ENHANCED MODELLING AND EFFICIENT REALISATION OF **CYBER-PHYSICAL SYSTEMS**

Electrical and Computer Engineering
Technical Report ECE-TR-21

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

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

```
public goForward: () ==> ()
goForward () ==
(
    motorsLeft.setValue(FullSpeed);
    motorsRight.setValue(FullSpeed);
);
```

AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

DATA SHEET

Title: Enhanced Modelling and Efficient Realisation of Cyber-Physical Systems
Subtitle: Electrical and Computer Engineering
Series title and no.: Technical report ECE-TR-21

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: 2014 Pages: 28
Editing completed: October 2014

Abstract: This report describes the work carried out for the first half of the PhD project titled Enhanced Modelling and Efficient Realisation of Cyber-Physical Systems (CPSs). The work contributions cover methods and tools to address the challenges of CPS development using a model-based approach. First the report introduces a technique to assist stakeholders in organising design information produced during modelling. Then a modelling language extension that supports reasoning about the energy consumption of the CPUs of a CPS is presented. Afterwards a technique that enables one to include real system components into the system simulation is described and finally the report presents a technology that enables construction of code generators for multiple target languages. In addition to covering the work contributions, this report also describes future work plans that will lead to the completion of the PhD project by April 2016.

Keywords: cyber-physical systems, code generation, hardware-in-the-loop simulation, co-simulation, co-model, energy-aware design

Supervisor: Peter Gorm Larsen

Please cite as: Peter W. V. Jørgensen, 2014. Enhanced Modelling and Efficient Realisation of Cyber-Physical Systems. Department of Engineering, Aarhus University. Denmark. 28 pp. - Technical report ECE-TR-21

ISSN: 2245-2087

ENHANCED MODELLING AND EFFICIENT REALISATION OF CYBER-PHYSICAL SYSTEMS

Peter Würtz Vinther Jørgensen

Aarhus University, Department of Engineering

Abstract

This report describes the work carried out for the first half of the PhD project titled Enhanced Modelling and Efficient Realisation of Cyber-Physical Systems (CPSs). The work contributions cover methods and tools to address the challenges of CPS development using a model-based approach. First the report introduces a technique to assist stakeholders in organising design information produced during modelling. Then a modelling language extension that supports reasoning about the energy consumption of the CPUs of a CPS is presented. Afterwards a technique that enables one to include real system components into the system simulation is described and finally the report presents a technology that enables construction of code generators for multiple target languages. In addition to covering the work contributions, this report also describes future work plans that will lead to the completion of the PhD project by April 2016

1

Introduction

This chapter introduces the reader to the field of research, describes the challenges that will be addressed in this PhD project, and finally provides an overview of the different chapters.

1.1 Introduction to the field of research

In a Cyber-Physical System (CPS) computational units (CPUs) are collaborating on controlling physical entities such as electrical and mechanical devices in order to achieve a common goal. A well-known example of a CPS is a modern car where steering and braking is orchestrated by several computational units that communicate across a network in order to ensure that the car can be manoeuvred safely and efficiently under different driving conditions.

Many modern cars are equipped with a cruise controller that maintains a steady speed of the car by regulating the throttle actuator, based on inputs from sensors that monitor the velocity and acceleration of the car. For the cruise control to provide a pleasant ride, the control decisions need to take the relationship between physical quantities such as mass, velocity and acceleration into account. Therefore, the technical knowledge required to successfully develop such a system, spans multiple engineering disciplines, which causes a high degree of design complexity, and requires the collaboration among various types of stakeholders with different skills.

A common way to deal with the design complexity of a CPS is to use multidisciplinary models to reason about the desired properties of the system. A model is an abstract representation of the system that encapsulates its key characteristics in order to enable the development team to focus on the most important aspects of the system. Simulation of the model is one way to assist the development team in understanding the impact of design decision in the early development phases in order to determine if the design meets the system requirements.

If the system model describes the system realisation with a sufficiently high degree of fidelity then the outcome of the simulation can be used to predict the properties for the final version of the system. Simulation of the system model also enables analysing properties that would otherwise be too expensive or difficult to do using a system realisation. For example, using traditional approaches to CPS development, such as physical prototyping, determining the most energy efficient hardware architecture for a CPS, results in several different system realisations each with a different hardware architecture. However, the nature of a model allows such analysis to be carried out virtually, and therefore the impact of changing the physical parameters of a system can be automated and analysed more thoroughly. Other examples of such analyses include investigating

how changing things such as the axle width or wheel size of a car effects its manoeuvrability, which would require more cars to be build using physical prototyping.

1.2 Purpose of the PhD project

The purpose of this PhD project is twofold: First, it aims to enhance existing modelling technologies and develop new ones to accommodate the challenges of CPS development. Secondly, it seeks to propose ways to efficiently transfer knowledge obtained during model analysis to the subsequent phases of the development process. The contributions of this PhD will therefore target different phases of the development cycle – from the early stages of CPS development where the system is defined – to the realisation phase where it is constructed.

This leads to the hypothesis of this PhD project, which assumes *it is possible to develop new methods and tools to address the challenges of CPS development that enable modelling to be enhanced and realisation to be carried out more efficiently in model-based CPS development.*

Use of models is one way to reduce the resources needed in order to develop a CPS. A model enables exploration of a large design space and testing of a system to be done virtually in order to reduce the number of physical prototypes that need to be constructed. This is desirable for systems such as cars and aircrafts where construction of physical prototypes is expensive. However, modelling activities produce additional design artefacts, and therefore this PhD explores how these artefacts can be leveraged in an efficient way.

To demonstrate the hypothesis the PhD project will:

- Explore existing tools and methods in the context of CPS development,
- Identify limitations of existing tools and methods,
- Develop tools and methods to address the identified limitations,
- Apply the tools and methods to different CPS case studies.

1.3 Document structure

The first part of this document provides the background information needed to understand the work for the first half of this PhD project (Chapters 1 and 2). Chapters 3 through 6 continue by describing different work contributions that together support modelling and realisation of a CPS. Finally, the last part of the document summarises the work that has been carried out so far and discusses ideas for future plans that will contribute to achieving the goals of this PhD project (Chapter 7). The description below briefly summarises the content for each chapter and the appendices and briefly mentions the publications that the different chapters are based on.

Chapter 1 – Introduction: Presents the field of research and the goals to be achieved for this PhD project.

Chapter 2 – Background: Provides the background information that is needed in order to understand the current as well as the planned work contributions for this PhD project. This chapter contains extracts from two papers that document the modelling platform, Overture, used in all the work contributions in the first half of this PhD. The paper *An Architectural Evolution of the Overture Tool* [1], published and presented at the Overture workshop, 2013, describes an architectural change made to Overture in order to enable this platform to be extended more easily in other research projects. The paper *Migrating from a Performant to an Extensible Architecture in the Overture Tool* [2], submitted to the ACM SIGAPP Symposium on Applied Computing (SAC), 2015, documents the experiences of migrating to the new (extensible) version of Overture and includes metrics to document the change in both performance and extensibility. Finally, to provide the background information needed to understand Chapters 3 through 6, extracts from the publications that these chapters are based on are also included in Chapter 2.

Chapter 3 – Organising Design Information: Presents a way to organise design related information resulting from the early modelling activities. This chapter contains extracts from the paper *Five Views of a Collaborative Model* which was accepted as a short paper at the Simultech conference, 2014. This paper was withdrawn and will be revised and submitted to a different publication venue.

Chapter 4 – Analysing Energy Usage using Modelling: Presents a language extension that enables modelling of different power states for computational units in a CPS in a modelling language called VDM. Using this approach it is possible to use model simulation to investigate different strategies to managing a fixed energy budget. This work contribution is based on the paper *Modelling Energy Consumption in Embedded Systems with VDM-RT* [3], published and presented at the ABZ conference, 2014. This language extension also relates to the work carried out at the Overture Language Board (LB), where maintenance of VDM is carried out. The work carried at the Overture LB is described in the paper *The Overture Approach to VDM Language Evolution*, published and presented at the Overture workshop, 2013.

