, it should be pointed out that having the system realization in place, should not exclude further maintenance of the co-model, if the resulting benefits outweigh the working costs. In fact this may bring several advantages, as pointed out in sections 6.2.1.1 and 6.2.1.2. To start a discussion on when to begin the domain specific activities, and the role of the co-model when doing so, five cases are considered below. The characteristics of the Autonomous Robot System and the Line Following Robot are similar, thus they are treated together.

Flare Dispensing System: If a system is expensive to produce, as in the case of the Flare Dispensing System, it may make sense to postpone the domain specific activities, until a *mature* co-model is available. As explained in section 3.2.1, this means waiting with the transitioning, until the co-model fulfills its purpose. For the Flare Dispensing System, co-modeling will be significantly cheaper to perform, compared to realizing the system. The co-modeling activity could then be used for obtaining a good understanding of the system and the upcoming domain specific activities, in order to minimize the amount of expensive rework. Still, if no data or measurements are available for determining the parameters of the co-model, one might question the fidelity of the co-simulation outcomes. In such cases, data from similar systems or back-on-the-envelope calculations must be used. For example, video recordings of various flare dispensing systems may be used for adjusting parameters, when performing sanity checks of the co-model.

Paper Pinch: A printer represents an example of a system, where physical prototyping is expensive. Furthermore, the development work may include the realization of several physical prototypes, for instance if the system is to serve as a platform in a product line: Each time a new physical prototype needs to be constructed, the costs of materials and assembling work will be recurring. In such a case, the virtual nature of a co-model will enable easier reuse of the physical components, which demonstrates another advantage of a co-model. Since the construction of a

printer is expensive, it would be appropriate to develop a mature co-model before moving to the domain specific activities. For the product-line example, the maintenance of the co-model representing the product platform, would enable cheap experimentation with new types of printers.

Water Tank: Physical prototyping for this particular system would most likely be in between the Flare Dispensing System and the Autonomous Robot System in terms of financial costs. If only one instance of the water tank was to be developed, and the costs of materials were small, then using a physical prototype would be appropriate. On the other hand, the water-tank may be mass produced in many shapes and sizes. For such a system, co-modeling may be useful for obtaining a good understanding of the system being developed, before starting the production. For instance by answering questions such as "what is the minimum hole size leading to the preferred water outflow, when opening the valve?". Hence, co-modeling could be used for minimizing the chances of costly error-corrections after production has begun. If other shapes of the tank need to be investigated, then this could be done by having a 20-sim sub-model with an implementation for each of the shapes to be analyzed.

Autonomous Robot System/Line Following Robot: If the system to be produced is fairly cheap or the number of physical prototypes to be produced is low, it may make sense to realize it gradually, as the co-model evolves. This has been the approach chosen during the Autonomous Robot System case study. Moreover, the co-model may be better suited for certain testing and analysis activities than the realization of the system, and vice versa. As an example, if the optimal spacing between two light sensors of the Line Following Robot were to be determined, it could potentially require a lot of tests to be performed. This would be very resource demanding to carry out, without making use of tools such as Automated Co-model Analysis (ACA). Although the optimal sensor spacing determined using co-simulation does not necessarily reflect the exact one of the system realization, this may still be useful for limiting the number of tests performed with it. For this particular example, the co-model is used for making predictions about the system realization, albeit at a higher level of abstraction. The system realization could then be confirming the outcomes of the ACA, for instance by performing test runs around the suggested optimal sensor spacing.
If the test and analysis work is less complex, such as determining the completion time of a route, it may be more appropriate to perform the test using the realization of the system. After all, a precise physical prototype can be expected to perform closer to the realization of the final system, compared to a co-model. A physical prototype of the system could also be used for increasing the fidelity of the co-model, by using real measurements for adjusting the design parameters etc. Hence, the co-model and physical prototype may supplement each other.

6.2.2.2 Investigating feasibility using co-modeling

The abstractions chosen for the Autonomous Robot System co-model, enable focusing on the important things such as the route following routine used by the decision controller. As an example, the device drivers and the algorithm for tracking the position of the robot, have been completely neglected during co-modeling. These tasks later turned out to be very time-consuming during the realization of the system. This demonstrates a major advantage of using a co-model: When using a physical prototype for demonstrating the feasibility of an idea, things such as device drivers need to be developed, if not already available. If the idea turns out to be infeasible, then all the effort spent on developing device drivers, assembling the system parts etc. may not be sufficiently rewarded. If the feasibility of the idea could be demonstrated using less costly co-modeling, then this approach certainly is more attractive.

For the Autonomous Robot System, getting a co-model to follow a route using the DESTECS tool-chain, has proven significantly easier, than getting the equivalent functionality working for the system realization. As an example, abstracting device drivers out of the co-model, reduced the time spent on co-modeling substantially. Furthermore, since the DE model was already specified in VDM, it was easily transformed into code to be executed on the target platform of the decision controller. Mainly because the development of the DE model, contributed considerably to the understanding of the decision controller behavior. Thus, the decision controller algorithms could be readily reused, which reduced the development time of the physical prototype significantly.

6.2.3 Incremental development approach

Realization work will not be included as part of an iteration, if the approach chosen is to develop a mature co-model, before moving to the domain specific activities. On the other hand, one may choose to gradually realize the system as the co-model evolves. The former approach will be referred to as *iterative co-modeling*, and the latter as *iterative realization*.
For the Autonomous Robot System iterative realization is appropriate, since functionality can be gradually introduced, in order to improve the route following capabilities of the robot. For instance by introducing another sensor. Hence the *fitness* of this particular system is considered good with respect to iterative realization. However, for the Flare Dispensing System the dispensing of flares due to incoming missiles, constitutes an important and complex function of the system. Although it may be broken down into several sub-functions, only together will they be able to do the dispensing of flares, and hence bring substantial value to the user of the system. Thus, iterative realization for this particular system is more difficult as compared to the Autonomous Robot System. Still, such a system could be developed by gradually introducing the intelligence of the system using iterative co-modeling.
One of the advantages of iterative realization is the rapid feedback. In the context of co-modeling, this could be measurements obtained from the system realization. It is possible that such data is needed, in order to reach the required fidelity of the co-model. When using iterative co-modeling alone, it may take a long time before such data is obtained. If data from similar systems is not available, it may be difficult to use the co-model for predicting properties regarding the system realization. Still, if the purpose of the co-model is to obtain insight into some part of the system at an abstract level, such as the controller algorithms, then the co-model may still serve its purpose.

6.2.4 Co-simulation performance

Co-modeling relies on co-simulation for analysis, which requires the co-model being executed. In general the performance of tool simulation is an important topic, which may limit the modeling choices. Two important factors influencing the performance of a tool simulation is the size of the design space to be investigated, and the abstraction level of the model being simulated. More specifically, if the size of the design space is very large, it may be computationally impractical to investigate all of it. Similarly, if the model being simulated is very detailed, then it may result in impractical simulation times. The consequences may be that the model is of no practical use, which ruins the point of maintaining one. This sub-section points out different ways of addressing these particular issues.

6.2.4.1 Making qualified abstractions

Co-simulation time heavily depends on the modeling abstractions made, and thus it is important to understand the common pitfalls and good practices of abstractions. This particular topic is

addressed by the DESTECS project in [Pierce&12b], where some suggestions are given on how to avoid unnecessary co-simulation overhead. The point is that making *qualified* abstractions, may contribute significantly to the overall co-simulation performance. A qualified abstraction helps obtaining the required insight into the system being developed, in order to fulfill the co-model purpose. First, one should try to minimize the frequency and amount of information traversing the co-simulation interface. Secondly, unnecessary co-model details should also be avoided, which is easier said than done.

Co-simulation interface: It may be the case that some of the identified constructs during the system decomposition are fit for both the DE and CT domains. If a Proportional-Integral-Derivative (PID) controller setting the speed of a DC motor is added to the CT model, then it may be that only the set point needs to traverse the co-simulation interface. On the other hand, having the PID controller on the DE side, requires the transfer of both actuator signals and sensor values. Thus, it may be appropriate to treat the PID controller as a CT construct during co-modeling, and as a DE construct during realization. However, one should carefully consider the model purpose when making such decisions, which may result in a co-model, not representing the system realization properly. Still, one should understand that a PID controller implemented in software indeed contributes to the workload of the controller executing it. If the PID controller is treated as a CT construct during co-modeling, and this overhead is not accounted for in the DE model, then the co-model may not reflect the system realization properly. Furthermore, making such an abstraction may prevent obtaining the insight needed to move to the domain specific activities. If understanding how to implement a PID controller is important, then using a PID sub-model as provided by 20-sim, may be inappropriate. As another example, a DC motor of the Autonomous Robot System realization uses two separate signals for controlling the rotational speed and direction of the corresponding wheel. However, at the co-modeling level the motor direction is changed by regulating the sign of the controlled variable representing the speed signal. This leads to less information traversing the co-simulation interface, which contributes to the performance of the co-simulation.

Details of a co-model: Adding too much detail to the co-model also increases the co-simulation time. The modeling of a sensor may be done in details, where things such as sampling, scaling and Analog-to-Digital Conversion (ADC) is being taken into account. Alternatively, the easier way would be to simply send the value directly across the co-simulation interface. This particular example may not contribute a lot to the performance of the co-simulation, but may avoid unnecessary complication of the co-model. One should keep in mind that complex models are more difficult to maintain.

6.2.4.2 Applying local simulation

Sometimes the model analysis can be done sufficiently well using only a DE or CT formalism, although this approach is likely to lead to overly simplified domain constructs. In such situations there is no need for information traversing the co-simulation interface, which may result in very fast simulation times. This technique was applied during the case study work for performing sanity checks of the CT-model using a 20-sim 3D animation. Although this technique is commonly used for performing initial model analysis, it may still be useful as the co-model evolves and details are added to it.

6.3. Literature related methodological extensions

Some of the suggestions given throughout this section, can be seen as extensions for the tool-oriented approach of the methodology, influenced by the DESTECS project. Other inputs are more general in the sense that they would be equally applicable across tool-chains.

6.3.1 Real-time constraints

The development process focuses on the development of embedded systems, which sometimes must obey to specific timing properties. However, the methodological guidelines do not provide any suggestions on how this can be incorporated into the Systems Modeling Language (SysML) descriptions or the co-model. This sections discusses how timing properties can be dealt with, when applying the development process.

6.3.1.1 Modeling timing properties at the system-level

Modeling of timing properties can be done in SysML using either the constraint, sequence or requirements diagram. Additionally, the SysML traceability features may supplement the modeling of timing properties, by associating them with system components using the SysML *satisfy* relation. However these timing properties would still rely on informal descriptions. A stronger notation for such non-functional properties may be found in the MARTE UML profile[1]. An interesting thing about MARTE is that it enables the modeler to annotate the models with information, which can be used for performing scheduling or performance analysis etc.

6.3.1.2 Validating co-model system-level timing properties

The work in [Fitzgerald&07b] presents an extension to VDM-RT, enabling the validation of system-level timing properties. *Validation conjectures* are descriptions of temporal relations between system-level events, which can be evaluated over the execution trace, described in section 2.4.2. Later this work has been extended in [Ribeiro&11], to include run-time validation. Predefined standard forms of validation conjectures have been defined, directly supporting the validation of a deadline or separation between two events, called the *trigger* and the *response*. The trigger event could be the press of a button, and the corresponding response may be the update of a display. From the standard forms, more specific validation conjectures can be constructed. Violations of the validation conjectures can be visualized using an extension for the showtrace plugin. This plugin was previously covered in section 2.4.2. The presented techniques are however meant for rapid feedback on the system-level timing properties, and the passing of a conjecture cannot be seen as a proof of correctness.

6.3.2 Design space exploration

The methodology specifically advocates the use of tools for performing DSE. More specifically, it is pointed out that ACA and parameter sweeping may be useful for automating this task. This section points out other ways of automating DSE, which is not explicitly mentioned in methodology.

[1] MARTE UML profile: http://www.omgmarte.org

6.3.2.1 Comparing alternative electronic system architectures

It is possible that many electronic system architectures exist (number of CPUs and buses etc.), which fulfill the system requirements. The identification of such electronic system architectures, may constitute a critical part of the development work, especially if the system is to be mass produced. The work of Lausdahl et al. [Lausdahl&11] enables automated exploration of different electronic system architectures, without the modeler dealing with the manual work of changing the **`system`** class, like in a traditional VDM-RT setting. The suggested approach can be used for investigating different deployment parameters such as CPU and bus capacity automatically. This may be useful, if replacement of components with cheaper equivalents is of importance. In order to evaluate an electronic system architecture, but still avoid manual inspection work, the technique mentioned in section 6.3.1.2 for automatic checking of validation conjectures, may be applied. Still, further analysis of system metrics such as performance and cost, may be needed for choosing the final electronic system architecture.

6.3.3 Organizing design related information

A view model defines a coherent set of views, each showing different aspects of a system. As an example, the CAFCR methodology presented in section 3.4.1 introduced a five view structure, used for capturing the relationship between the customer and the product. As pointed out in section 3.4.2, the methodologies suggested by the BODERC project and Wolff constitute commercial and scientific approaches, respectively. Regardless of the approach, it may be necessary to manage vast amounts of design related information.