Chapter 5 – Hardware-In-The-Loop Simulation: Proposes a methodology to gradually realise a CPS by including hardware in the system simulation. This chapter contains extracts from the paper *Hardware In the Loop for VDM-Real Time Modeling of Embedded Systems* paper [4], published and presented at the Modelsward conference, 2014.

Chapter 6 – Code Generation: Presents a code generation platform that enables the software implementation to be generated automatically from a system model. The work presented in this chapter is based two papers: *Towards an Overture Code Generator* [5] describes the early work on a VDM to Java code generator, and the paper *A Code Generation Platform for VDM* [6] describes a code generation platform for VDM, that enables construction of code generators for different target languages.

Chapter 7 – Current Status and Future Plans: Summarises and discusses the results achieved so far in this PhD project. Ideas for future plans are presented and concluding remarks are given.

Appendix A: Lists the publications for this PhD along with other papers that are currently in review or preparation.

Appendix B: Provides an overview of the courses completed in the first half of the PhD as well as courses planned for remaining part.

2

Background

This chapter presents the background information needed in order to understand the work contributions for the first half of this PhD project. More specifically, this chapter describes the modelling technologies and languages that are used in the different contributions.

2.1 The Overture tool

The Overture project [7] supports the Vienna Development Method (VDM) [8], which is a set of modelling techniques with a long history of developing computer-based systems in both research and industrial projects. Since 2003 the Overture community has developed and maintained the Eclipse/Java-based, open-source Overture tool for analysing models written in VDM. The mission of the Overture project is to provide industrial strength tools for formal modelling and to foster an environment that allows researchers and interested parties to experiment with the tool and the language. Today Overture has evolved into a platform that is utilised in both the teaching of and research into formal methods.

The basic architecture of the Overture tool is similar to that of any tool whose primary purpose relates to modelling languages: a parser creates and instantiates an Abstract Syntax Tree (AST) of a model, and every other component interacts with the AST in some way.

2.2 The Vienna Development Method

The Vienna Development Method (VDM) is one of the most well-established formal methods for describing computer-based systems [9], and it has evolved into the three dialects: VDM-SL, VDM++ and VDM-RT. VDM-SL is the flat specification language used for modelling of sequential systems, VDM++ introduces concurrency and object-orientation and VDM-RT further extends VDM++ with support for modelling of real-time systems on distributed hardware architectures.

VDM-RT models the system architecture in a special **`system`** class using language constructs for CPUs and buses. CPUs are characterized by speed and scheduling policy and allocated by objects of active classes. An object is deployed on a CPU by passing it to the `deploy` operation of the CPU instance. Buses connect CPUs and enable communication at user-specified bandwidths using predefined communication protocols. Objects can invoke operations of other objects deployed on different CPUs, which causes data to be transmitted on the connecting bus.

The VDM-RT interpreter maintains a global notion of time that can be referred to using the **time** keyword. The global time progresses by a default number of nanoseconds as functions and operations are invoked. This default increase in time can, however, be overruled using the **cycles** and **duration** statements, which enable specification of execution delays relative to processor speed or as absolute time measures, respectively.

Multiple threads can exist on a CPU, but only one thread can execute at a given point in time. Threads are periodic and can be declared in the **thread** section of a class using the **periodic** keyword. Periodic threads are specified as a four-tuple of the form (p, j, d, o) where p denotes the period, j is the jitter, d is smallest time to elapse between consecutive executions of a periodic operation and o is the offset.

VDM-RT supports modelling of synchronisation 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 use 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 [10].

2.3 The System Modelling Language

Multi-disciplinary models demand for a modelling language accommodating all of the involved domains. Such a language is found in SysML [11], which is an extension to the Unified Modelling Language (UML). Using SysML, it is possible to model constructs of different domains in the same system description. In CPS development, this is used to describe the interaction among software and physical system components. The following description provides an overview of the most important SysML concepts.

Requirements: A *use case diagram* supports identification of the important *actors* and major functions of the system. Modelling 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 and consistency across the different SysML diagrams. Similarly, it is possible to indicate that a system component satisfies a requirement using the *satisfy* relation.

Structure: SysML introduces the concept of a *block*, which constitutes the basic unit of structure such as a software class or a hardware component. Blocks can be arranged in a *Block Definition Diagram* (BDD) in order to describe the hierarchical system structure. The internal structure of a block is specified using the *Internal Block Diagram* (IBD), while interaction among the internal parts is described using *ports*.

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

Constraints: *Constraint blocks* are used to capture equations such as physical laws. In the *parametric diagram* these blocks can be used to describe the constraints on system properties (performance, physical properties etc.).

2.4 Co-simulation and supporting technologies

Engineers who use models as a basis for developing CPSs normally make use of tools and formalisms optimised for a particular domain, in order to model the individual system parts. Often the mechanical and electrical parts of a system are based on *Continuous Time* (CT) models of physical phenomena expressed using differential equations, while the computer-based system parts are based on *Discrete Event* (DE) models of the system's control logic, described using discrete mathematics.

A co-modelling approach combines the DE and CT models to bridge the gap between the involved engineering disciplines. By creating and simulating high-level models of the entire CPS, systems engineers are able to reason about system-level properties at design time. This ensures early feedback on system properties and eases communication across the involved engineering disciplines, as the impact of design decisions are made visible to the entire team.

Several technologies for co-modelling and co-simulation are available. The subsections below will briefly cover some of these in relation to the work described in Chapter 3, focusing mostly on the Crescendo toolchain, which is used in this PhD project.

2.4.1 The Crescendo toolchain

The Crescendo toolchain expresses CT models as differential equations using bond graphs [12], and DE models in VDM-RT. Bond graphs is a domain-independent non-causal way to model ideal transfer of energy in a physical system. Energy exchange is characterised by flow and effort variables. The meaning of these variables depends on the physical 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 simulation engines supporting bond graphs and VDM-RT are 20-sim [13] and Overture, respectively. A co-simulation engine is responsible for managing the coordination between the DE and CT simulators during a co-simulation. This allows exchange of data and time synchronization between the two simulators as the DE and CT models are executed.

2.4.2 Other co-simulation technologies

The MADES Co-simulation Approach [14] allows designers to combine logic formula, describing the controller, with non-casual CT models created in Modelica [15]. Stateflow extends the MATLAB/Simulink [16] toolbox with control logic for modelling of reactive systems using state charts and flow diagrams. Advanced Modelling Environment for performing Simulations of engineering systems [17] extends the MathWorks toolbox with a non-causal modelling approach. Finally, Ptolemy II [18] is a modelling and simulation framework for multi-disciplinary systems that uses actor-oriented design principles for describing CPSs.

3

Organising Design Information

Co-modelling supports reasoning about system-level properties through simulation of the co-model, for example using one of the technologies described in Section 2.4. Stakeholders using co-simulation technologies do however lack a systematic approach to present and organise design artefacts to different stakeholders. To address this deficiency, this chapter presents a view model serving as a common viewpoint framework for these technologies to assists stakeholders in comprehending the challenges of CPS development.

3.1 Introduction

Modelling helps focusing on the right questions early in the development cycle and thus reduce the number of design artefacts that only add to the complexity of the system documentation. However, co-models are themselves complex and give rise to production of additional design artefacts such as system descriptions and documentation of new insights obtained. To make information easier accessible to stakeholders a view model for co-models organises design artefacts into a set of predefined *views*.

A view model defines its views based on the characteristics of the system under consideration. For example, for a system where dependability is of paramount importance, a view can be used to address this aspect of the system. Use of views for defining the important system aspects (hardware architecture, software design etc.) enforces structure to the development work by assisting stakeholders in addressing the critical development tasks already at the co-modelling level. Based on the technical challenges of CPS development this chapter proposes such a view model that supports CPS development using any co-simulation technology. The view model will be demonstrated using a robot case study. However, due to page limitations only two of the five views will be exemplified.