6.3.3.1 Common approaches

The use of view models is not limited to, but has been widely used within the field of software engineering, for the construction of software architectures. The proper choice of view model(s) used for managing design related information, will be influenced by many factors such as the type of stakeholders involved in the project, and the characteristics of the system being developed (e.g. mission critical). As an example, if the purpose of a single view is to enable the division of work, it may arrange the system into software packages to be developed by separate teams. A well-known example is the “4+1” view model [Kruchten95], used for describing the architecture of software-intensive systems. However, one may argue that the software oriented approach of this view model, is not very fit for the multi-disciplinary nature of embedded systems. The “+1” represents the scenarios, which drive the identification of architectural elements, similar to the approach suggested by the development process applied during the case study work. The Five Views of Architecture [Douglass&02b] is another view model suggested by the Rapid Object-Oriented Process for Embedded Systems (ROPES). This view model introduces a “Subsystem & Component” view, which relates to the initial part of the system decomposition, suggested by the development process. This view identifies the large-scale pieces of the system, and the association among them. The view model further identifies the “Deployment” and “Safety and Reliability” views, which relate to topics not explicitly addressed by the development process and methodology. Still, they might be worthwhile investigating already at the co-modeling level. Here “Deployment” refers to the mapping of software architecture onto physical devices such as processors (see section 6.3.2.1). On the other hand, “Safety and Reliability” relates to the topic of fault modeling, described in section 2.6. The “4+1” and The Five Views of Architecture view models do not emphasize the need for describing the CT constructs, which is needed for the construction of a co-model. Thus these approaches are not optimal for managing co-modeling

information.

6.3.3.2 Organizing co-modeling information

A co-model representing an embedded system, is an abstract representation of a system realization. Although it really depends on the purpose of the co-model, this often causes communication protocols and device drivers to be postponed to later phases of the project. During the early phases of a project the focus should be understanding the system to be developed, determine a proper partitioning of the DE and CT construct etc. This should be reflected in a way in which the design related information of the co-model is organized. A suggestion for how the information identified during the application of the development process could be organized, is given in the five view structure description below:

Scenarios view: Captures the co-model purpose, use cases and requirements of the system. This view is special, in that it drives the identification of elements in the remaining views.

Subsystems and Partitioning view: Identifies the role of relevant sub-systems and the association among them. This view also covers the partitioning work of the identified sub-system elements, represented as blocks in the SysML descriptions.

Domain Constructs and Contract view: Covers the descriptions of the DE and CT constructs using the types of diagrams, suggested by the development process. Interactions among the elements of different domains also belong to this view.

Fault modeling view: Describes the faults addressed by the co-model, and how fault tolerance mechanisms have been applied, to enhance the resilience of the system.

Distribution view: Describes the electronic system architecture of the controller, and the mapping of DE constructs onto processing elements.

The artifacts resulting from the application of the development process, should cover the first three views. The remaining two are not explicitly addressed by the methodology. Still, the related tasks may constitute critical parts of the development work. As a minimum, they should be considered during the co-modeling activities, where the cost of corrections are lower as compared to later project phases. Investigating different electronic system architectures is tool-enabled as mentioned in section 6.3.2.1, whereas modeling of faults is supported by the DESTECS methodology (see section 2.6).

6.4. Tool enabling extensions

The approach of the development process heavily relies on the use of tool-support for co-modeling, co-simulation and DSE. Hence, enabling tool extensions are likely to significantly improve the value of applying the development process. This section describes the tool improvements resulting from the work of this thesis, but also points out possible improvements for the existing DESTECS tool-chain.

6.4.1 Break-analysis

The identification of optimal sensor spacing between the light sensors of the Line Following Robot demonstrates an example, where ACA may be useful. However, situations may occur during an

ACA run, where the robot completely looses track of the line. If the optimal sensor spacing is regarded as the one leading to the smallest completion time of a track, then it may be desirable to skip and ignore the outcome of some of the co-simulation runs under certain conditions. Such a "skip" will subsequently be referred to as a *break*. Otherwise every co-simulation will need to run to completion, which may be a waste of time. Looking at the Line Following Robot in particular, one may require for the co-simulation not to break that "a maximum of 5 seconds may pass between line detections". For simplicity the line is assumed to be detected initially. To break a co-simulation run of an ACA, some notation is needed for describing the criterion. For this particular example, this could be formulated using validation conjectures. More specifically, for the co-simulation not to break, the trigger and the response event, representing the last and next detection of the line, are required to meet a deadline of 5 seconds. Although the formulation of break criteria is enabled using validation conjectures, it is not possible at the current time of writing to apply this technique during an ACA. However, the tool-chain may be extended later to include this feature. Similarily, a technique for breaking a co-simulation is also found in 20-sim using the `stopsimulation` function, and thus it is possible to stop the co-simulation from the CT model, when using the DESTECS tool-chain.

6.4.2 The Real-Time Log Viewer plugin

The current version of the showtrace plugin publicly available is currently working with a textual representation of the execution trace. However, the file representing the execution trace tends to reach very large sizes for complex executions. The main problem is that working with a textual representation, is not a scalable solution. As part of this thesis, the showtrace plugin has been extended to work with a binary representation of the execution trace. This new version of the plugin will be referred to as the *Overture Real-Time Log Viewer* (RTLV) plugin. Still, a textual representation of the execution has the advantage over the binary representation of being human-readable. For this reason, and to retain backward-compatibility, the old way of visualizing execution traces, will still be present in the upcoming release of Overture. The binary representation reduces the size of the execution trace file significantly, but also improves the time it takes to parse it before visualizing it, thus leading to more efficient DE model analysis. Details on the design, implementation and performance of the RTLV plugin, can be found in appendix A.

6.5. Evaluation

In chapter 1 the three evaluation categories *advantage*, *relevance* and *effectiveness* were presented. *Advantage* determines the value the development process and methodology bring, which are impossible or difficult to obtain using traditional approaches to embedded system development. *Relevance* judges whether the approach taken by the development process is suited for different types of systems (e.g. mission-critical and non-mission-critical). Finally, *effectiveness* judges the use of enabling tools and techniques for increasing the productivity of the development work.

A summary of the evaluation is provided in table 6.1, listing the *possibilities* and *challenges* identified for the development process and methodology. Possibilities can be thought of as potential benefits, while challenges are potential drawbacks resulting from the application of the development process. Challenges which have been treated by the methodological extensions, are marked with an asterisk ("*"). Throughout this section, identified possibilities and challenges will be written in *italic*. They are listed in table 6.1 according to the order in which they appear in the text. The evaluation is performed by discussing each of the evaluation categories separately.

Table 6.1: Summary of the evaluation. The possibilities and challenges listed in the table are identified and described throughout this section

Evaluation summary	
Possibilities	**Challenges**
Cheaper and reproducible tests	Sufficient co-model fidelity
New fault analysis opportunities	Maintenance of two systems
Property prediction	Tool-chain assumptions
Reduction of physical prototypes	Learning of tools and formalisms
Automated DSE	Risks and resource limitation
Globally optimal system solutions	*Choosing realization approach
Improved stakeholder collaboration	*Organizing design related information
Cheaper feasibility studies	
Tool-chain freedom	
Light-weight development	
Gradual introduction of complexity	
Easier domain specific transitioning	
Convenient methodological reference	

6.5.1 Evaluation category 1: Advantage

Most of the advantages identified for the development process and methodology are related to the virtual nature of a co-model. Others result from collaborative modeling and use of heterogeneous and holistic modeling notations.

6.5.1.1 Using a co-model

Whether the best approach to development is co-modeling or physical prototyping, depends on the characteristics of the system to be developed, and the value a co-model may bring. As argued in sections 6.2.1.1 and 6.2.1.3, the virtual nature of co-modeling is likely to cause *cheaper and reproducible tests*, and enhance the *fault analysis opportunities*. To benefit from these possibilities, a co-model of *sufficient fidelity* is required. This adds extra overhead to the development work, as it requires the *maintenance of two system representations*. One way of obtaining fidelity is by adjusting the parameters of the co-model using estimates or data measurements from the system realization. When the co-model reaches the required level of fidelity, it supports *prediction of properties* concerning the system realization. An example of this was given in chapter 5, where the Autonomous Robot System co-model was used for predicting route completion times for the system realization. However, the process of ensuring a high level of fidelity for the co-model is a time-consuming and difficult task, and therefore it remains one of the main challenges of co-modeling.

Traditional approaches to embedded system development often require the construction of several physical prototypes. As explained in section 5.4.8.3, this is likely to be the case if the feasibility study involves DSE, where physical dimension such as width and height are involved. In such cases it would be desirable if at least part of the DSE could be performed virtually, in order to save time and money (see section 6.2.2). A co-model makes it possible to minimize the design space to be investigated, which potentially lowers the costs of physical prototyping. Thus, a co-model of sufficient fidelity can be used for *reducing the number of physical prototypes* as described in section 6.2.2. *Automated DSE* tools like ACA may be of help for this task.

6.5.1.2 Collaborative development

The methodology emphasizes the importance of treating development as a collaborative task. During the model purpose and requirements phase, this was useful for reaching a common understanding of the system to be developed. Similarly, the descriptions of the domain constructs served well for cross-disciplinary knowledge transfer. This helped ensuring the compatibility among the different domains, which is required for the system to fulfill the overall system objectives.

In general, such activities help raising awareness of how changes may take effect across domains. The hypothesis is that this contributes to a *globally optimal solution*, as argued in section 1.2.2. To demonstrate the importance of collaborative development, section 6.2.1.3 describes how bit flips were guarded against in the Paper Pinch DE model. If a DE expert was not aware of the underlying problem, then a protection mechanism, such as the kernel pattern, might have been left out of the controller realization. This could potentially lead to unsafe controller behavior, harmful to the physical parts of the system.

6.5.1.3 Heterogeneous and holistic modeling notations

In general the notations used for describing the documentation artifacts worked well as enabling tools. Initially, informal descriptions were used for reaching a common agreement concerning the co-model purpose as well as the system definitions and assumptions. Thus, understanding the notation did not require any special prerequisites, which also allows non-technical stakeholders to participate on the same terms as everyone else.

Most of the SysML descriptions rely on a notation closely related to that of UML. Thus, they are likely to be understood by people within the DE domain, as well as CT experts working on the edge of software development (e.g. electrical or robotics engineering). Still, most SysML diagrams rely on intuitive notations, and therefore they are likely to serve their purpose of facilitating *collaboration among the different stakeholders*. Even when the participating stakeholders have no prior experience with UML modeling. As an example, the Block Definition Diagram (BDD), used for describing the hierarchical structure of the system, was well understood by the participants of the case study. This diagram was useful for reaching a common understanding of the system, and served well as a check-list for the following partitioning activity. In general, there was a good connection between the activities and the produced artifacts, the latter serving as input for the work following. This was useful, as it contributed positively to the traceability of the system descriptions.

The most significant issue concerning the suggested modeling notations was encountered, when the physical laws and dependencies were to be described using constraint blocks and the parametric diagram. As explained in section 4.4.2, this was caused by the circular dependencies of the wheel forces. Therefore, the parametric diagram was hard to read due to the many interrelations represented in the diagram. Instead the physical descriptions of the system were made using informal sketches, and a classical mathematical notation. These sketches served well for discussion among the domain representatives.

6.5.2 Evaluation category 2: Relevance

One important factor affecting the relevance, is the opportunity of using different tool-chains. Another is the scientific approach of the methodology as described in section 3.4.2.

6.5.2.1 Abstractions and tool-chains

During the case study a high level of abstraction was applied to the tracking system, data logging and device drivers. This significantly reduced the time spent on co-modeling, and helped avoiding introducing unnecessary complexity to the development work. For mission-critical systems this is a desirable ability, as it helps ensuring that the critical system aspects are paid the required amount of attention. In addition to this, co-modeling also supports *cheaper feasibility studies*, as the most time-consuming tasks potentially can be completely neglected (see section 6.2.2.2).
The type of tool-chain being used determines the possibilities of applying abstractions. As described in section 2.7, some tool-chains base their DE model on SystemC (C++), while DESTECS relies on an abstract mathematical notation (VDM). This *tool-chain freedom* makes it possible to apply the development process for a wide range of systems. Still, some *tool-chain assumptions* are made by the development process and methodology, which may become a challenge. As an example, the use of a 3D animation during the case study work, was directly suggested by the methodological guidelines (see section 5.2). In general, this cannot be assumed to be supported by a tool-chain.
The *learning of a new tool-chain* and CT formalism during the thesis was a challenge, which influenced the productivity of the development work negatively. However, this is more of a one-time challenge, as the experiences gained using a tool-chain are likely to be transferable to future projects. Mainly because the crucial steps such as setting up the communication between the two domain models and creating a 3D animation will be similar. Still, it is being considered a challenge, as the unfamiliarity with new tool-chains is likely to cause reluctance to co-modeling.

6.5.2.2 Development approach

The methodology provides no support for management of *risks and resource limitation*. Generally it does not introduce as much overhead to the development work as compared to the BODERC methodology (see section 3.4). Therefore it is regarded as being *lightweight*, which may be an advantage for one-time projects involving Research & Development (R&D). For these projects, feasibility is often of paramount importance, while risk and resource limitation is not paid much attention. For bigger projects where huge costs are at stake, this may not be the best way of approaching embedded system development.
The methodology neglects guidance for *choosing a realization approach*. More specifically, no guidance is provided for choosing between developing a mature co-model before realizing the system, or doing it gradually together with the co-model. This motivated the work of section 6.2.3, where different approaches to incremental development are discussed. Similarly, the methodology did not suggest any way of *organizing design related information*, which resulted in a suggestion for a view model in section 6.3.3.2.

6.5.3 Evaluation category 3: Effectiveness

The transitioning between the different activities such as moving from the SysML descriptions to co-modeling, is affecting effectiveness. The same applies to the overhead introduced by the methodology.