3.2 Five views of a co-model

View models have been widely used in, but are not limited to, the field of software engineering, where they are used to manage complex software projects and develop software implementations. During the early phases of co-modelling the focus is on understanding the system under consideration at system-level. This should reflect how design artefacts produced during co-modelling

are organised. The five view structure below arranges the design artefacts according to the main technical challenges of CPS development.

Scenarios view: This view captures the co-model purpose, use cases, requirements, assumptions (e.g. constant temperature) and definitions (e.g. units of measure) of the system. This view drives the identification of elements in the remaining views, and the use cases support system validation.

Subsystems and Partitioning view: This view describes relevant subsystems and the associations among them. This view also covers the partitioning of the identified subsystem constructs, i.e. deciding whether they belong to the DE or CT domain. When a subsystem construct has been subject to partitioning, it is referred to as a *domain construct*. Dividing the system into subsystems is a useful way to deal with complexity, delegate work to development teams, but it also serves as input for the partitioning activity.

Domain Constructs and Interaction view: This view covers the descriptions of the DE and CT constructs and their interaction. Interaction can be both internal and external, i.e. within a domain construct and across domains, respectively. When developing a CPS with tightly coupled DE and CT constructs it is necessary to understand the links between computer-based control and the physical quantities (acceleration, force etc.) in order to meet the system objectives.

Distribution view: This view describes the system architecture of the controller, and the mapping of DE constructs onto processing elements. This view also covers distributed communication, e.g. modelling of protocols.

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

For documenting the five views we encourage the use of SysML, which provides support for multi-domain modelling and consistency checks across views.

3.3 The view model

The Autonomous Robot System has been developed using the co-modelling based development process and methodology suggested by Wolff in [19]. This methodology is captured in the form of step-wise *methodological guidelines* and can be used with any of the co-simulation technologies described in Section 2.4. The methodological guidelines support developers in using best co-modelling practices, but without being explicit about what system aspects to consider. This is the responsibility of the view model, which is an extension to the work of Wolff.

3.3.1 Case study

To demonstrate the use of the view model we apply it to the Autonomous Robot System case study, where a co-model was developed using the Crescendo tool by following the methodological guidelines proposed by Wolff. In the Autonomous Robot System case study a four wheeled robot follows a known route of two-dimensional waypoints 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 waypoints in the correct order. Route following is

performed in known surroundings by having a controller running software responsible for making the driving decisions. This controller will subsequently be referred to as the decision controller. Using collaborative activities the Autonomous Robot System was developed by a software engineer and a robotics engineer subsequently referred to as the DE and the CT (domain) experts, respectively [20]. The DE model describes the software executing the driving decisions, while the CT model expresses the physics of the robot (motor forces, wheel forces, robot orientation etc.).

3.3.2 Subsystems and Partitioning view

In Figure 3.1 the top-level block, the specification of the Autonomous Robot System, has been decomposed into two subsystems. The first subsystem represents the robot and the second represents the surroundings being formed by the surface the robot is driving on. Besides the body frame, the robot is composed of wheels, motors and sensors. The motors are prefixed by their attachment position, e.g. RF (right front), in order to distinguish between them. The same naming convention is used for wheels and encoders — for reading convenience only a single motor/wheel/encoder configuration is shown in Figure 3.1.

The decision controller executes the control logic that makes the driving decisions. Similarly, the secondary controller will execute the device drivers sampling the encoders and the gyroscope. Therefore, the two controller blocks in Figure 3.1 are put in 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 to load 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.

Documenting the subsystems and performing the partitioning as a collaborative activity helped the domain experts in exchanging the knowledge necessary to establish interaction across the domains that would lead to the desired system behaviour. For example, the CT expert would explain to the DE expert the workings of the encoders and the gyroscope and how the output of these sensors could be used for determining the distance travelled and the orientation of the robot.

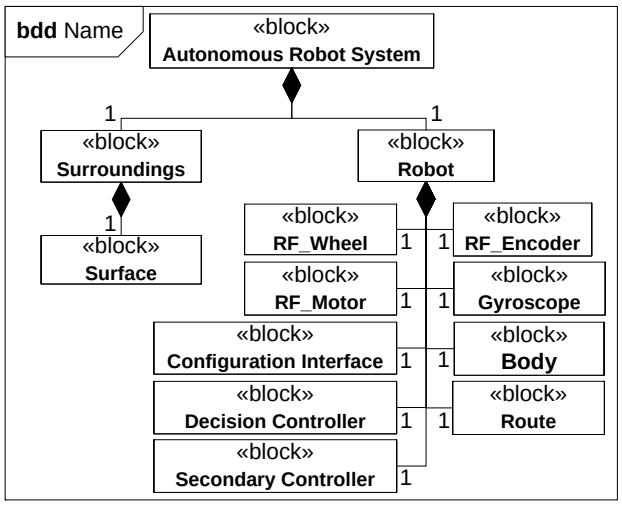

Figure 3.1: Subsystems and Partitioning view of the Autonomous Robot System illustrated using a SysML BDD

3.3.3 Domain Constructs and Interaction view

The DE constructs identified during the partitioning activity can be described using UML diagrams. For example, the class diagram is used to describe the structure of the software making the control decisions and the IBD is used to document the internal composition of domain constructs.

Figure 3.2 highlights a small part of the complete internal robot structure to demonstrate how IBDs help obtaining insight into domain constructs. The motor-wheel-encoder configuration is completely symmetric for every wheel. Therefore Figure 3.2 only shows the right front wheel (along with the motor and the encoder).

The SysML descriptions alleviated moving to domain models written in VDM-RT and bond graphs. Translating the DE descriptions (class diagrams, sequence diagrams etc.) to VDM-RT was easily done since the two modelling languages share many equivalent concepts. Alternatively, a code generator could have been used (Chapter 6). Furthermore, the CT descriptions (IBDs, parametric diagrams, equations etc.) were used to describe the structure and the physical laws of the system and served as a starting point to express the CT model using bond graphs in 20-sim. The SysML descriptions further helped defining the cross-disciplinary interaction. For the Autonomous Robot System the decision controller (a DE construct) is responsible for changing the variables representing the speed and direction of a motor (a CT construct), i.e. `direction` and `pwm` in Figure 3.2. The Crescendo technology terms these as *shared variables*, which together define how the DE and CT model interact.

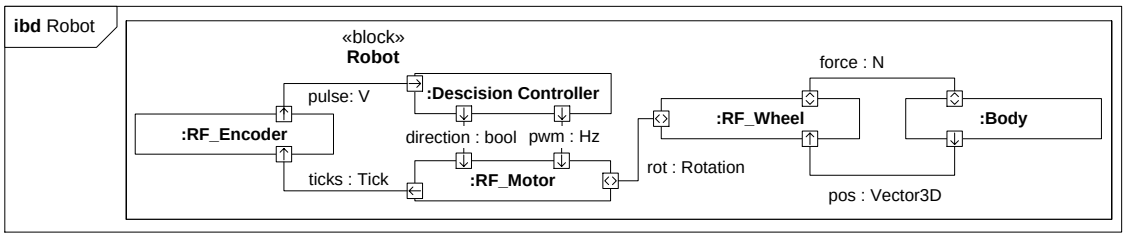

Figure 3.2: Domain Constructs and Interaction view of the Autonomous Robot System illustrated using a SysML IBD

3.4 Final remarks

Collaborative modelling is used to address some of the key challenges of CPS development, but reasoning about complex systems using a co-model is in itself a difficult task. The view model will assist stakeholders in comprehending this complexity by enforcing structure to the development work, so important issues can be identified and addressed early in the development cycle, where corrections are cheaper to make. A view model provides a tool-independent way to reason about a CPS by establishing a common vocabulary so stakeholders with different backgrounds can understand each other. Therefore, the five views of a co-model proposed in this chapter will serve as a useful tool-independent viewpoint framework for addressing the technical and non-technical challenges of CPS development.

4

Analysing Energy Usage using Modelling

A CPS is often a battery powered device with a limited energy budget, which makes optimal usage of energy a critical task faced by CPS developers. One approach to making a CPS energy-efficient is by putting its CPUs into less energy consuming modes when their computational resources are not needed. In order make VDM-RT a better modelling language for analysing such non-functional properties of a CPS, this chapter presents a language extension to VDM-RT that enables reasoning about a system's energy consumption before the system is realised.