6.5.3.1 From SysML descriptions to co-modeling

The descriptions of the DE constructs enabled easy transitioning from SysML to VDM, since both notations support the Object-Oriented (OO) concepts, and almost every modeling construct

of UML has a VDM equivalent. The process of describing the DE constructs is also assisted by the methodological DESTECS guidelines [Broenink&12b], where patterns for structuring of the DE model are presented. Understanding such general shapes enables reuse across projects, which contributes positively to the productivity of the development work. As an example, the state pattern was be used for describing the transitioning between the different states of the Autonomous Robot System state machine (see section 4.5.1).
When transitioning from the SysML descriptions to 20-sim, the guidelines provided a natural way of structuring the resulting CT model into sub-models. In general these intermediate SysML descriptions enable a systematic approach for the construction of the two domain models. Alleviating the process of transitioning to the domain models is particularly useful, in case the modeler only has little experience using the chosen tool-chain. In general this means that the intermediate SysML descriptions can be seen as a way of *gradually introducing complexity*.

6.5.3.2 From co-modeling to realization

The most important outcome of co-modeling, concerning the transitioning to the domain specific activities, is the insight obtained into the system being developed. During the case study work VDM modeling was used for developing the decision controller algorithms. The produced VDM model could easily be translated into code for the target platform, thus reducing the time spent on system realization (see section 6.2.2.2). Therefore co-modeling also *facilitates the transitioning to the domain-specific activities*, by contributing to a higher degree of confidence in the system design.

6.5.3.3 Value of guidelines

Capturing the essence of the methodology in the form of guidelines as presented in chapters 4 and 5, allows for *convenient methodological reference* throughout development. During the case study the guidelines served well as input for the development work, since the enforcement of this kind of structure helps ensuring that important system aspects are dealt with in timely manner. Furthermore, the number of guidelines is kept at an appropriate level, not to make the application feel cumbersome and artificially enforced. Also, introducing too many guidelines would potentially hamper the creativity of the development work. In this way, the in-depth description of the methodology can be referred to as appropriate.

Chapter 7

Concluding Remarks and Future Work

In chapter 3 the development process and methodology to be evaluated were presented. In chapters 4 and 5 this development process was applied during the Autonomous Robot System case study. Based on the thesis findings, chapter 6 provided the final evaluation of the development process and methodology, but also contributed with suggestions for methodological extensions. In this chapter, the achieved results are presented and related to the thesis goals, defined in chapter 1.

7.1. Introduction

This thesis has demonstrated how collaboration supports the development of embedded systems, where the knowledge required for their successful development spans multiple domains. This collaborative aspect is motivated by the need to manage the design complexity of embedded systems, caused by the inherent engineering disciplines. This approach differs from traditional physical prototyping, where these disciplines are treated separately, thus opposing a globally optimal solution. Hence, the increasing design complexity of embedded systems demands for new development processes and methodologies, for guiding the development work. This thesis has also demonstrated how a co-model supports embedded system development, by offering new ways of performing system analysis. For example, investigating faulty system behavior often involves destructive modifications to the system, while testing of a system realization requires a lot of preparation. This naturally causes an increase in the money and time spent on working with a physical system prototype. These costs can be significantly reduced, by performing some of the system analysis virtually, where it can be done cheaper and faster.
This chapter returns to the initial thesis goals in section 7.2. Next, the personal learning outcomes and the achieved results are presented in sections 7.3 and 7.4. This is followed by the future work in section 7.5. Final remarks conclude this thesis in section 7.6.

7.2. Returning to the thesis goals

The initial goals of this thesis presented in chapter 1 were:

G1: To evaluate the development process and methodology [Wolff12] for co-models suggested by Wolff, through case study development. The result of this work will provide inputs for potential changes to the development process and methodology.

G2: To extend and suggest new ways of using the current tools and techniques for collaborative modeling, supporting the application of the development process.

G3: To improve personal skills concerning researching a scientific topic and disseminating the findings.

Throughout this chapter a critical approach will be adopted, when assessing the achievement of these goals. This assessment, and the questions fostered by the resulting thesis work, together form the future work of this thesis. Accomplishing the future work tasks, will contribute further to the achievement of the thesis goals.

7.3. Personal learning outcomes

The most important personal learning outcomes of this thesis, are directly related to the personal improvement in researching a scientific topic. Other learning experiences have contributed to the author's confidence in using multi-disciplinary models, for reasoning about system-level properties.

7.3.1 Co-modeling skepticism

The most convincing co-model related experience encountered during the case study, was the opportunity to neglect the most time-consuming tasks such as device driver development. The fact that this resulted in only a small amount of time being spent on co-modeling compared to realization, has surpassed the author's initial skepticism to the development process and methodology. Skepticism to modeling is quite common, and thus good arguments for using a co-model are needed, in order to obtain the joint support of stakeholders.
When using traditional physical prototyping, the system realization is tangible evidence of progress for every stakeholder involved in the project. The same does not apply to a co-model, which at best can be demonstrated using a 3D animation. Although a 3D animation enables visualization of the underlying co-model, this will hardly be as convincing an argument as a physical prototype. Therefore, non-technical stakeholders (which often comprise the decision makers in a project) will be difficult to impress, if all they see is extra time being spent on analysis. The challenge is to convince them that the cost of developing and maintaining a co-model, will be outweighed by the resulting benefits in the long run. Thus, it is necessary to make sure that the value a co-model may bring the development work, is communicated properly. Otherwise, it will be difficult to obtain joint support for using a co-model.

7.3.2 Acquiring new information

For research to be successful, it is important to be good at judging the relevance of information, when performing a literature study. This is a major difference from traditional lecturing, where the relevant information is normally provided by the lecturer. During the writing of this thesis, being responsible for judging the quality and relevance of the studied literature, was an unaccustomed and challenging experience. Especially, because studying every little detail is not possible, given the amount of information available. During the first weeks of the thesis work, too much time was spent on diving into the details of the papers read, which influenced the initial productivity negatively. The lesson learned from this experience suggests getting an overview, before spending time on the details. Often this means briefly reading the abstract and introduction of a paper,

and use this for judging whether it is worth further reading. For bigger pieces of work, like the BODERC project introduced in section 3.4, the approach chosen was to find other references summarizing the work (see section 7.4.3). The author becoming better at acquiring new scientific information during the thesis work, has contributed to the achievement of the third thesis goal.

7.3.3 Learning a new tool-chain

Although the creation of a co-model requires the acquaintance with a tool-chain, the experiences gained using it, are likely to be transferable to future projects. The reason for this, is that most of the crucial steps carried out when creating a co-model, will be similar the next time. The concepts of creating a co-simulation contract, setting up communication between the DE and the CT models etc., remain the same. Hence, the productivity of developers applying co-modeling is likely to increase significantly, as experience with a certain tool-chain is gained. On the other hand, when a physical prototype is used for investigating the feasibility of a project, it is possible that things such as device drivers need to be developed, as pointed out in section 6.2.2.2. The crucial part of programming device drivers, is to understand the underlying workings of the hardware. Unfortunately, this task is often very time-consuming, and it has to be repeated every time a device driver for a new piece of hardware must be developed. As hardware is likely to change between projects, the experience of developing a device driver, is not as easily transferable to future projects.

7.4. Achieved results

Before applying the development process, the tools and technologies targeting multi-disciplinary system modeling were studied. Based on the findings, it was decided to use the DESTECS tool-chain for the case study work. This choice was mainly based on the abstraction and visual validation techniques, this tool-chain is offering. Time was spent on learning the tool-chain, the associated CT formalism and terminology, in order to get prepared for the forthcoming co-modeling activities.
The theoretical study also covered the field of development processes and methodologies addressing multi-disciplinary concerns, described in chapter 3. Although this field was found to be rather novel, an interesting piece of work was found in the BODERC project. The BODERC methodology differs from that supporting the development process evaluated as part of this thesis, as it pays great attention to the customer-product relation, as well as management of risk and resource limitation. Studying the BODERC project contributed to a deeper understanding of the complexity and need for managing embedded system development. This approach demonstrates other possibilities and challenges, not characterizing the development process and methodology evaluated as part of this thesis. Therefore, it has served well as a reference point throughout the thesis.
The description below briefly describes why the thesis goals are considered to be achieved. Next, the achievement of these goals are assessed in more detail in the sub-sections following.

G1 achievement: The evaluation work highlights the possibilities and challenges of the development process and methodology (see section 6.5). Together with the methodological extensions in chapter 6, this serves as input for potential changes to the development process and methodology.

G2 achievement: The methodological extensions provide suggestions for use of tools and techniques for co-modeling, but also contributes with the Real-Time Log Viewer (RTLV) plugin

(see section 6.4.2). Furthermore, the proposed view model considered to be the main result of this thesis, will be published (see sections 7.4.2.3 and 7.5.1).

G3 achievement: The writing of this thesis, and the acquiring of new information during the literature study, has been an instructive experience. This has significantly contributed to the author improving his personal skills in researching a scientific topic (see section 7.3.2).

7.4.1 G1: Evaluation of development process and methodology

To obtain practical experience for the evaluation, the development process was applied during the case study, described in chapters 4 and 5. In these chapters, the most important observations made, have been captured as retrospective sections. The application covered all the suggested activities, resulting in a co-model and system realization, the former being used for predicting route completion times for the latter. In general the evaluation of the development process and methodology is considered to be positive. The reason for this, is that the development process and methodology successfully contributed to the development of the Autonomous Robot System.

7.4.1.1 The evaluation work

The evaluation work contributed to the identification of several possibilities and challenges as listed in table 6.1. Many of these were directly associated with the use of a co-model, which also distinguishes the development process and methodology from traditional approaches to embedded system development. Others were caused by the collaborative activities and use of the supporting methodology. Understanding the possibilities and challenges of applying the development process, helps choosing wisely between this approach and traditional physical prototyping. Still, this choice is influenced by the characteristics of the system being developed, which should be understood before making the decision (see section 6.2.2.1).

7.4.1.2 Limitations of the case study

Due to the time-frame of this thesis, other interesting aspects of embedded system development, were omitted during the case study work. For example, the work did not involve any analysis of timing properties, nor exploration of different electronic system architectures. These aspects would have been interesting to include, as the associated activities remain common challenges in embedded system development. Analysis of timing properties is supported by the showtrace plugin, whereas the work of Lausdahl et al. [Lausdahl&11], enables automated exploration of different electronic system architectures (see sections 6.3.1.2 and 6.3.2.1).
Another point of criticism of the thesis work, is the development setting of the case study, which is influenced by the thesis being the work of a single author (see section 1.5.3). Although the robotics engineer was participating during the initial white-board aided sessions, co-modeling was carried out by the author alone. Naturally, this minimizes the risk of misunderstandings leading to unforeseen cross-disciplinary consequences, as the collaborative aspect is non-existing. In the context of the thesis subject this is unfortunate, as it does not reflect the development setting, which the development process and methodology address.

7.4.2 G2: Suggestions for methodological extensions

To improve on the current methodology, suggestions for methodological extensions are provided in chapter 6. These extensions describe the advantages and disadvantages of co-modeling, which determine the value of applying the development process (see section 6.2.1). Examples include

reproducible testing and possibilities of investigating faulty system behavior, which would otherwise be difficult to do using a system realization.

7.4.2.1 General appeal

To make the methodological extensions appeal to a wider range of systems, the cases introduced in section 1.5.4, were taken into consideration. These cases served as inspiration for a discussion, supporting the choice between physical prototyping and co-modeling (see section 6.2.2). Assuming a co-modeling approach, the methodological extensions also provided guidance for when to move to the domain-specific activities, as well as choosing a realization approach (see section 6.2.3). This led to the definition of the concepts *iterative co-modeling* and *iterative realization*. The former is the development of a co-model which fulfills its purpose before starting the realization work, while the latter is gradual realization as the co-model evolves.

7.4.2.2 Tool extensions

Some of the methodological extensions are directly based on the DESTECS tool-chain. For example, the analysis of timing properties using the showtrace plugin, and the technique for exploration of different electronic system architectures. In addition to this, a new generation of the showtrace plugin has been developed, called the RTLV plugin (see appendix A). It now works with a binary representation of the execution trace, as opposed to a textual representation in the previous version. The binary representation not only reduces the size of the execution trace file significantly, it also improves the time it takes to parse it, thus leading to a more efficient DE analysis. However, the tool related methodological extensions are directly dependent on the use of the Overture platform, which naturally limit their general use. Still, they may serve as inspiration for extensions to other tool-chains, as they support some of the development activities, which are of general concern (e.g. timing analysis).

7.4.2.3 View model proposal

The methodological extensions also suggest a five view structure for organizing design related information (see section 6.3.3.2). The author is not aware of other proposals for view models targeting co-model development, which makes this a particular interesting outcome of the thesis work. View models have proven useful in for instance software engineering, for managing design related information. Since there is a need for managing design related information during co-modeling as well, the view model proposal will potentially bring value to the development work. Therefore, it is being considered the main result of this thesis.
The Inspiration for this proposal was found in the BODERC project, which suggests using the CAFCR methodology, for structuring of design related information. However, the five view structure suggested in the methodological extensions clearly differs from the structure used by the CAFCR methodology. Mainly because the suggestion made as part of this thesis reflects the collaborative activities of the development process, and assumes the presence of a co-model.

7.4.3 G3: Researching a scientific topic

The work of this thesis is the result of the author's first experience in researching a scientific topic. It has been a great challenge, as the relevant information not only has to be found, it must also be disseminated concisely in the form of this thesis. As an example, the BODERC project referenced throughout this thesis, comprise a major piece of work. Although a lot of time was spent on

studying it, only minor parts have contributed to the methodological extensions and the evaluation work. The approach taken to get an overview of the BODERC project, was to first study the work of Verhoef [Verhoef08]. Then the original source of the BODERC project was referred to as appropriate, when detailed insight was needed. This was a useful strategy for acquiring new knowledge, and thus it was used throughout the thesis.
In the beginning of the thesis work, the biggest challenge was determining the proper level of detail with respect to the written work. Initially, the theory was exhaustively covered in chapter 2, which later changed to include only what was necessary, for the reader to continue to the next chapter. The same applies to the draft version of chapter 4, which did a comprehensive coverage of the physical dynamics of the Autonomous Robot System. Later chapter 4 was changed, by moving details of less significance to an appendix. The improvements on the written work is the result of supervisory feedback, which significantly contributed to the conciseness of the writing.