4.1 Modelling CPU power states

This work presents a formal modelling approach for analysing and evaluating the energy consumption of a CPS. The approach is based on the VDM-RT dialect and specifically focuses on including CPU sleep mode in the simulation. This mode is an operational state that most of the modern CPUs used in today's microcontrollers implement. In this mode the system enters a low power consumption mode where a CPU is deactivated until its computational resources are needed again.

Initially, we proposed the representation of sleeping states in VDM-RT by applying a design pattern structure at the modelling level [21]. This approach, while effective, introduced an additional level of complexity at the modelling stage. The extensions made to VDM-RT, presented in this chapter, instead incorporates the notion of a sleeping state into the CPU class. The preliminary application of this modelling approach to a simple case study has shown that it is possible to produce estimates of approximately 95% accuracy without having to create a physical prototype for each solution under consideration.

4.1.1 Language modifications

Extending the VDM-RT CPU class with a `sleep` operation makes it possible to represent embedded software that sleeps the CPU responsible for executing the application logic. This is shown in Listing 2. In this example the `ApplicationLogic` executes in `APP_TIME` time units and then the CPU (microcontroller unit, `mcu` in Listing 2) is put to sleep. Once woken up, execution resumes by invoking the `PostWakeUpLogic` operation.

```
duration (APP_TIME) ApplicationLogic();
System`mcu.sleep(); -- Blocks until activated externally
duration (POST_WAKE_UP) PostWakeUpLogic();
```

Listing 2: High-level representation of the embedded software putting the CPU in a sleeping state.

The operation `active` can be invoked by the parts of the model that represent internal wake-up sources such as a sleep timer or external sources such as interrupts (see Listing 2). This allows the sleeping CPU to resume execution (see Listing 3).

```
System`mcu.active();
```

Listing 3: Static call waking up (activating) the `mcu`.

4.1.2 Modelling different wake-up policies

The language extensions presented above, and more specifically the way the `active` operation is used, can model different wake-up policies. In order to represent an external interrupt based wake-up policy, a model needs to incorporate additional logic to represent external events that can be fed to the system. These events will trigger the wake-up logic at a certain time.

A periodic thread deployed on a non-sleeping CPU can be used to represent a hardware block that waits for the external interrupt. Once the event has been recognized by the model representing the system, it can invoke the `active` operation and resume application processing.

The second wake-up policy is based on sleep timer expiration. From the modelling perspective it can be treated in a manner analogous to the external event modelling approach presented above. A periodic thread runs the sleep timer logic that periodically checks if the `overflowOn` time units have elapsed.

If this condition is satisfied the CPU is activated and resumes execution of the application logic. It is the responsibility of the application logic to set up the sleep timer again before going to sleep the next time.

4.2 A modelling structure for modelling the wake-up policies

A generic modelling structure that can be used to model the wake-up policies described above is shown in the class diagram in Figure 4.1. This modelling structure uses two VDM-RT CPUs that are connected through a VDM-RT bus (`esBUS`):

WakeUpHWSource: represents a hardware block that features the necessary components to wake up the CPU from a sleeping state.

MCU: represents a microcontroller unit containing the CPU that will be operating in active and sleep mode depending on the control logic under study.

These CPUs run the following model components:

WakeUpSource: defines a generic template that must be realised by the components that wake up the CPU. It specifies the use of the operation `mcu.active()`.

SleepTimer: is a concrete realisation of the `WakeUpSource` that models a wake up configuration based on a resettable timer.

WakeUpInterrupt: is a concrete realisation of the `WakeUpSource` that models a wake up configuration based on an interrupt generated on the occurrence of an external event. This is modelled using the approach proposed above.

Application Logic: models the application logic running on the main CPU. This application is able to sleep the CPU through the invocation of the `mcu.sleep()` operation. This logic will timestamp the transitions between the active and sleep states and log the timestamps to a file.

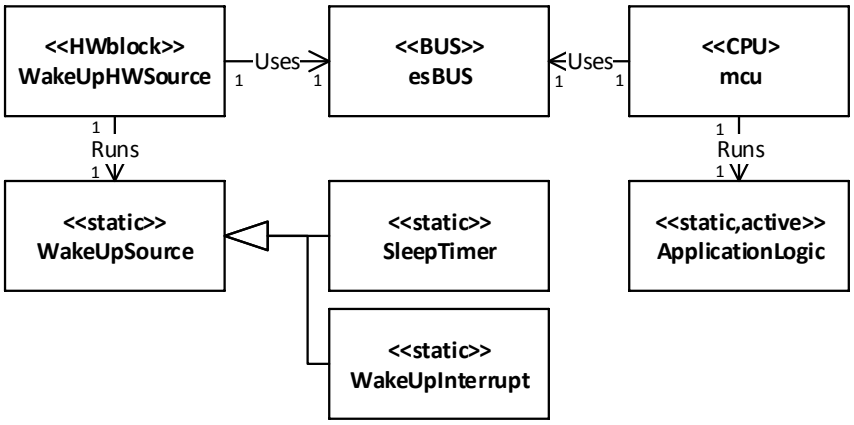

Figure 4.1: Generic structure for models using the sleep extension.

The Log produced by the application logic can be analysed once the simulation has completed in order to study both power and energy consumption. Based on this log, and taking into consideration the manufacturers specification for the CPU under study, one can plot the power consumption of the CPU over time. In order to determine the energy consumption this curve should be integrated. This analysis is considerably easier than a study based on prototypes and measurements and provides a high level of accuracy (≈95%). Details for a case study using this modelling structure can be found in [22].

4.3 Final remarks

Exploring different strategies for efficient energy usage in CPS development by controlling the power modes of CPUs has most traditionally been done through the construction of different physical prototypes. In this chapter we have enabled design space exploration of energy usage in a CPS using the VDM-RT notation. This enables design decisions to be made based on accurate energy consumption predictions without having to realise multiple system architectures. We have implemented language extensions for VDM-RT in the Overture tool, which enable controlling power modes of system CPUs at the modelling level. Combining our language extensions with existing support for describing hardware architectures enables reasoning about energy usage in a CPS using VDM-RT.

5

Hardware-In-The-Loop Simulation

Hardware-In-The-Loop (HIL) simulation is a technique for incorporating hardware in the simulation of the system model. The purpose of this approach is two-fold: First it increases the fidelity of the simulation outcome, and secondly it provides a way to gradually realise a CPS. To exploit the advantages of HIL simulation, this chapter presents a method for introducing hardware into the simulation of a VDM-RT model in order to enable a combined execution (co-execution) of model and hardware/software elements.

5.1 HIL setup design

The main objective of the HIL technology is to allow a stepwise transformation of models of functionality into components that implement that functionality. In order to benefit from the models one can combine the components implemented with the models of those that still have not been implemented. This is the key idea behind the HIL system presented in this chapter, which has not been applied previously for VDM-RT. Figure 5.1 illustrates this concept with an example based on the design of a CPS that enters three stages: `Acquire Data`, `Process` and `Provide Output`. The design of such a system starts with the modelling of the complete system functionality. This is illustrated in the upper part of Figure 5.1 (Full Modelling). The figure shows that all the components are modelled (represented on the VDM side) and no components have been implemented on the target (the hardware (HW) side). After the system has been modelled the component providing the `Process` functionality is implemented. It is the intention to substitute the modelled `Process` component with its corresponding implementation and still use the rest of the model for system simulation. This approach is shown in the lower part of Figure 5.1. At this point it is possible to start the system simulation on the VDM-RT side in the `Acquire Data` stage and then the HIL system will hand-over the execution of the `Process` stage to the implementation deployed on the target. Once `Process` has completed, execution will return from the target CPS to the VDM-RT model and continue with the `Provide Output` stage.
By applying the approach proposed in this chapter the design engineer is able to benefit from: the expressiveness of VDM-RT to represent real-time constraints and system properties, the simulation capabilities of Overture, and the insight gained through physical prototyping using a combined-modelling prototyping approach.