7.5. Future work

During the thesis work, areas lacking further research have been identified. Some relate directly to the need for tools and techniques supporting the co-modeling activities of the development process, whereas others concern the aspects of embedded system development, not covered during the case study work.

7.5.1 Submission of article

An article disseminating the main result of this thesis (the view model proposal), will be submitted for the 10th International Conference on integrated Formal Methods (iFM 2013), also hosting the 11th Overture Workshop. The article submission is intended to bring feedback on the suggested view model, and disseminate the findings, for others to make use of them. This contributes to the second and third thesis goals, as it extends the current tools and techniques for co-modeling, and it involves disseminating research findings.
The use of view models have been widely accepted in for instance systems engineering and software engineering, for comprehending system complexity. A view model targeting embedded system development already exists, but it reflects traditional approaches (see section 6.3.3.1). The development process and methodology assume the presence of a co-model, and thus call for a different way of organizing design related information.

7.5.2 Case study

The points of criticism concerning the case study work identified in section 7.4.1, make it interesting to consider applying the development process again under different circumstances. Below are some suggestions for things to address in a future application, which were not covered during the case study work. If these suggestions are carried out, the resulting work will provide input for additional evaluation work (first thesis goal). In addition to this, it may also serve as inspiration for further work regarding tools and techniques for co-modeling (second thesis goal).

Uncovered aspects: The case study work did not cover activities such as timing analysis and exploration of different electronic system architectures. As these activities constitute common challenges in embedded system development, it would be relevant to include them in future case study work.

A more realistic development setting: It would be interesting to apply the development process again under different circumstances, reflecting a more realistic development setting, than the one used throughout the case study. Ideally, the collaborative work among the DE and CT representatives, should be existing during all phases of the development process.

Co-model fidelity: In general many things could have been done differently to improve the fidelity of the co-model, during the validation and verification phase described in chapter 5. For instance, by performing more calibration iterations, introduce more detail into the co-model, use more sophisticated equipment (for robot parts, measuring etc.) and so on. Doing these things in a future application of the development process, may reveal further challenges with respect to the calibration process, not identified during the case study work.

Trying other tool-chains: The case study work only represents a single instantiation of the development process, as the DESTECS tool-chain was being used. It would be interesting to investigate the application of the development process using a different tool-chain such as Ptolemy (see section 2.7.1). This may reveal further issues with respects to the tool-chain assumptions addressed in section 6.5.2.1.

Using the proposed view model: To determine the value of the proposed view model, it could be included in a future application of the development process. However, a view model will likely bring most value to the development work, in case many design related artifacts are produced, by several collaborating people. Therefore, the Autonomous Robot System case study, and the development setting under which it has been carried out, will most likely not create the ideal circumstances for this.

7.5.3 Automation: SysML to bond graphs

During the application of the development process, many intermediate artifacts such as the System Modeling Language (SysML) descriptions resulted from the suggested activities. One way of increasing the productivity in such cases, is to use automation tools to alleviate some of the manual work, involved in moving between the different notations.
As pointed out in [Wolff12], some of the concepts of SysML have equivalents found in 20-sim, which could be exploited by automating the transitioning process. The mapping between the SysML notation and the bond graph formalism, has already been investigated in [Turki&05]. In this work, an extension to the activity diagram is mapped to the bond graph formalism. However, no mapping exists to support the SysML Internal Block Diagram (IBD), used for describing the CT constructs during development, and therefore this area lacks further research. On the other hand, the mapping between the Unified Modeling Language (UML) and VDM is well-understood, and automated transitioning between the two notations already exists [Lausdahl&09].
This future work suggestion primarily favors the DESTECS tool-chain, and relates to the second thesis goal of extending the current tools and techniques for co-modeling. If the results reveal good opportunities for a SysML to bond graphs mapping, then this could also be adopted by other tool-chains using the bond graph formalism.

7.6. Final remarks

The first thesis goal of evaluating a development process and methodology for co-models, has been successfully met. The resulting work serves two main uses. First, it provides inputs for

potential changes to the development process and methodology. Secondly, it offers guidance for potential users in choosing between a co-modeling approach and traditional physical prototyping. The work also contributes with methodological extensions, successfully fulfilling the second thesis goal concerning extending and suggesting new ways of using the current tools and techniques for co-modeling. These extensions also intend to support direct users of the methodology in understanding the advantages, as well as disadvantages, of using a co-model. Some methodological extensions provide general guidance equally applicable across the different tool-chains, whereas others assume the DESTECS tool-chain instantiation.

The results of this work required a lot of time being spent on studying tools, techniques, methodologies etc. addressing the multi-disciplinary concerns of embedded system development. This has been an instructive experience, contributing to the author improving his skills in researching a scientific topic. More specifically, the author has become better at acquiring new information, and writing the findings concisely. Thus the third thesis goal has been successfully met.

Some of the identified future work tasks are related to the current tools and techniques for co-modeling. Therefore, these will contribute to the achievement of the second thesis goal. In addition to this, the submission of the article also contributes to the achievement of the third thesis goal, as it involves disseminating some of the research findings of this thesis. Finally, the suggested case study activities not covered during this thesis, intend to provide input for further evaluation work, and additional methodological extensions. This will contribute to the achievement of the first and second thesis goals.

To conclude this thesis, co-modeling supports overcoming common challenges of embedded system development, which are difficult to address using traditional approaches. Collaborative development further supports the development work, by serving a useful tool for managing the inherent design complexity of embedded systems. However, in order to successfully overcome existing challenges, as well as the new ones introduced by co-modeling, new development processes and methodologies are needed. These will provide guidance to the development team, making sure that the right questions are being asked in a timely manner, before rework has to be done. Hopefully, this thesis can add to the evidence needed, in order to get collaborative development exploited increasingly in industrial development.

References

[Amerongen10] Job van Amerongen. *D*ynamical Systems for Creative Technology. Controllab Products, Enschede, Netherlands, 2010. [cited at p. 14]

[Black&04] David C. Black and Jack Donovan. *S*ystemC: From The Ground Up. Springer, first edition, 2004. 244 pages. [cited at p. 17]

[Boehm88] B. Boehm. A Spiral Model of Software Development and Enhancement. *I*EEE Computer, 21(5):61–72, 1988. [cited at p. 3, 20]

[Broenink&10] Broenink, J. F. and Larsen, P. G. and Verhoef, M. and Kleijn, C. and Jovanovic, D. and Pierce, K. and Wouters F. Design Support and Tooling for Dependable Embedded Control Software. In *P*roceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems, pages 77–82, ACM, April 2010. [cited at p. 5]

[Broenink&12b] Jan F. Broenink and John Fitzgerald and Carl Gamble and Claire Ingram and Angelika Mader and Jelena Marincic and Yunyun Ni and Ken Pierce and Xiaochen Zhang. *M*ethodological Guidelines 3. Technical Report, The DESTECS Project (INFSO-ICT-248134), October 2012. [cited at p. 15, 16, 70]

[Douglass&02b] Douglass, Bruce Powell. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Addison-Wesley Longman Publishing Co., Inc., 2002. [cited at p. 64]

[Eker&03] J. Eker and J.W. Janneck and E.A. Lee and Jie Liu and Xiaojun Liu and J. Ludvig and S. Neuendorffer and S. Sachs and Yuhong Xiong. Taming Heterogeneity – the Ptolemy Approach. *P*roc. of the IEEE, 91(1):127–144, January 2003. [cited at p. 16]

[Fitzgerald&05] John Fitzgerald and Peter Gorm Larsen and Paul Mukherjee and Nico Plat and Marcel Verhoef. *Validated Designs for Object–oriented Systems*. Springer, New York, 2005. [cited at p. 13, 14]

[Fitzgerald&07a] John Fitzgerald and Peter Gorm Larsen and Shin Sahara. VDMTools: advances in support for formal modeling in VDM. page , Submitted for publication 2007. pages. . [cited at p. 14, 54]

[Fitzgerald&07b] John Fitzgerald and Peter Gorm Larsen and Simon Tjell and Marcel Verhoef. *Validation Support for Distributed Real-Time Embedded Systems in VDM++*. Technical Report CS-TR:1017, School of Computing Science, Newcastle University, April 2007. 18 pages. [cited at p. 63]

[Fitzgerald&10a] John Fitzgerald and Peter Gorm Larsen and Ken Pierce and Marcel Verhoef and Sune Wolff. *Collaborative Modelling and Co-simulation in the Development of Dependable Embedded Systems*. Technical Report CS-TR-1213, School of Computing Science, Newcastle University, July 2010. [cited at p. 6]

[Fitzgerald&11] John Fitzgerald and Peter Gorm Larsen and Ken Pierce and Marcel Verhoef. A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems. *T*o appear in Mathematical Structures in Computer Science, 2012. [cited at p. 13]

[Fowler&03] Martin Fowler and Kendall Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2003. 185 pages. [cited at p. 11]

[Friedenthal08] Sandford Friedenthal, Alan Moore, Rick Steiner. *A Practical Guide to SysML*. Morgan Kaufman OMG Press, Friendenthal, Sanford, First edition, 2008. ISBN 978-0-12-374379-4. [cited at p. 11]

[Gamma&95] E.Gamma, R.Helm, R.Johnson and J.Vlissides. *D*esign Patterns. Elements of Reusable Object-Oriented Software. Volume of *A*ddison-Wesley Professional Computing Series, Addison-Wesley Publishing Company, edition, 1995. 395 pages. . [cited at p. 36, 87]

[Goodwin&01] Goodwin, Graham C. and Graebe, Stefan F. and Salgado, Mario E. *Control System Design*. Prentice Hall, September 2001. [cited at p. 32]

[Heemels&07] Maurice Heemels and Gerrit Muller. *Boderc: Model-Based Design of Hightech Systems*. Embedded Systems Institute, Den Dolech 2, Eindhoven, The Netherlands, second edition, March 2007. [cited at p. 2, 6, 24, 59]

[Henzinger&07] Tom Henzinger and Joseph Sifakis. The Discipline of Embedded Systems Design. *I*EEE Computer, 40(10):32–40, October 2007. [cited at p. 2]

[Horowitz&89] Horowitz, P. and Hill, W. *The Art of Electronics*. Cambridge University Press, second edition, July 1989. [cited at p. 33]

[Kruchten95] Philippe Kruchten. Architectural Blueprints – The 4+1 View Model of Software Architecture. *I*EEE Software, 12(6):42–50, November 1995. [cited at p. 7,

64]

[Larsen&09] Peter Gorm Larsen and John Fitzgerald and Sune Wolff. Methods for the Development of Distributed Real-Time Embedded Systems using VDM. *I*ntl. Journal of Software and Informatics, 3(2-3), October 2009. [cited at p. 13, 14]

[Larsen&10a] Peter Gorm Larsen and Nick Battle and Miguel Ferreira and John Fitzgerald and Kenneth Lausdahl and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. *A*CM Software Engineering Notes, 35(1):, January 2010. 6 pages. [cited at p. 7, 13]

[Larsen&10b] Peter Gorm Larsen and Kenneth Lausdahl and Nick Battle. *The VDM-10 Language Manual*. Technical Report TR-2010-06, The Overture Open Source Initiative, April 2010. [cited at p. 13]

[Lausdahl&09] Kenneth Lausdahl and Hans Kristian Agerlund Lintrup and Peter Gorm Larsen. Connecting UML and VDM++ with Open Tool Support. In *F*ormal Methods 09, Springer-Verlag, November 2009. LNCS-5850. [cited at p. 77]

[Lausdahl&11] Kenneth Lausdahl and Augusto Ribeiro. Automated Exploration of Alternative System Architectures with VDM-RT. In *9*th Overture Workshop, June 2011, Limerick, Ireland, 2011. [cited at p. 14, 64, 74]

[Muller04] Gerrit Muller. CAFCR: A Multi-view Method for Embedded Systems Architecture; Balancing Genericity and Specificity. 2004. [cited at p. 24]

[Nicolescu&06] G. Nicolescu and F. Bouchhima and L. Gheorghe. CODIS – A Framework for Continuous/Discrete Systems Co-Simulation. In C. G. Cassandras and A. Giua and C. Seatzu and J. Zaytoon, editors, *A*nalysis and Design of Hybrid Systems, pages 274–275, Elsevier, 2006. [cited at p. 17]

[Pierce&11] Ken Pierce and John Fitzgerald and Carl Gamble. Modelling Faults and Fault Tolerance Mechanisms in a Paper Pinch Co- model. In *P*roceedings of the ERCIM/EWICS/Cyber-physical Systems Workshop at SafeComp 2011, Naples, Italy (to appear), ERCIM, September 2011. [cited at p. 6, 58]

[Pierce&12a] Pierce, K. G. and Gamble, C. J. and Ni, Y. and Broenink, J. F. Collaborative Modelling and Co-Simulation with DESTECS: A Pilot Study. In *3*rd IEEE track on Collaborative Modelling and Simulation, in WETICE 2012, IEEE-CS, June 2012. [cited at p. 6]

[Pierce&12b] Ken Piece and John Fitzgerald and Carl Gamble and Yunyun Ni and Jan F. Broenink. *Collaborative Modelling and Simulation — Guidelines for Engineering Using the DESTECS Tools and Methods*. Technical Report, The DESTECS Project (INFSO-ICT-248134), September 2012. [cited at p. 62]

[Qualen&12] von Qualen, Mads and Askov Andersen, Martin. *A Methodology for Transforming Java Applications Towards Real-Time Performance*. Master's thesis, Aarhus University School of Engineering, December 2012. pages. [cited at p. 85]

[Ribeiro&11] Augusto Ribeiro and Kenneth Lausdahl and Peter Gorm Larsen. Run-Time Validation of Timing Constraints for VDM-RT Models. In *9*th Overture Workshop, June 2011, Limerick, Ireland, 2011. [cited at p. 63]

[Scherz&00] Scherz, Paul. *Practical Electronics for Inventors*. McGraw-Hill/TAB Electronics, first edition, April 2000. [cited at p. 32]

[Turki&05] Skander Turki and Thierry Soriano. A SysML Extension for Bond Graphs Support. In *P*roc. of the International Conference on Technology and Automation (ICTA), Greece, 2005. [cited at p. 77]

[Verhoef08] Marcel Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. PhD thesis, Radboud University Nijmegen, 2009. [cited at p. 20, 76]ISBN 978-90-9023705-3

[Wolff12] Sune Wolff. Methodological Guidelines for Collaborative Modelling — Managing Heterogeneous System Complexity. *S*ubmitted for proposal — Journal of Software and Systems Modeling, 2012. [cited at p. 3, 6, 19, 20, 32, 35, 71, 77]

Appendices

Appendix A

Overture Real-Time Log Viewer

A.1. Introduction

This appendix provides additional details about the *Real-Time Log Viewer* (RTLV) plugin for the Overture tool (see section 6.4.2). Throughout this appendix *"the old RTLV plugin"* refers to the version of showtrace publicly available, while *"the new RTLV plugin"* refers to the re-design of it, developed as part of this thesis. This plugin has been developed in collaboration with Mads Von Qualen and Martin Askov Andersen, who provide a similar version of this appendix in their Master's thesis [Qualen&12].
The new design of the RTLV plugin, and the performance gain resulting from it, are described in sections A.2 and A.3 .

A.2. Design

When executing a VDM-RT model, a series of events are logged and timestamped by a component of the Overture tool called the *RT Logger*. During execution, these events are triggered by various actions such as function calls, object creations and thread activations. So far these events have been written to a text file (the trace file), understandable to humans. However, these files potentially contain immense sizes of up to thousands of lines. As a consequence, this makes it difficult for a human to use them for getting an overview of the entire execution. The RTLV plugin (both the old and the new) facilitates this problem, by enabling graphical visualization of these events. It offers three different view types for inspecting the logged events:

Architecture Overview: This is a simple overview of the CPUs and buses comprising the system, executing the model. From this view, it is possible to see how CPUs are connected via different communication buses.

Execution Overview: This is a detailed overview of the CPUs and buses illustrating thread swaps, operation calls and communication across the buses.

CPU Overview: This is a detailed view, provided for each CPU of the system. From this view, it is possible to inspect how threads execute code contained within different objects, and how they block on synchronous bus communication etc.

Within these views, it is possible for the user to scroll through all the logged events, and inspect the execution, at a particular point in time. This makes the trace file analysis much more manageable.

A.2.1 The old vs. the new design

The old approach of writing and reading the text files, containing the event data, is illustrated in figure A.1.

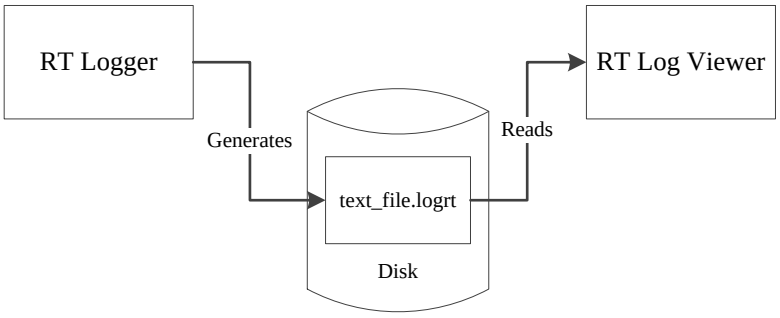


Figure A.1: Old RT Log Viewer

The problem with this approach is the performance of it, which also motivated the RTLV plugin re-design. Using the old approach, the time it takes to read the large text files from the disk is not only slow. The data structure representing the events and their relations, is also time-consuming to process, prior to the drawing of the different views. In addition to this, the code doing the processing of all the events, was very inefficient. These issues resulted in a bad user experience, as the entire Overture tool would stall for minutes, before showing the overviews. In worst case, it would even result in the Overture tool crashing.
Therefore, the RT Logger and the RTLV plugin needed a re-design, in order to become efficient. Before the work of this thesis commenced, the RT Logger was re-designed, and re-implemented. This new implementation produces an Object Oriented (OO) data structure, used for drawing the different overviews efficiently. However, the RTLV plugin was not yet able to parse this data structure, and display the data graphically. Therefore, this functionality was implemented as part of this thesis. The new and more efficient approach to writing and reading events, is illustrated in figure A.2.

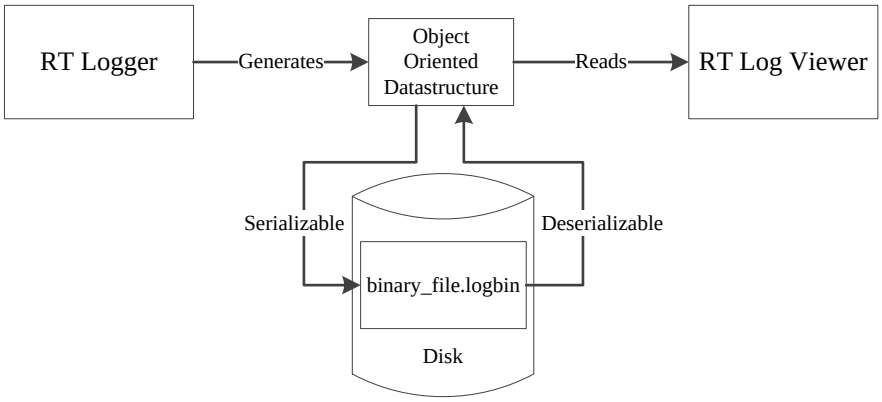


Figure A.2: New RT Log Viewer

The new data structure can be serialized and deserialized in binary. This makes it possible to save this data structure to a file, or keep it in memory, in order to visualize it using the RTLV plugin immediately.

A.2.2 Detailed design of the new RT Log Viewer

The new RTLV plugin design promotes a clear separation of concerns, and loose coupling between classes. This is done by defining three layers (as Java packages): *data*, *draw* and *view*. Each layer has its own area of responsibility, e.g. the classes within the *draw* layer, are responsible for drawing information specific to the user interface. The design is illustrated in figure A.3. Classes inheriting from the `TraceViewer`, are part of the *draw* layer. The *data* layer is composed of classes, inheriting from `EventHandler` and `TraceData`. Finally, all top-level classes, such as the `VdmRtLogEditor`, are part of the *view* layer. This architecture makes it easier to extend the RTLV plugin with additional functionality, as well as changing the data representations, drawing functionality etc.
The approach taken by the old RTLV plugin design, was to iterate through all events, when the trace file was loaded. This allowed the RTLV plugin to save the state of each data item (CPU, bus, threads etc.) at each point in time, but introduced severe performance overhead, when loading the trace file. Instead the new design loads the binary file, and parses only the visible amount of events. However, when the user moves the inspection to another point in time, the RTLV plugin must determine the current state of each data item, at that specific time. This requires processing of all events, for that specific data item, up until the given time. For example, a thread may be active or inactive, based on events which occurred prior to the current time. To accommodate this, the new design saves all state information in a series of classes, managed by the `TraceData` class (`TraceThread`, `TraceCPU` etc.). The classes inheriting from the `EventHandler` class, are part of a *strategy pattern* [Gamma&95], where the active event-handler is changed, based on the specific event being processed. The iteration and processing of events, are controlled by the `TraceFileRunner` class.
The event-handlers process the current event, update the corresponding data item (`TraceCPU`, `TraceBus` etc.), and invoke the required drawing functions through instances of `TraceViewer`. These are also based on the strategy pattern, where the active strategy is changed based on which view is selected (CPU, architecture or execution).

A.3. Results

The impact on the load time of the RTLV plugin overview, resulting from the new design, is illustrated in figure A.4.
It can be seen how the load time of the old RTLV plugin grows non-linearly, as the number of events to display increase. The line representing the load time of the new RTLV plugin, can be hard to see, as its load time is constantly low. This is due to the new design, which only parses and draws a small amount of events, when the RTLV plugin is loaded. The old approach was to load all events, when initiating the RTLV plugin. A disadvantage of the new approach, compared to the old one, is that scrolling and moving inspection between different events can exhibit longer load times, as the new RTLV plugin loads the events as needed. However, practical experience has shown that this relatively small increase in load time, does not inhibit the user experience of the plugin.

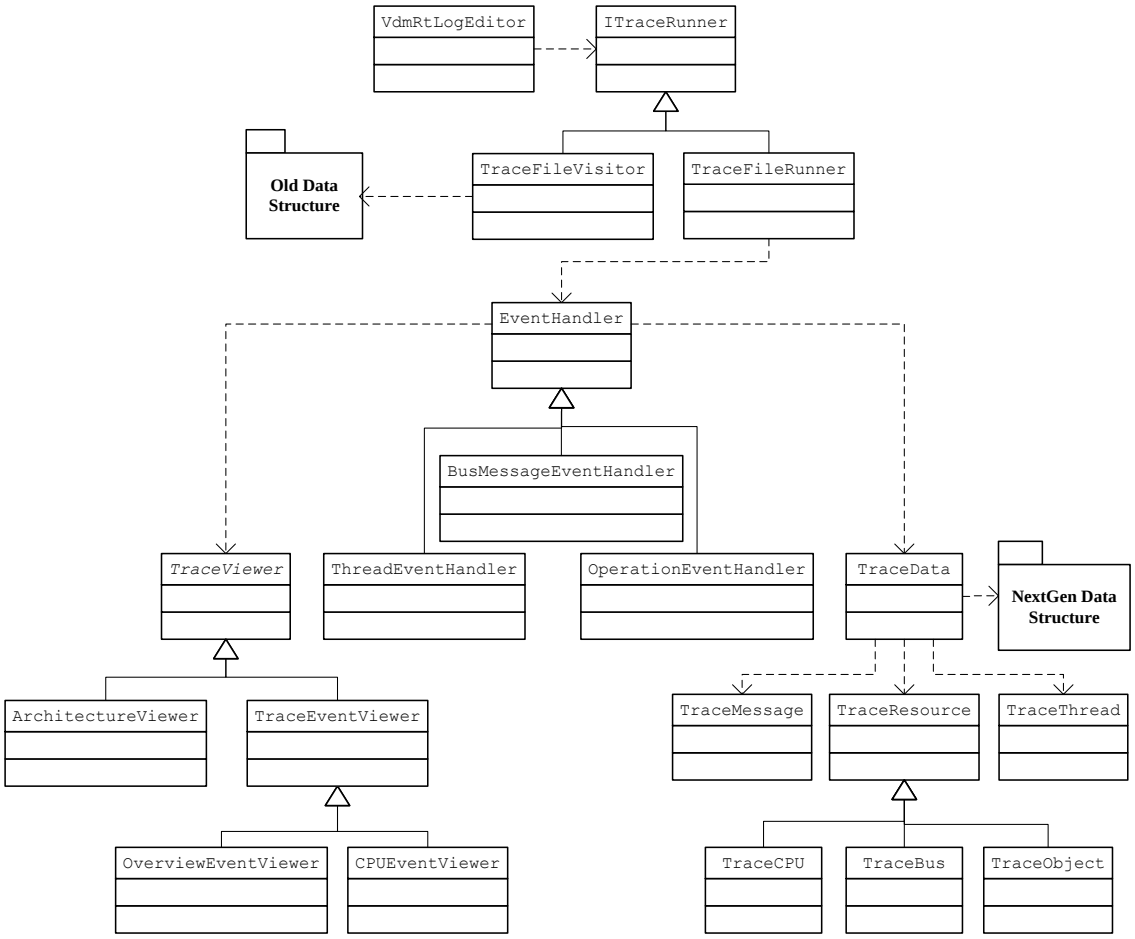


Figure A.3: Class diagram of the new RT Log Viewer design

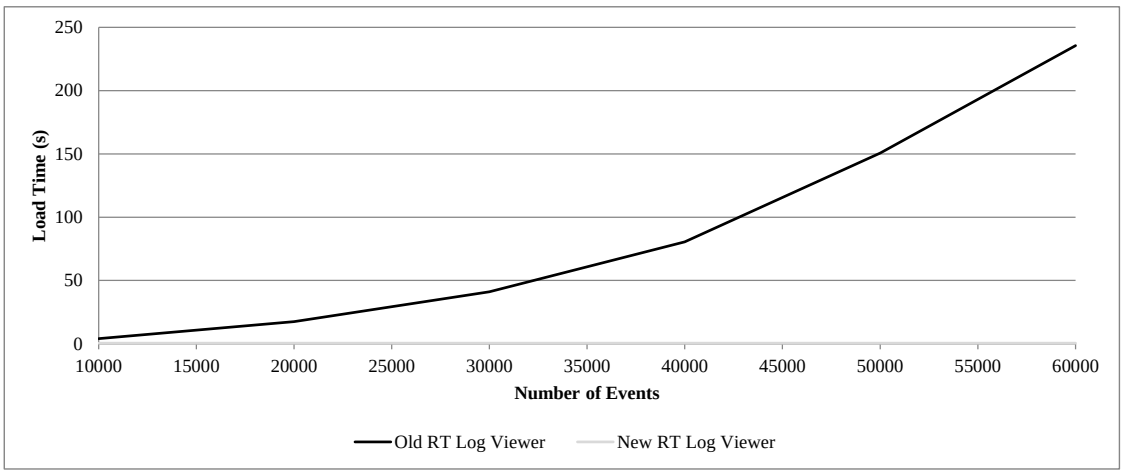


Figure A.4: New vs. old RT Log Viewer load times

Therefore, the new version of this plugin heightens the usability by an increase in performance and load times. The design of the plugin has been changed to support a clear separation of logic concerns to promote reuse and additional optimizations in future releases.