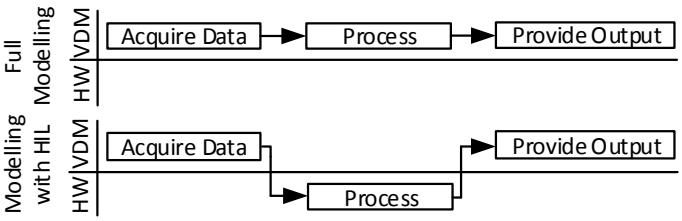

Figure 5.1: Device modelling (upper part of the figure) and combined device modelling with HIL (lower part of the figure).

5.1.1 Prototype hardware

The Device Under Test (DUT) selected for this work is a System on Chip (SoC). This kind of platform allows the combined deployment of hardware and software components in the same piece of silicon. The DUT allows testing of the HIL concept for VDM-RT in a more complete manner, taking into account both hardware and software components. These components can act together or in isolation while running on the same piece of hardware.

In order to monitor the output produced by the DUT on its digital buses we have used a logic analyser (Saleae Logic 8). This device samples a number of logical lines over a period of time. Any changes on the lines during this period of time will be logged. The logic analyser selected for this work is connected to a PC and it is possible to control it from custom software, created using a software development kit provided by the manufacturer[1].

5.1.2 Software and modeling components

The workstation software and modelling components composing the HIL setup are shown in the class diagram in Figure 5.2. Each of the components are treated individually in the description below.

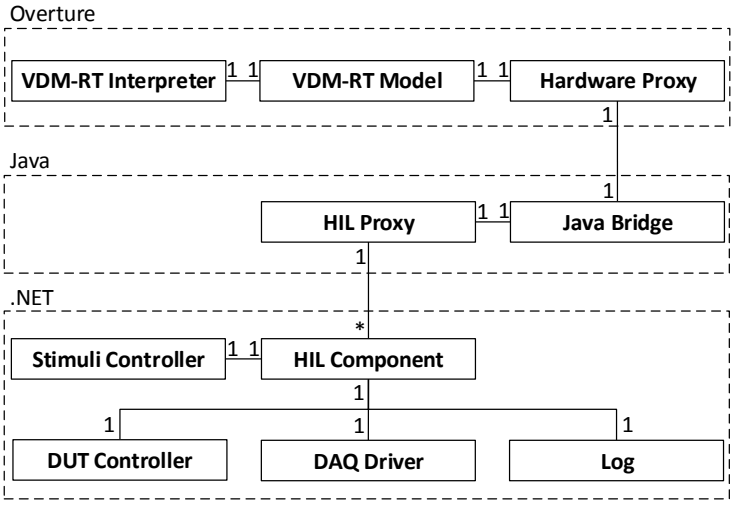

Figure 5.2: The software and modelling components of the HIL setup shown in a class diagram.

[1] Additional information on Saleae Logic 8 is available at: `http://www.saleae.com/logic/`

VDM-RT Interpreter: executes the `VDM-RT Model` representing the embedded software under consideration.

Hardware Proxy: specifies the interface of the hardware being controlled from the VDM-RT execution. All invocations are initiated in VDM-RT and communicated through the `Java Bridge` before they reach the physical hardware.

Java Bridge: enables delegation of VDM-RT functionality execution to Java. In this way one can invoke a VDM-RT operation or function and have the body specified and executed as a Java method. The `Java Bridge` does automatic conversion and transferring of input and output values between Java and VDM-RT.

HIL Proxy: is a thin software component that accepts inputs from the `Java Bridge` inside Overture and relays the invocations originating from the VDM-RT execution to a `HIL Component` responsible for communicating with the hardware used in the HIL setup. The `HIL Proxy` is interoperability glue that enables communication between Overture and the hardware.

HIL Component: is responsible for communicating directly with the hardware used in the HIL setup through serial connections. Although we only connect to a single instance of the `HIL Component` it would be possible to extend this to multiple instances as indicated in Figure 5.2. Finally, the `HIL Component` uses the `Log` for writing the data acquired from the Digital Acquisition (DAQ) to the file system.

Stimuli Controller: instructed by the `HIL Component` the `Stimuli Controller` provides the DUT with input signals prior to executing the functionality under test.

DUT Controller: instructs the external DUT to execute some function in software or hardware.

DAQ Driver: provides the software interface enabling the `HIL Component` to launch the DAQ so the data needed can be acquired from the DUT execution.

5.2 HIL setup prototype

This section presents the initial setup for the HIL system by describing the prototype as well as the co-execution of the model and the partial implementation.

5.2.1 Prototype description

The current prototype for this case implements a control algorithm for a mechanical device. This algorithm is divided into the following stages: 1) `Initialization`, 2) `Regulation Loop` and 3) `Results Logging`. We have modelled the algorithm in VDM-RT and continued by producing a preliminary implementation of the `Regulation Loop` that can be deployed to the DUT. The aim of this first approach is to combine in a single execution trace the models of `Initialization` and `Results Logging` with the implementation of `Regulation Loop` that is deployed on the DUT. Additionally, we are interested in measuring the execution time of the regulation cycle on real hardware in order to verify that the real-time constraints are met. This real-time information will also be incorporated into the VDM-RT model afterwards using the **`duration`** statement. The following paragraphs describe the structure and the functionality of the embedded hardware and software that the DUT is composed of.

Embedded hardware The embedded hardware supports communication with external logical interfaces; pulse-width modulation control of an actuator; communication with the workstation running the VDM-RT model (over a serial connection) and finally time measuring using pin toggling.

Embedded software The embedded software, executing on the DUT, is composed of hardware drivers, Functionality Under Test (FUT) and additional HIL support. The HIL support contains the logic to start the execution of the FUT upon request from the `DUT Controller` running on the workstation. This functionality can be divided into three steps:

1. **Wait for command:** The system waits for a parametrised command describing the functionality to execute.
2. **Serve HIL request:** The system initiates the execution of the FUT. This can be surrounded by pin toggling operations to measure the execution time of the function. Optionally additional pin toggling can be performed inside the function to measure the time it takes to reach different points during the execution.
3. **Return to DUT controller:** Once the system has completed the execution of the FUT it returns to the DUT controller so it can stop signal sampling and notify the model execution environment to resume model execution. Additionally it can return result values produced by the execution of the FUT.

5.3 HIL execution results

The co-execution of the model and the implementation made it possible to validate the real-time behaviour of the FUT component as well as the correct operation of its control logic. We have been able to validate that the time slot allocated to the regulation cycle is not overrun and obtained a precise time measurement that can be incorporated into the models. Additionally it has been possible to exercise the implemented control logic together with the modelled components on a number of scenarios. Finally, this HIL setup has facilitated the stepwise implementation from a system level design approach.

Our approach is further supported by tools for automated generation of the system implementation. For software components it is possible to produce Java and C++ from VDM using code generators [23, 6]. However, for hardware components there is currently no support for generating the implementation in a hardware description language. Even though automated hardware generation is not available in VDM-RT this language can be used to study component-level time constraints and to some extent hardware/software partitioning problems [24].

5.4 Final remarks

In this chapter we have presented the initial work we have carried out to enable HIL simulation using models written in VDM-RT. This proof-of-concept demonstrates the potential of our approach but there is significant future work remaining [4] to make this a useful feature for industrial use.

6

Code Generation

To support the development of CPS control logic, this chapter presents a code generation platform for VDM that enables code generation to different implementation languages. This work has been used to implement the VDM++-to-Java code generator available in the Overture tool as well as a proof-of-concept VDM++-to-C++code generator [6]. The code generation platform currently only supports VDM++, but there are plans to extend it to work for VDM-RT also (Subsection 7.2.3).

6.1 Generating the software implementation

Code generating a software model of a CPS can be an efficient way to transition to the realisation phase. This approach also minimises the chances of introducing inconsistencies in the software implementation that makes it deviate from the system specification due to manual translation of the model into code.