Appendix B

Case Study Details

B.1. Introduction

This appendix provides the reader with extra insight into the development work of the Autonomous Robot System case study, presented in chapters 4 and 5. Details on the co-model and the system realization will be described. Furthermore, this appendix also provides data on the following of the "box" and "arrow" routes, not presented in chapter 5. The co-model and the source code for the system realization of the Autonomous Robot System, can be found on the enclosed CD.

First details on the CT and DE constructs are provided in sections B.2 and B.3. Then additional parameters identified during the system decomposition are listed in section B.4. Finally, realization related information is provided in section B.5.

B.2. System decomposition: CT constructs

B.2.1 Robot structure

The IBD shown in figure B.1 highlights the structure of the robot. The motor-encoder-wheel configuration is completely symmetric for every wheel. Therefore, only the two rear wheels (along with motors and encoders) are shown. It is important to note that the driving capabilities of the robot will be influenced by how and where the wheels are connected. Not only the coordinates of an attachment are important, the orientation must also be taken into account. Therefore this attachment is described using the `Attachment` type. For the gyroscope it suffices to describe the attachment by a two-dimensional vector, since it measures the robot orientation. One special thing to note about the gyroscope is that it is attached to the body of the robot by an additional atomic port. The reason for this is that the gyroscope gets its input directly from the moving body of the robot and the estimate of the robot orientation can be determined directly from this external torque. When choosing where to attach the gyroscope one should keep in mind that the torque gets smaller when approaching the center of rotation of the robot, which may reduce the precision of the gyroscope. The decision controller makes all the choices regarding the driving, which is done by analyzing the different sensor outputs.

The motors cause the wheels to turn by applying a rotational force. Putting the wheels in contact with the environmental surface will cause the motions to be transformed from the rotational to the translation domain. This makes the robot move around.

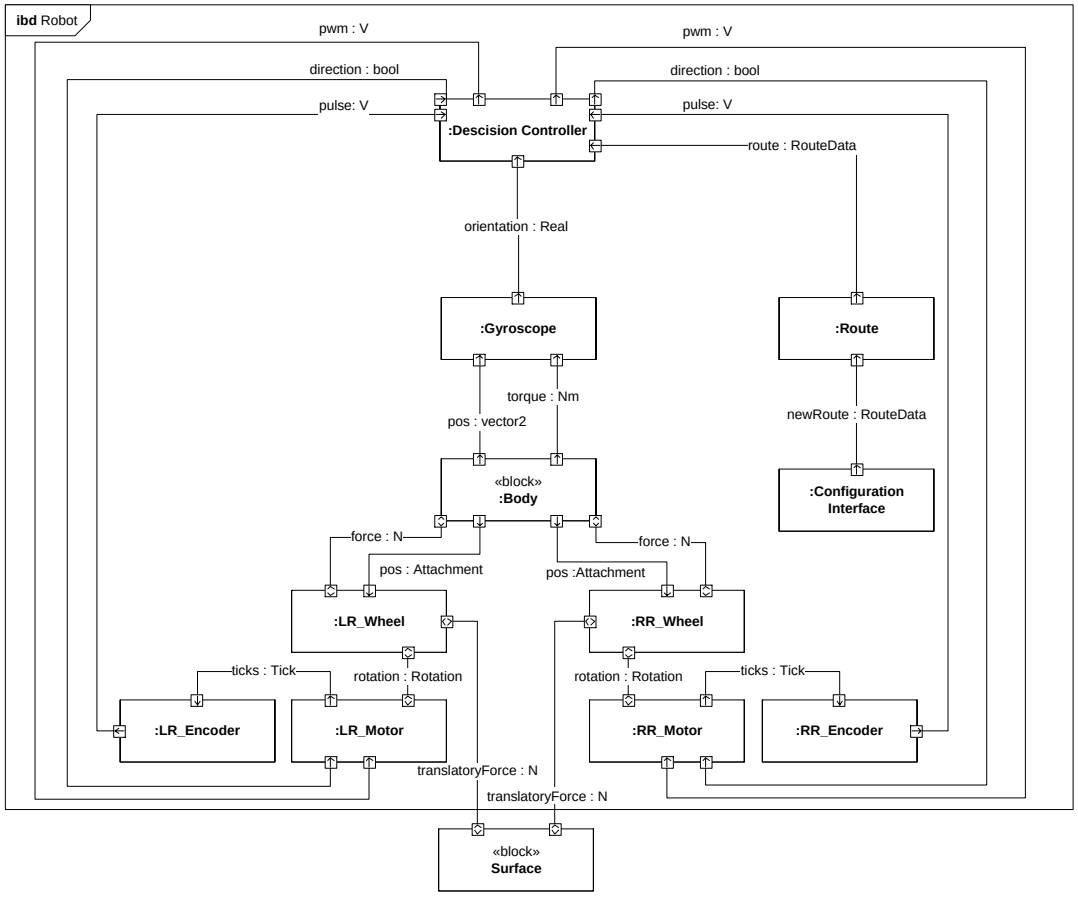


Figure B.1: The internal structure of the robot

B.2.2 The wheels

In order to make the electrically powered robot move, a power conversion from the electrical to the mechanical domain is made by a DC motor. This conversion can be described by the following equations:

$$T = K_m i \tag{B.1}$$

$$u = K_m \omega \tag{B.2}$$

where *i* is the current flowing through the motor, *T* is the torque, and K_m denotes the conversion factor which is DC motor specific. Similarly, the voltage *u* is proportional to the the angular velocity ω produced by the motor. Note that it is assumed no energy is lost due to dissipation. Next, the rotation of the wheel is converted into a translational motion when put in contact with the surface. Ideally, this conversion can be described as:

$$T = rF_{gen} \tag{B.3}$$

Here *r* denotes the radius of the wheel. Substituting the torque expression into equation (B.3) allows writing the generated force F_{gen} as follows:

$$F_{gen} = \frac{K_m i}{r} \tag{B.4}$$

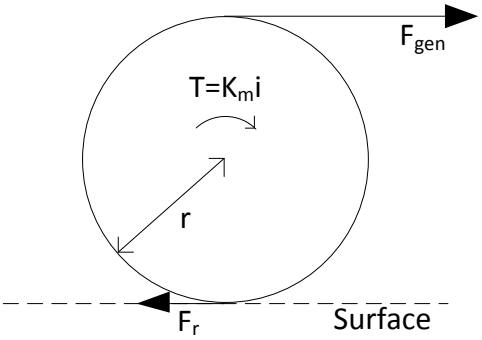


Figure B.2: Sideways view of the robot wheel

However, due to rolling resistance F_r the effective force F_e generated by a single motor is better described as follows:

$$F_e = F_{gen} - F_r \tag{B.5}$$

The forces acting on a single wheel covered so far are shown in figure B.2. For this particular model, the forces contributing to the vertical and horizontal deformation of the wheel have been neglected. The rolling resistance may be considered as being proportional to the normal force acting on the wheel. Hence (B.5) can be written as:

$$F_e = F_{gen} - C_{rr}F_N \tag{B.6}$$

where C_{rr} is the dimensionless rolling resistance coefficient. Finally, by combining (B.6) and (B.4), F_e is obtained as a function of the current through the DC motor i:

$$F_e(i) = \frac{K_m}{r}i - C_{rr}F_N \tag{B.7}$$

B.2.3 The robot

In order to be able to built a robot model for simulation, it is necessary have some understanding of the forces acting on the robot as a whole. Every wheel will produce a force which acts on the body of the robot, and tries to pull it in some direction. In section B.2.2, this force was denoted F_e and it is produced entirely by the motor itself. However, things get more complicated when the remaining wheels are taken into account. Not only will the resulting force from a single wheel be affected by the effective force being produced by the motor, it will also be influenced by the resulting forces of the other wheels. This is shown in figure B.3. Here F_e is the effective force generated by the motor, and F_{rx} denotes the force produced by the other wheels. By adding these two forces the resulting force F_{res} can be written as:

$$F_{res} = F_e + F_{rx} \tag{B.8}$$

Note that this constitutes a circular dependency among the different wheels. In order to be able to distinguish between the resulting forces of the wheels, the corresponding position will be used as subscript. Hence the resulting force of the left front wheel will be denoted F_{LF} and so on. An illustration of these forces and the entire robot as seen from above, is shown in figure B.4.

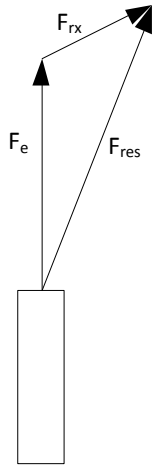


Figure B.3: Forces acting on a single robot wheel

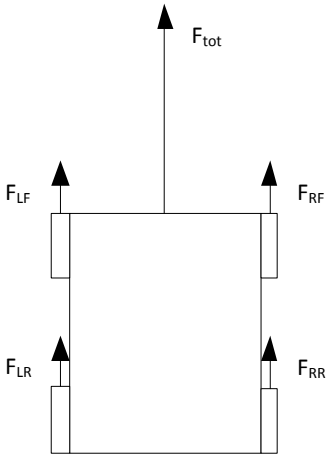


Figure B.4: The robot seen from above

As indicated in the figure the total force acting on the robot is the sum of the resulting wheel forces:

$$F_{tot} = F_{LF} + F_{RF} + F_{LR} + F_{RR} \tag{B.9}$$

So considering specifically the left front wheel, the received force $F_{rx_{LF}}$ produced by the other wheels can be calculated as:

$$F_{rx_{LF}} = F_{RF} + F_{LR} + F_{RR} \tag{B.10}$$

and the resulting force of the left front wheel F_{LF} is given by:

$$F_{LF} = F_{e_{LF}} + F_{rx_{LF}} \tag{B.11}$$

where $F_{e_{LF}}$ denotes the effective force produced by the left front motor. Note that this is just a concrete case of (B.8). By combining with (B.10) the resulting force can finally be written as:

$$F_{LF} = F_{e_{LF}} + F_{RF} + F_{LR} + F_{RR} \tag{B.12}$$

B.2.3.1 The robot orientation

Another important thing with respect to the simulation of the co-model, is the robot orientation θ. In order to determine this, figure B.5 shows the robot as seen from above, with the center of rotation positioned at the origin. Note that the front and rear wheels are not positioned symmetrically around this point. From the figure it can be seen that distances from the origin to the center of the wheels can be described as in (B.13), (B.14), (B.15) and (B.16) .

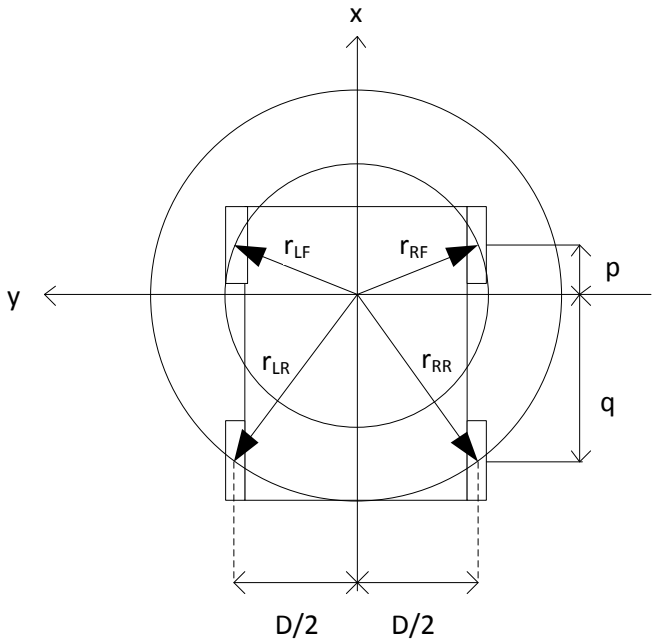


Figure B.5: The robot shown in the robot coordinate system

$$r_{LF} = (p,\ D/2) \tag{B.13}$$
$$r_{RF} = (p,\ -D/2) \tag{B.14}$$
$$r_{LR} = (-q,\ D/2) \tag{B.15}$$
$$r_{RR} = (-q,\ -D/2) \tag{B.16}$$

The total torque T_{tot} contributing to the rotation of the robot can be obtained by adding the torques produced by the wheels individually as shown in (B.17):

$$\begin{aligned} T_{tot} &= T_{LF} + T_{RF} + T_{LR} + T_{RR} \\ &= r_{LF} \times F_{LF} + r_{RF} \times F_{RF} + r_{LR} \times F_{LR} + r_{RR} \times F_{RR} \end{aligned} \tag{B.17}$$

By integrating the total torque and dividing this by the rotational inertia of the robot J the angular velocity ω is found.:

$$\omega = \frac{1}{J} \int T_{tot}\, dt \tag{B.18}$$

Finally the angular position θ can be obtained by integrating the angular velocity and adding the initial angular position θ_0:

$$\theta = \int \omega\, dt + \theta_0 \tag{B.19}$$

B.2.3.2 Simulation state

The state S describing the robot position and orientation is formed by the two-dimensional position vector x in the x-y plane and the orientation around the vertical z-axis θ. More formally:

$$S = (x, \theta) \tag{B.20}$$

B.3. System decomposition: DE constructs

B.3.1 Route following routine description

To complete a route, the robot needs to visit all the ways points in the correct order. The routine used for this, is visualized in figure B.6.
The robot starts out in the orientation state (`a` and `c` in figure B.6), where it starts spinning around its center of rotation until the target orientation, which will set it towards the next way point, has been reached. The robot chooses the spinning direction that will require the least amount of spinning, in order to reach the target orientation as fast as possible. Now the robot enters the navigation state (`b` and `d` in figure B.6), where it starts driving in a straight line until the target way point is assumed to be reached. The routine powers the motors symmetrically, and thus assumes that they share the exact same properties as explained in the system assumptions (see section 4.2.1). This process of alternating between the orientation and navigation states continues as long as there are more way points to be visited. When the robot reaches the final way point of the route it enters the stopping state (`e` in figure B.6), which terminates the state machine. This routine leads to the idealized behavior seen in figure B.6.