With the existence of many popular target languages it is common for code generators to provide support for multiple target languages in order to target a larger group of users. This can, however, easily lead to duplication of efforts when implementing code generators — especially if the target languages follow the same paradigms such that the rules used to code generate a source language are the same.

Ideally it should be possible to reuse the transformations used to code generate constructs of a source language. As an example, consider the VDM set comprehension `{x|x in set S & pred(x)}`, which constructs a new set from the elements of `S` for which `pred(x)` is true. In imperative languages such as Java and C++ this language construct is non-trivial to code generate since Java and C++ do not have similar constructs included. The same functionality can be obtained in those languages, but it requires use of multiple language constructs for iterating over a set, evaluating a predicate on each set member, adding elements to a resulting set etc.

The potential for different *backends* (a code generator that extends the code generation platform) to use the same transformations is particularly good when the target languages belong to the same paradigm (e.g. they are object-oriented or functional in style). In that case they will have many language constructs in common and thus face many of the same challenges with respect to code generation. When the same transformation can be used by different backends to code generate a source language construct it is beneficial to apply the transformation to the code generator input before it reaches the backend in order to obtain a transformed structure that is easier for a

backend to code generate. The idea is therefore to structure a code generator such that it is possible to select the transformations that will lead to a structure that requires the least effort for a backend to code generate. In this chapter we explore this approach to code generation in order to reduce the efforts needed to implement code generation for multiple backends.

6.2 Intermediate representations

One approach adopted by compiler developers is to transform the Abstract Syntax Tree (AST) specified in the source language into an Intermediate Representation (IR) that preserves the semantics of the input and from which the backend generates code in the target language. The IR helps managing the complexity of the compilation process by being independent of details specific to the source language and the target language. An IR obtained from the VDM AST serves a similar purpose by mitigating the complexity of generating code from a VDM model. This would, for example, enable the code generator to unify VDM functions and operations into the concept of a method as seen in a programming language such as Java. Then the backend only needs to treat a single (language) construct without having to distinguish between functions and operations.

In this work we address the challenges of code generating constructs where no obvious mapping exist. We do this by translating the VDM AST into an IR to which a series of transformations are applied. By transforming language constructs that are difficult to code generate into new tree structures, based on concepts that are easier to code generate, the implementation of a backend can be simplified. If the backend provides support for code generating the replacement constructs used by the transformations then it follows that the backend already supports code generation for the complex construct. In that case the complexity of the code generation process is comprehended entirely using tree transformations. The advantage of this approach is that the transformations can be made such that they are independent of the target language. This enables other backends to benefit from the same transformations when code generation is implemented for other languages.

6.3 Tree transformations

The difficulty of code generating a VDM model depends on the style of modelling and the target language. VDM is a multi-paradigm modelling language, using constructs of both the object-oriented and the functional paradigm, and therefore backends will experience situations where a construct does not have a one-to-one mapping into the target language.

One approach to simplifying code generation of a VDM model is to have the modeller refine the model such that it uses constructs that are easier to code generate. However, eliminating constructs that are problematic to a code generator using model rewriting, limits the modeller to use only a subset of the source language. This also clutters the model with details used to assist the backend in generating code from the model, thus going against the point to have a model that abstracts away details that do not contribute to obtaining the insight needed.

A more sophisticated approach is to have this kind of model refinement done at a later stage to make it transparent to the modeller and avoid restricting modelling to only a subset of the source language. This could be done by applying transformations to the IR such that constructs that are problematic to code generate get replaced with other IR constructs in order to obtain a *simplified IR* that is easier to code generate.

The use of transformations is part of a larger platform architecture that is used to construct backends. In Section 6.4 the architecture of this code generation platform is detailed to make it clear how it facilitates the construction of code generators.

6.4 Architecture of the code generation platform

The code generation platform, shown in Figure 6.1, takes a VDM++ model as input and use it to construct an IR that represents the generated code. After the IR has undergone a transformation process it is input to a backend that translates it into source code in a target language. To further detail the approach taken to construct code generators this section describes the architecture of the code generation platform and how it interacts with the backend of a target language.

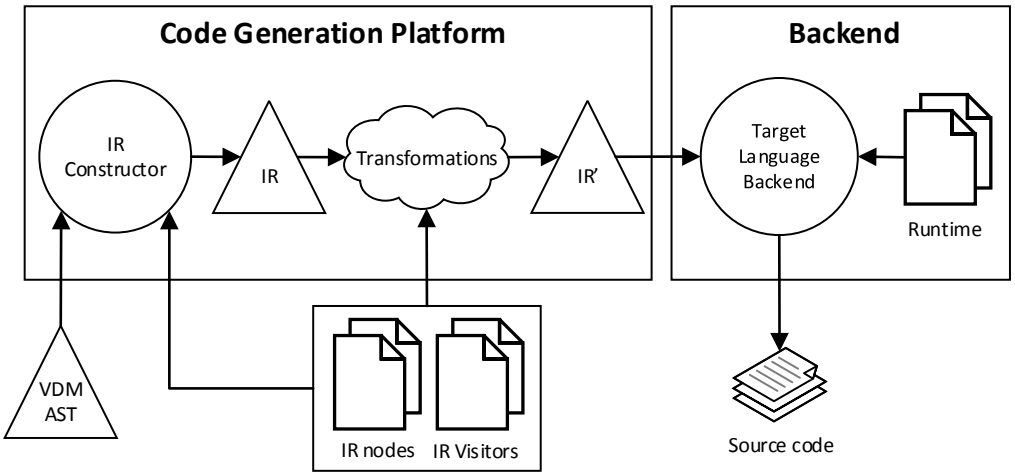

Figure 6.1: An overview of the code generation platform architecture

6.4.1 The intermediate representation life cycle

The IR as first constructed from the VDM AST represents a slightly simplified and extended version of the VDM model. For example, in this stage the IR has eliminated or replaced certain operators using other operators: writing a logical implication on the form $A \Rightarrow B$ where A and B are propositions, is convenient in a mathematical language such as VDM. However, since it is a derived and, not elementary operation of boolean logic, this operator is rarely seen in a programming language. Therefore the expression $A \Rightarrow B$ is represented as $\neg A \vee B$ in the IR, which is semantically equivalent.

Afterwards, code generation enters the transformation process where constructs that are difficult to code generate (or even unsupported by the backend) are translated into new tree structures that can be code generated. The IR before and after it has undergone the transformation process is denoted `IR` and `IR'` in Figure 6.1, respectively. Finally, the simplified IR is input to the backend that translates it into a target language.

6.4.2 Design of the intermediate representation

The IR nodes are generated using the ASTCreator tool [25], which is a SableCC [26] inspired tool. As shown in Figure 6.2 the ASTCreator takes a description of the AST as input and outputs nodes

from which concrete ASTs can be constructed. The generated AST structure uses bidirectional node relations which make it easier to search the tree both upwards (e.g. finding the enclosing class of a node) and downwards (e.g. looking up type information of child nodes). The nodes also have functionality for making changes to the tree structure, which is needed when nodes must be replaced with new tree structures during the transformation process.

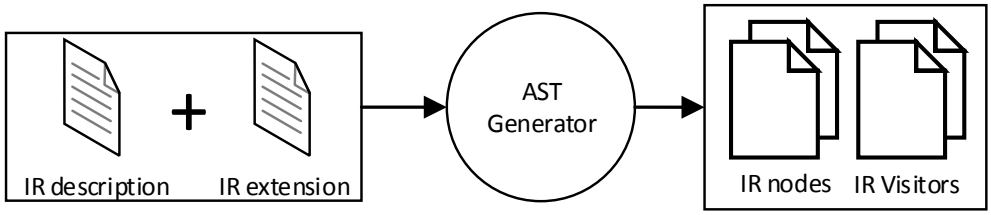

Figure 6.2: The ASTCreator produces the IR nodes and visitors based on the IR description

The ASTCreator also produces functionality to traverse the AST. These tree walkers or visitors are implemented using the visitor pattern and play an important role in the current AST architecture used in the Overture tool where they, for example, are used to implement the type checker and the interpreter [1]. Similarly, the `IR Constructor` is a visitor that traverses the VDM AST and constructs the `IR` from it.

6.4.3 The backend

The final step of the code generation process translates IR constructs into source code of the target language. When transformations have simplified the IR then ideally these mappings should be trivial. The process of mapping IR constructs into source code of the target language for the Java backend is done using the template-based technology, Apache Velocity [27]. Optionally, the generated code can make use of a runtime. As an example, the Java backend includes a runtime to represent VDM types and implementation for some of the VDM operators such as the sequence modification.

6.5 Final remarks

The code generation platform presented in this chapter supports construction of different backends and reduces the efforts needed to code generate a source language to multiple target languages. It achieves this by representing the generated code as an IR and subjects it to a series of semantic preserving transformations in order to obtain a tree structure that is easier for a backend to code generate. Since the IR is independent of any source and target language backends facing similar challenges during the code generation process, can use the same transformations to simplify their implementation.

Applying transformations to an IR has proven useful for implementing the Java and C++ backends, but the the approach also supports code generation for other source languages. Therefore we hope that the work presented in this chapter will be useful for others working with code generation.

7

Current Status and Future Plans

This chapter summarises the work contribution for the first half of this PhD, then describes the current future plan for the remaining part of the project, and ends with concluding remarks.

7.1 Summary of current work

The work carried out in the first half of this PhD has resulted in several technologies being developed to address the challenges of CPS development and facilitate knowledge transfer between the different phases of the development process: The view model provides a structured way to reason about system-level properties of a CPS and for stakeholders to organise the produced design artefacts in a meaningful way; The VDM-RT language extension enables analysis of the energy consumption of a CPU before the system is realised; and finally the HIL technology and the code generation platform provide ways to realise a system model.

7.2 Future plans

The subsections below describe ideas for potential future directions to take in this PhD project and how this will contribute to demonstrating the hypothesis introduced in Section 1.2.

7.2.1 Code generating traces:

Whether it is code generated or not, a software implementation is most commonly validated using testing. In VDM it is possible to specify a trace definition, which can be expanded into a large collection of tests that when executed enables detection of deficiencies in the model (e.g. missing pre-conditions) [28]. The body of a trace definition is similar to a regular expression and describes the sequence of operation calls for each test that the trace definition expands into. If the code generation platform is extended to support code generation of traces then the tests specified at the modelling level can be used throughout system realisation to validate the software implementation – also after changing the generated code. So by completing this future plan item, this PhD project will provide a better coverage of the entire development cycle, by taking the aspect of testing into account.

7.2.2 Formally verifying the output of the code generator:

It would be interesting to demonstrate the proof-of-concept of formally verifying the transformations used by the code generation platform, in order to ensure that the generated code preserves the semantics of the input model. This would in principle make testing of the generated code unnecessary, but many of the transformations are non-trivial and would require a considerable amount of work to verify formally. However, this could be done for some of the transformations in order to show the way, leading by example, such that other people working with code generators can adopt this approach. This future plan item therefore contributes to improving the quality of the output of the code generator in order to avoid introducing problems in the software implementation and potentially reduce the amount of testing needed.

7.2.3 Extending the coverage of the code generation platform:

Currently three student projects are being carried out (two Master thesis projects and one R&D project) based on the code generation platform. The first Master thesis project carries out work to provide support for the concurrency elements of VDM++. The second Master thesis project tries to code generate the distribution aspects of VDM-RT by taking into account that an object may invoke another object deployed on a different CPU. Finally, an R&D project uses the code generation platform to develop a CPS using a Raspberry Pi and documents the challenges encountered. The outcomes of these projects will be extended further in the remaining part of this PhD project. This future plan item therefore adds to the coverage of the code generation platform by allowing the concurrency and distribution elements of VDM-RT to be generated. The R&D project serves as a case study to validate the code generation platform and the outcomes for this project are expected to be 1) a stepwise method for realising CPS models using code generation and 2) suggestions for improvements to the code generation platform (possibilities to configure code generation etc.).

7.2.4 INtegrated Tool chain for model-based design of CPSs (INTO-CPS)

By the beginning of 2015 a new H2020 project that focus on developing tools for comprehensive modelling of CPSs will start off. The project, which is named INTO-CPS, addresses challenges such as network distribution, combined use of different co-simulation technologies and system realisation. INTO-CPS aims to develop a tool that is based on Functional Mockup Interface (FMI)-compatible co-simulation, meaning that the tool can be co-simulated with other tools that meet the FMI standard [29]. Therefore, INTO-CPS is highly related to the work carried out in this PhD and covers several work tasks that involve things such as HIL simulation, software-in-the-loop simulation and code generation. Although this PhD is not officially tied to the INTO-CPS project it brings further collaboration possibilities in order to improve and extend many of the contributions of this PhD.

7.2.5 Going abroad to Berkeley:

This PhD project has received a grant of 25.000 DKK to go to Berkeley in spring 2015 for a period of two months and work together with Professor Edward A. Lee who is doing research within the area of CPSs. Edward A. Lee is involved in projects such as Ptolemy [18] and Ptides [30], where software tools for modelling, simulation and realisation (including code generation) of CPSs are developed. The researchers working at the University of Berkeley are highly specialized within the field of research of this PhD project. This makes visiting this research institution an ideal opportunity in order to become more knowledgeable on code generation for CPSs and establish

new connections with researchers within the area of research of this PhD. The plan is to work with the technologies developed at Berkeley and enhance them in regards to the work of this PhD – possibly by extending their work on code generation or enabling models developed using their tools to be simulated together with Crescendo models in a new FMI compatible version of the tool developed in the INTO-CPS project (Subsection 7.2.4).

7.2.6 Optimising harvesting processes:

Aarhus University is currently a partner in a project supported by the High Technology Foundation with a vision to optimise offline and online logistics planning of harvesting processes. In this project the harvesting process is modelled in VDM and the simulation is visualised with a graphical user interface running on top. One purpose of the model is to compare outcomes of simulations based on different harvesting strategies in order to find the most efficient one. The model also serves as a case study in the work with the code generation platform in order to speed up the process of moving to the realisation phase as well as to make the simulation run faster using the generated code. Use of strategies and code generation to optimise harvesting processes are interesting subjects that will be further researched in the second half of this PhD project. At some point this PhD project may try to code generate the distribution aspects of the model, i.e. the control logic of the resources participating in the harvesting operation (harvesters, service units, storages etc.) is executed on different hardware platform that communicate over a network.

7.3 Concluding remarks

The work carried out in the first half of this PhD project has enhanced modelling of CPSs by contributing with a view model for co-models and a language extension for VDM-RT enabling analysis of the energy consumption of the CPUs in a CPS (Chapters 3 and 4, respectively). Furthermore, the HIL technology enables the system to be realised gradually and increases the fidelity of the simulation by including real hardware in the system simulation (Chapter 5). This approach improves prediction of the system properties for the final version of the system and has the potential to reduce the expensive analysis needed with a complete system realisation. Finally, the code generation platform automates parts of the realisation process by generating the software implementation directly from the system model, which can be an efficient way to transfer the knowledge obtained during modelling to the realisation phase (Chapter 6).

Some of the future plan items aim to expand the coverage of the code generation platform, and case studies will be used to validate and improve the approach. The remaining part of this PhD project further aims to improve code generation by providing ways to make guarantees about the generated code or to estimate the code quality using code generated traces. It is believed that addressing the proposed future plan items will further contribute to enhancing modelling of CPSs and make realisation of these systems more efficient.