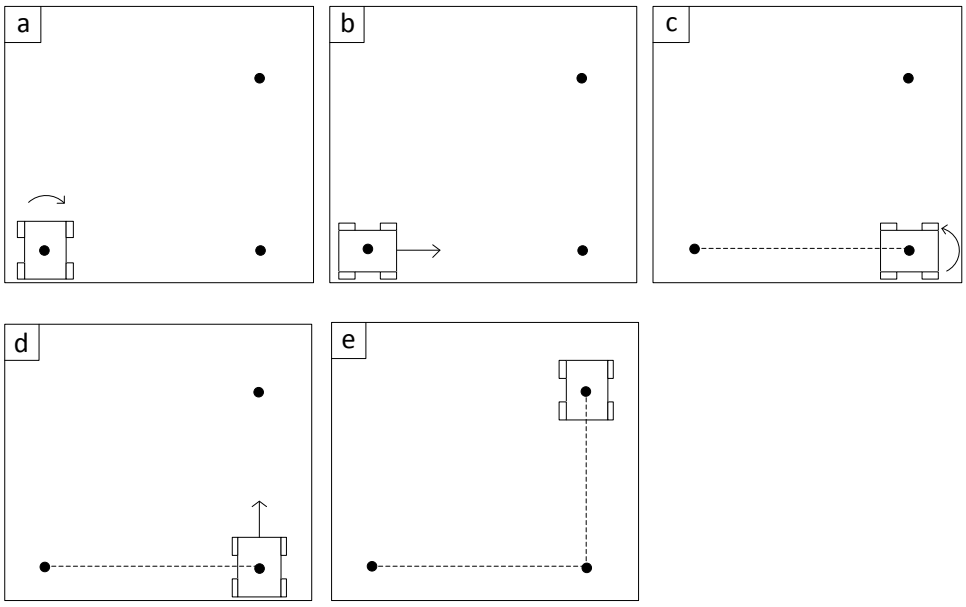


Figure B.6: The robot following a route by alternating between the orientation state (`a` and `c`) and the navigation state (`b` and `d`). At the final way point the robot reaches the stopping state (`e`). The dotted line indicates the driven path

Figure B.7 shows the state machine on which this route following routine is based.

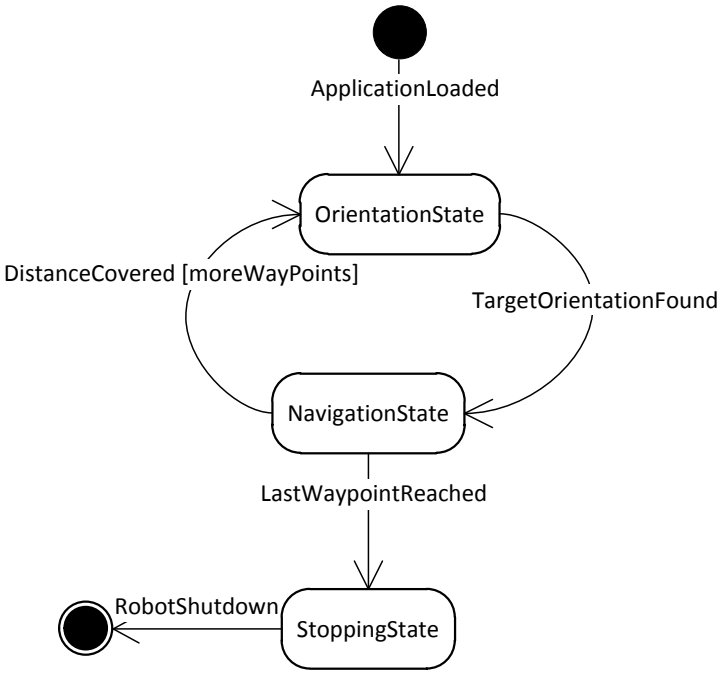


Figure B.7: The state machine of the robot

B.3.2 State machine design

To fully understand the decision controller, it is necessary to have insight into the communication among the different objects, making up the state machine. In the scenario shown in figure B.8, the robot is currently in the orientation state.

In the beginning of the scenario, a predefined period elapses, which causes the scheduler to invoke the step operation of the decision controller. The decision controller responds to this by invoking the step operation of the current controller state object, `orientState`. This causes the `orientState` object to read the current gyroscope value, and decide on an upcoming action.

For this particular scenario, the `orientState` object decides that the target orientation has been reached. This causes the `orientState` object to create an instance of the `NavigationState` class, and instruct the decision controller, to change into navigation state. The decision controller responds by calling the `leave` operation of the `orientState` object, which resets itself. Next, the robot controller invokes the `enter` operation of the `navState` object, which calculates the distance to be covered between the current way point and the target way point. Finally, the state change is performed, and after some time, return of control reaches the scheduler again.

Next time the predefined period elapses, the scheduler invokes the `step` operation of the decision controller. This now causes the decision controller to invoke the `step` operation of the `navState` object. This `step` operation will be invoked repeatedly by the scheduler, until the `navState` object decides that the distance between the current way point and the target way point has been covered. However, this part has been excluded from figure B.8.

When the distance between the current way point and target way point has been covered, the robot transitions into the orientation state, if there are more way points to be visited. Otherwise, it

transitions into the stopping state, and turns off the motors (the final way point has been reached). Although this is not shown in the diagram, the sequence of operation calls, is similar for every state transitioning.

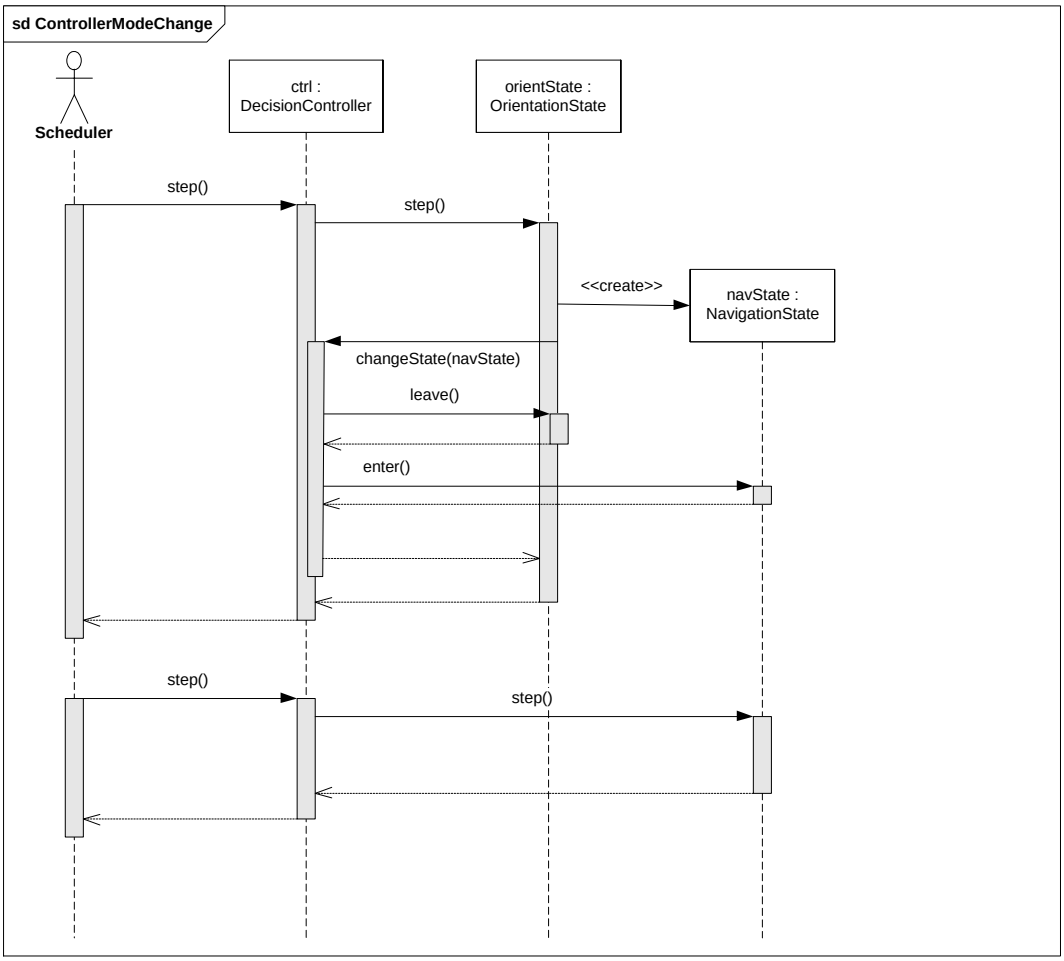


Figure B.8: The decision controller transitioning between states

B.4. System decomposition: Co-simulation contract

B.4.1 Other parameters

During the Autonomous Robot System case study, parameters not included in the co-simulation contract have been explicitly written down, for easy reference throughout development. These parameters are listed below, and serve as a check-list for parameter fine-tuning, to make sure that the important parameters, have been taken into account.

- Rotational inertia of the robot.
- Robot mass.

- Wheel radius.
- Center of rotation.
  - Wheel positions are calculated relative to this point (see section B.2.3.1).
- Position of wheels.
- Resistance to sideways movement.
- Resistance to forward movement.
- Scheduler period.
  - The period at which the main thread of the decision controller is invoked.

Other parameters not currently included in the model, but could be introduced in a later version, are listed below.

- Wheel inertia.
- Wheel mass.

B.5. Robot realization

B.5.1 Estimating speed

During the co-model calibration described in section 5.4.6, the resistance to forward movement and the resistance to sideways movement were adjusted, to make the co-model behave more realistic. In order to do this, the forward speed and the rotational speed of the system realization needed to be determined, which is covered in this sub-section. All times and distances have been measured using a stopwatch and a measuring tape, respectively.
The co-model calibration was made using two separate simulations. One, where the robot would drive in a straight line for five seconds, and another where the robot would spin left for five seconds. The latter simulation was only performed for the left direction, as the co-model assumes that the rotational speeds in both directions are the same. The target distance and orientation to be covered by the co-model during these five seconds simulation runs, were based on the rotational speed and the forward speed, calculated for the system realization.
In order to determine the forward speed of the system realization, it was instructed to drive in a straight line for ten seconds. This was repeated five times, and for each run, the distance covered was written down. The measurements for these five runs was then used for calculating the forward speed of the robot. The results concerning the forward speed of the system realization are shown in table B.1.
Similarly, the rotational speed of the system realization needed to be determined. In order to do this, the robot was instructed to spin around itself ten times, and the time it would take was measured. This test was repeated three times for each direction (left and right). From these measurements, the average time it would take to spin ten times in the left direction, was calculated. The same calculations were made with respect to the right direction. Based on the numbers of both directions, an average time it would take for a single spin was calculated. From this, the rotational speed was calculated. The results concerning the rotational speed of the system realization are shown in table B.2.

Table B.1: Measurements when determining the forward speed of the system realization

10 seconds forward run [cm]
180
180
181,5
180,5
181,2
Forward speed [m/s]
0,18064
Distance to be covered in 5 seconds [m]
0,9032

As it can be seen in table B.2, the system realization spins faster in the left direction. However, the rotational speed was calculated as an average of both directions, since the co-model assumes that the rotational speeds in both directions are equal. Naturally, adjusting the resistance to sideways movement like this, may impact the fidelity of the route completion time of the co-model, if it spins more left than right and vice versa, during the following of a route. This could be compensated for in the co-model, by regulating the resistance to sideways movement, depending on the spinning direction.

Table B.2: Measurements when determining the rotational speed of the system realization

10x spins in left direction [s]	**10x spins in right direction [s]**
66,7	75,1
67,1	76,1
66,8	77,1
Average of 10x spins in left direction [s]	**Average of 10x spins in right direction [s]**
66,86667	76,1
Rotational time for 1x spin [s]	**Rotational speed [rad/s]**
7,148333	0,878972
Angle to be covered in 5 seconds = 4,39486 rad	

B.5.2 Approximation of rotational inertia

The only parameter of the co-model, which was calculated, is the rotational inertia of the robot. As argued in section B.5.4, determining this value with a high fidelity is difficult. Therefore, the system realization was assumed to be a thin rectangular plate of height $h = 0.2\ [m]$, width $w = 0.174\ [m]$ and mass $m = 1.001\ [kg]$. Using the standard formula, the rotational inertia of the robot is given by:

$$J = \frac{m(h^2 + w^2)}{12} \approx 0.005862\ [kg \cdot m^2] \tag{B.21}$$

B.5.3 Technical details

This sub-section provides information about the different parts, which the system realization is composed of, and describes the physical connection among them. It does not concern the workings

of encoders or the gyroscope etc. These things have been covered in chapters 4 and 5.
The Autonomous Robot System realization, shown in figure B.9, is composed of the parts listed in table B.3. Note that the DC motors do not appear in this list, as they are part of the robot body frame. The diagram shown in figure B.10 provides an overview of how the different parts are connected. All the parts are physically attached to the robot body frame, which has been omitted from the figure B.10 (except for the DC motors). Neither is the camera used for tracking the robot during route following included in the diagram, as it was only being used for testing purposes (see chapter 5). This camera was mounted in the ceiling 2,46 meters above the ground, when capturing images (320 x 240 pixels) of the robot during route following.
The `Decision controller` is responsible for controlling the DC motors of the robot via the `Motor actuator` (H-bridge). Both the `Motor actuator` and `Decision controller` are being are supplied with a voltage from `Power supply 1`. In addition to this, the decision controller is also connected with the two encoders, from which it receives the generated pulses, caused by the wheels of the robot being turned. The `Secondary controller` is supplied with a voltage from `Power supply 2`. This controller is responsible for communicating with the gyroscope over a Inter-Integrated Circuit (I2C) bus, from which it reads the current robot orientation. Each of these readings must then be communicated to the `Decision controller` via serial communication (RS232), to support the driving decisions.

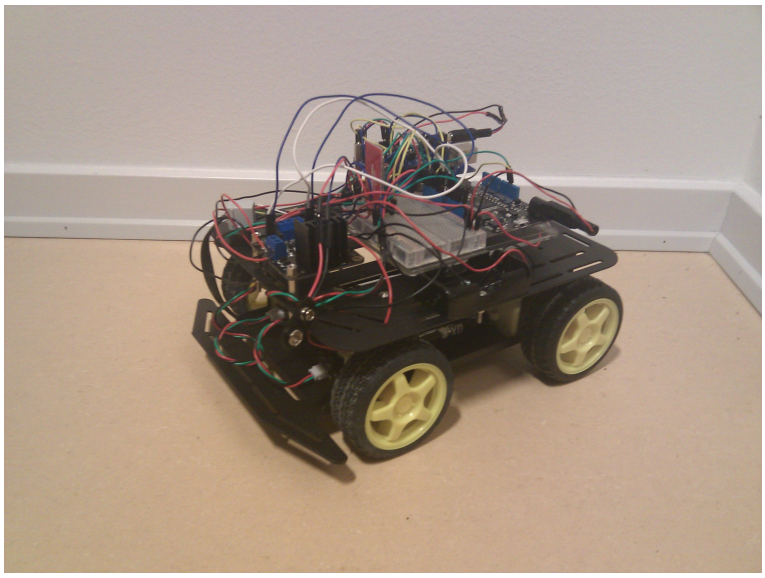


Figure B.9: An image of the Autonomous Robot System realization

B.5.4 Other things to be adjusted

Some parameters are more difficult to measure than others. For example, the mass of the robot was easily found using a weight. On the other hand, determining the rotational inertia of the robot with a high degree of fidelity is much more difficult, requiring several measurements and calculations. In such situations, it would be common practice to consider the robot as being composed of well-known objects such as cylinders and boxes, and use this assumption as a basis for determining the rotational inertia. If finding certain parameters are considered very difficult and costly, it may be appropriate to provide very rough estimates of the parameters and try to compensate for this by adjusting others. Rough approximations may even be sufficient, if the parameter is non-

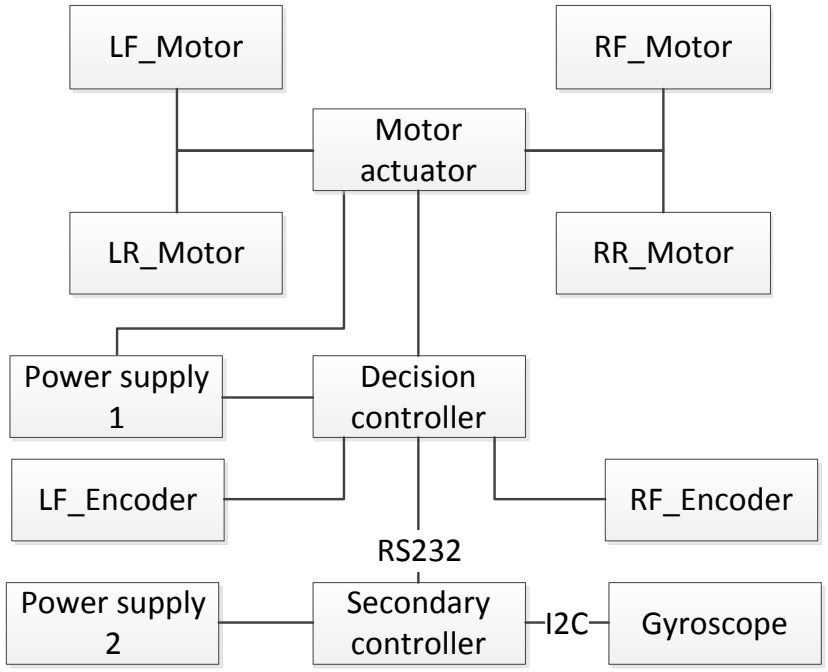


Figure B.10: Overview showing how the robot parts are connected

significant. For example, rough approximations with regards to the Autonomous Robot System, have been made for the rotational inertia and the center of rotation of the robot. In addition to this, it is assumed that motors share the exact same properties, i.e. they share the same performance characteristics. A description of how the forward speed, the rotational speed and the rotational inertia of the robot have been treated, can be found in sections B.5.1 and B.5.2.

B.5.5 Box route

The following of the "box" route for the calibrated co-model and system realization is illustrated in figure B.11. As would be expected from the results concerning the "square wave" route, the tendency of rotational overshoot shows (see sections 5.4.5 and 5.4.6). Furthermore, it can be confirmed from the data of table B.4 that the calibrated co-model completes the route 10,63% slower than the system realization with a deviation of 11,05% with respect to distance covered. Based on the numbers of the "square wave" route, a guess would not have fallen far from the data obtained using the "box" route.

B.5.6 Arrow route

The following of the "arrow" route for the calibrated co-model and the system realization is shown in figure B.12. The co-model deviates from the system realization by 17,08% and 15,92% (see table B.6) with respect to distance covered and route completion time. For the route completion time this is close to the result obtained using the "square wave" route of 14,60%, but still somewhat higher than that of the "box" route of 10,63%. However, the co-model deviating by 17,08% with respect to distance covered is quite high compared to the results obtained from the other

Table B.3: Autonomous Robot System realization part list

Part type	**Part description / Link**
Encoders	Wheel Encoders for DFRobot 3PA and 4WD Rovers
	`http://www.dfrobot.com/index.php?route=product/product&filter_name=encoder&product_id=98#.UBaDpbQtg0k`
Gyroscope	CruizCore XG1300L
	`http://www.cruizcore.com/e_p_robot05.html`
Decision controller	Netduino
	`http://netduino.com/netduino`
Secondary controller	Arduino Uno
	`http://arduino.cc/en/Main/ArduinoBoardUno`
Motor actuator (H-bridge)	2A Dual Motor Controller
	`http://www.dfrobot.com/index.php?route=product/product&product_id=66#.UBZ7wLQtg0k`
Camera	Kinect for Xbox 360
	`http://www.microsoftstore.com/store/msstore/pd/Kinect-for-Xbox-360-Refurbished/productID.226908400`
Power supply 1	2500 mAh NiMH Battery - AA (five pieces)
	`https://www.sparkfun.com/products/335`
Power supply 2	Duracell Rechargeable Standard 9v 170 mAh
	`http://www.amazon.co.uk/Duracell-Rechargeable-MN1604-170-Battery/dp/B0002FQXJK`
Robot body frame	The Pirate-4WD Mobile Platform
	`http://www.robotshop.com/dfrobot-4wd-arduino-mobile-platform-3.html`

Table B.4: Comparing the co-model to the system realization after calibration using the “box” route

“box” route						
Description	D_{sys} [cm]	Δd_{abs} [cm]	Δd_{rel}	t_{sys} [s]	Δt_{abs} [s]	Δt_{rel}
System realization	409,00	-	-	35,18	-	-
Calibrated co-model	363,81	45,19	11,05%	38,92	3,74	10,63%

routes. An explanation is found in the “arrow route” being the smallest route: As it can be seen in figure B.12, the overshoot when driving in a straight line causes a high relative deviation with respect to distance covered.

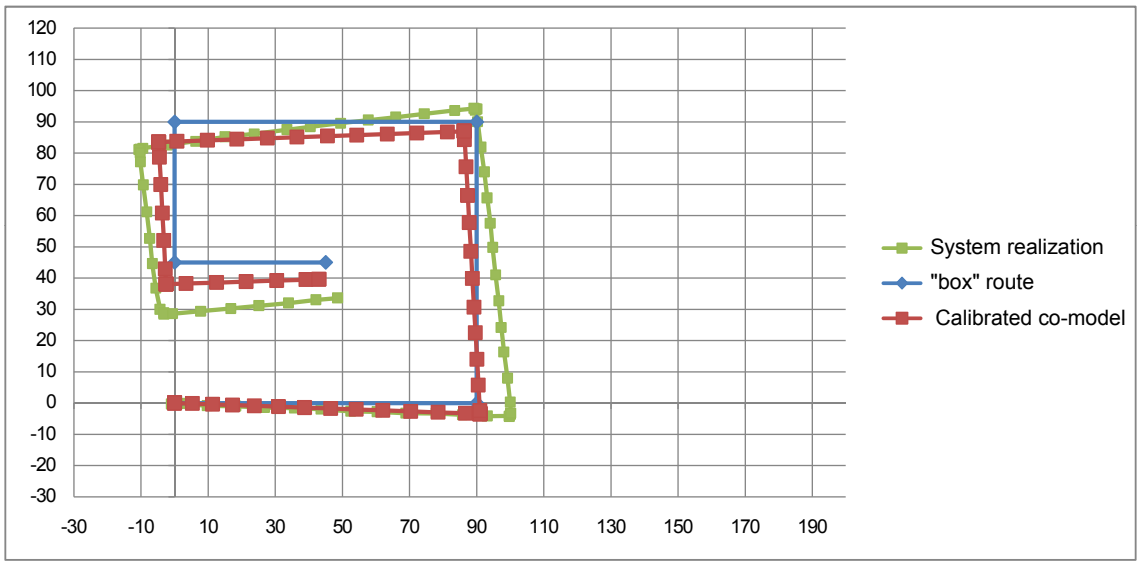


Figure B.11: Co-model and system realization “box” route following after calibration

Table B.5: Comparing the data results to the “box” route after calibration

“box” route, $D_{ideal} = 360$ cm			
Description	D_{sys} [cm]	ΔD_{abs} [cm]	ΔD_{rel}
System realization	409,00	49,00	13,61%
Calibrated co-model	363,81	3,81	1,06%

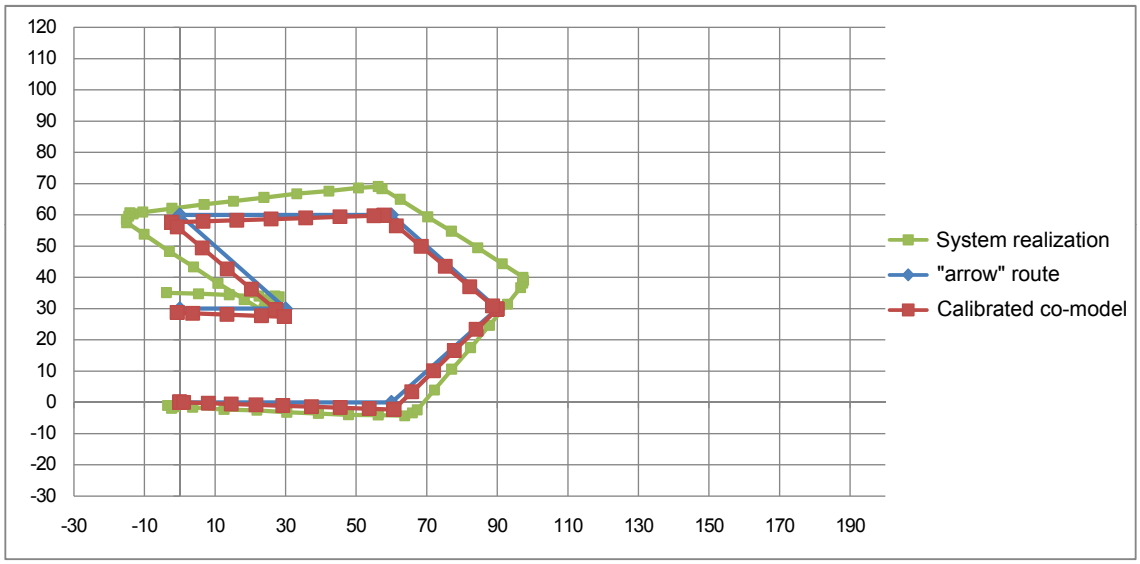


Figure B.12: Co-model and system realization “arrow” route following after calibration

Table B.6: Comparing the co-model to the system realization after calibration using the “arrow” route

“arrow” route						
Description	D_{sys} [cm]	Δd_{abs} [cm]	Δd_{rel}	t_{sys} [s]	Δt_{abs} [s]	Δt_{rel}
System realization	340,66	-	-	30,54	-	
Calibrated co-model	282,46	58,20	17,08%	35,40	4,86	15,92%

Table B.7: Comparing the data results to the “arrow” route after calibration

“arrow” route, $D_{ideal} = 277,28$ cm			
Description	D_{sys} [cm]	ΔD_{abs} [cm]	ΔD_{rel}
System realization	340,66	63,38	22,86%
Calibrated co-model	282,46	5,18	1,87%

Peter Würtz Vinther Jørgensen, Evaluation of Development Process and Methodology for Co-Models, 2013

Department of Engineering
Aarhus University
Edison, Finlandsgade 22
8200 Aarhus N
Denmark

Tel.: +45 4189 3000