Publications

Published

- Peter Würtz Vinther Jørgensen, Morten Larsen, and Luis D. Couto. A Code Generation Platform for VDM. In the proceedings of the 12th Overture workshop, August 2014.
- José Antonio Esparza Isasa, Peter Würtz Vinther Jørgensen, and Claus Ballegård Nielsen, Modelling Energy Consumption in Embedded Systems with VDM-RT. In the proceedings of the 4th International ABZ conference, July 2014.
- José Antonio Esparza Isasa, Peter Würtz Vinther Jørgensen, and Peter Gorm Larsen, Hardware In the Loop for VDM-Real Time Modelling of Embedded Systems. In the proceedings of MODELSWARD 2014, Second International Conference on Model-Driven Engineering and Software Development, January 2014.
- Peter Würtz Vinther Jørgensen, Kenneth Lausdahl, and Peter Gorm Larsen. An Architectural Evolution of the Overture Tool. In the proceedings of the 11th Overture workshop, August 2013.
- Peter Würtz Vinther Jørgensen and Peter Gorm Larsen, Towards an Overture Code Generator. In the proceedings of the 11th Overture workshop, August 2013.
- Nick Battle, Anne Haxthausen, Sako Hiroshi, Peter Jørgensen, Nico Plat, Shin Sahara, and Marcel Verhoef. The Overture Approach to VDM Language Evolution. In the proceedings of the 11th Overture workshop, August 2013.

Submitted

- Luis Diogo Couto, Peter Würtz Vinther Jørgensen, Joey W. Coleman, and Kenneth Lausdahl. Migrating from a Performant to an Extensible Architecture in the Overture Tool. Submitted to the 30th Symposium On Applied Computing: (SAC 2015), April 2015.

Awaiting publication venue

- Peter Würtz Vinther Jørgensen and Sune Wolff. Five Views of a Collaborative Model. Withdrawn short paper submitted to Simultech 2014, August 2014.

Courses and Dissemination

Table B.1 shows courses completed, planned and currently in progress for this PhD. So far 20 ECTS points of the (approximately) 30 ECTS points have been earned. By completing the courses currently in progress and the course planned, this PhD project will earn a total of 31 ECTS points.

Table B.1: Overview of courses for this PhD.

Course name	Start date	End date	Completion status	Credits
ICCES 2013	18/06/2013	05/07/2013	Completed	5 ECTS
Compilation (Q1/Q2)	26/08/2013	18/12/2013	Completed	10 ECTS
Programming Language Paradigms (Q4)	03/04/2014	30/05/2014	Completed	5 ECTS
Semantics (Q1/Q2)	26/08/2014	18/12/2014	In progress	5 ECTS
Journal Club on Scientific Writing in Engineering and Natural Sciences (Q2)	15/09/2014	19/12/2014	In progress	3 ECTS
Science Teaching - Module 1: Introduction to Science Teaching (Q2)	25/11/2014	16/12/2014	Planned	3 ECTS
Total:				31 ECTS

Bibliography

[1] P. W. Jørgensen, K. Lausdahl, and P. G. Larsen, "An Architectural Evolution of the Overture Tool," in *The Overture 2013 workshop*, August 2013.

[2] L. D. Couto, P. W. Jørgensen, K. G. Lausdahl, and J. Coleman, "Migrating from a Performant to an Extensible Architecture in the Overture Tool," in *Submitted to the 30th Symposium On Applied Computing: (SAC 2015)*, April 2015.

[3] J. A. E. Isasa, P. W. Jørgensen, and C. Ballegaard, "Modelling Energy Consumption in Embedded Systems with VDM-RT," in *Proceedings of the 4th International ABZ conference.*, July 2014.

[4] J. A. E. Isasa, P. W. Jørgensen, and P. G. Larsen, "Hardware In the Loop for VDM-Real Time Modelling of Embedded Systems," in *MODELSWARD 2014, Second International Conference on Model-Driven Engineering and Software Development*, January 2014.

[5] P. W. Jørgensen and P. G. Larsen, "Towards an Overture Code Generator," in *The Overture 2013 workshop*, August 2013.

[6] P. W. Jørgensen, L. D. Couto, and M. Larsen, "A Code Generation Platform for VDM," in *The Overture 2014 workshop*, June 2014.

[7] P. G. Larsen, N. Battle, M. Ferreira, J. Fitzgerald, K. Lausdahl, and M. Verhoef, "The Overture Initiative – Integrating Tools for VDM," *SIGSOFT Softw. Eng. Notes*, January 2010.

[8] C. B. Jones, *Systematic Software Development Using VDM*. Englewood Cliffs, New Jersey: Prentice-Hall International, second ed., 1990. ISBN 0-13-880733-7.

[9] P. G. Larsen, K. Lausdahl, and N. Battle, "The VDM-10 Language Manual," Tech. Rep. TR-2010-06, The Overture Open Source Initiative, April 2010.

[10] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef, *Validated Designs for Object–oriented Systems*. Springer, New York, 2005.

[11] R. S. Sandford Friedenthal, Alan Moore, *A Practical Guide to SysML*. Friendenthal, Sanford: Morgan Kaufman OMG Press, First ed., 2008. ISBN 978-0-12-374379-4.

[12] D. Karnopp and R. Rosenberg, *Analysis and simulation of multiport systems: the bond graph approach to physical system dynamic*. MIT Press, Cambridge, MA, USA, 1968.

[13] J. F. Broenink, "Modelling, Simulation and Analysis with 20-Sim," *Journal A Special Issue CACSD*, vol. 38, no. 3, pp. 22–25, 1997.

[14] L. Baresi, G. Ferretti, A. Leva, and M. Rossi, "Flexible logic-based co-simulation of modelica models," in *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*, pp. 635 –640, july 2012.

[15] P. Fritzson and V. Engelson, "Modelica - A Unified Object-Oriented Language for System Modelling and Simulation," in *ECCOP '98: Proceedings of the 12th European Conference on Object-Oriented Programming*, pp. 67–90, Springer-Verlag, 1998.

[16] MathWorks, "http://www.mathworks.com," October 2011. Matlab official website.

[17] LMS Engineering Innovation, "AMESim: Advanced Modeling Environment for performing Simulations of engineering systems," 2012. http://www.lmsintl.com/imagine-lab-amesim-rev-11.

[18] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous System," in *Int. Journal of Computer Simulation*, 1994.

[19] S. Wolff, *Methodological Guidelines for Modelling and Design of Embedded Systems*. PhD thesis, Aarhus University, Department of Engineering, 2013.

[20] P. W. Jørgensen, "Evaluation of Development Process for co-models," Master's thesis, Aarhus University/Engineering College of Aarhus, December 2012.

[21] J. A. E. Isasa and P. G. Larsen, "Modelling Different CPU Power States in VDM-RT," in *Proceedings of the 11th Overture Workshop 2013*, Aarhus University, June 2013.

[22] J. A. E. Isasa, P. G. Larsen, and F. O. Hansen, "Energy-Aware Model-Driven Development of a Wearable Health Care Device," in *Manuscript under preparation for the 4th Symposium FHIES/SEHC.*, June 2014.

[23] CSK, "VDMTools homepage." *http://www.vdmtools.jp/en/*, 2007.

[24] J. A. E. Isasa, P. G. Larsen, and K. Bjerge, "Supporting the Partitioning Process in Hardware/Software Co-design with VDM-RT," in *Proceedings of the 10th Overture Workshop 2012*, School of Computing Science, Newcastle University, 2012.

[25] "The ASTCreator website," 2014. https://github.com/overturetool/astcreator.

[26] "The SableCC website," 2014. http://www.sablecc.org/.

[27] "The Apache Velocity website," 2014. http://velocity.apache.org/.

[28] P. G. Larsen, K. Lausdahl, and N. Battle, "Combinatorial Testing for VDM," in *Proceedings of the 2010 8th IEEE International Conference on Software Engineering and Formal Methods*, SEFM '10, (Washington, DC, USA), IEEE Computer Society, September 2010.

[29] T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel, "The Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models," in *Proceedings of the 9th International Modelica Conference*, September 2012.

[30] Y. Zhao, J. Liu, and E. A. Lee, "A Programming Model for Time-Synchronized Distributed Real-Time Systems," in *13th IEEE Real Time and Embedded Technology and Applications Symposium, 2007. RTAS '07*, pp. 259 – 268, April 2007.

Peter W. V. Jørgensen, Enhanced Modeling and Efficient Realisation of Cyber-Physical Systems, 2014

Department of Engineering
Aarhus University
Inge Lehmanns Gade 10
8000 Aarhus C
Denmark

Tel.: +45 8715 0000