Mälardalen University Press Licentiate Thesis
No.224

Raising Abstraction of Timing Analysis through Model-Driven Engineering

Alessio Bucaioni

December 2015

School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden

ISSN
ISBN —
Printed by Mälardalen University, Västerås, Sweden

Abstract

The complexity of software running on vehicular embedded systems is constantly increasing and this negatively affects its development costs and time to market. One way to deal with these issues is to boost abstraction in the form of models to (i) ease the reasoning about the system architecture, (ii) automate certain stages of the development, (iii) early detect flaws in the system architecture through fundamental analysis and (iv) take appropriate countermeasures before the system is implemented.

Considering the importance of timing requirements in the design of software for vehicular embedded systems, in this licentiate thesis we leverage Model-Driven Engineering for realizing a semi-automatic approach which allows the developer to perform end-to-end delay timing analysis on design models, without having to manually model timing elements and set their values.

The proposed approach, starting from a design model of an automotive software functionality, automatically generates a set of models enriched with timing elements whose values are set at generation time. End-to-end delay timing analysis is run on the generated models and, based on the analysis results, the approach automatically selects the generated models which better meet a specific set of timing requirements.

Sammanfattning

Nuförtiden finns inbyggda datorsystem i de flesta elektroniska och elektriska produkter. Allt ifrån strömbrytare och mikrovågsugnar till bilar och tåg är beroende av moderna inbyggda datorsystem. I fordonsindustrin, ökar ständigt antalet inbyggda system som ersätter hydrauliska och mekaniska delar som inte kan leverera moderna tjänster som kollisionsskydd och antisladdsystem. Samtidigt ökar komplexiteten av dessa datorsystem och deras mjukvara, och detta påverkar negativt utvecklingskostnader och tid. Ett sätt att minska dessa problem är att använda abstraktioner i form av modeller för att i) enklare resonera över systemets arkitektur, ii) automatisera vissa utvecklingsfaser, iii) tidigt spåra brister i systemets arkitektur genom grundläggande analyser och iv) åtgärda brister innan systemet färdigställs.

I denna avhandling använder vi modelldriven utveckling för att skapa en ny metod som förenklar utvecklingen av inbyggda system för fordon. Detta sker genom att utvecklaren beskriver systemet som en abstrakt arkitekturmodell som används av metoden för att automatiskt generera ett antal möjliga konkreta modeller. End-to-end tidsanalys körs på de genererade modellerna och analysresultat används av metoden för att automatiskt välja bland de genererade modellerna dem som uppfyller de uppsatta tidskraven bäst.

Acknowledgements

First and foremost, my utmost gratitude to my supervisors Mikael Sjödin, Antonio Cicchetti and Federico Ciccozzi whose guidance is making this a possible and pleasant journey. They are mentors, colleagues and friends and they are helping me in becoming a better person before than a better researcher.

I would like to express my deepest gratitude to my "buddy" Saad Mubeen, who helped me in moving the first steps into the research world. We shared unforgettable moments and i could not ask for a better "buddy".

I would like to thank Kurt-Lennart Lundbäck and all the people from Arcticus Systems AB for giving me the possibility to work in such a successful company without never interfere with my researches.

I would like to thank all the administrative staff, especially Carola Ryttersson and Susanne Fronnå for making "paper-work" easier.

I would like to thank my friends and colleagues at the department for all the good and funny moments and for the inspiration they give me each day.

I will never thank enough my family for supporting me, even financially, and for never discussing my decisions, even when these brought me far away from their life. Without you, i would be lost.

I would like to thank all my friends, both in Italy and Sweden, for standing by my side and for never making me feel alone. Especially, i would like to thank Manuel, Mirco and Giada for having brought some joy in a difficult period of my life.

I would like to thank my grandpas Vincenzo and Terzilio, my aunts Ines and Nunziata and my uncle Pasquale for watching over me.

Last, but not least i would like to thank the One above us all, God, for answering my prayers and for giving me the strength to never throw in the towel.

Alessio Bucaioni
Västerås, December, 2015

List of Publications

Publications Included in this Licentiate Thesis[1]

Paper A – A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation from EAST-ADL, *Alessio Bucaioni, Saad Mubeen, Federico Ciccozzi, Antonio Cicchetti, Mikael Sjödin*, Conditionally accepted at the Journal of Systems and Software (JSS).

Paper B – Exploring Timing Model Extractions at EAST-ADL Design -level Using Model Transformations, *Alessio Bucaioni, Saad Mubeen, Antonio Cicchetti, Mikael Sjödin*, IEEE 12th International Conference on Information Technology: New Generations (ITNG), Las Vegas, Nevada (USA), April, 2015.

Paper C – Raising Abstraction in Timing Analysis for Vehicular Embedded Systems through Model-Driven Engineering, *Alessio Bucaioni*, Doctoral Symposium at Software Technologies: Applications and Foundations (STAF), L'Aquila, Italy, July, 2015. *Best paper award.*

Paper D – Anticipating Implementation-Level Timing Analysis for Driving Design-Level Decisions in EAST-ADL, *Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, Mikael Sjödin*, 1st International Workshop on Modelling in Automotive Software Engineering (MASE) at ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (Models), Ottawa, Canada, September, 2015.

[1] The included publications are reformatted to comply with the licentiate thesis printing format

Related Publications not Included in this Thesis

Comparative Evaluation of Timing Model Extraction Methodologies at EAST-ADL Design Level, *Alessio Bucaioni, Saad Mubeen, Federico Ciccozzi, Antonio Cicchetti, Federico Ciccozzi, Mikael Sjödin*, IEEE 12th International Conference on Embedded Software and Systems (ICESS), New York, New York, August, 2015.

Towards a metamodel for the Rubus Component Model, *Alessio Bucaioni, Antonio Cicchetti, Mikael Sjödin*, 1st International Workshop on Model-Driven Engineering for Component-Based Software Systems at ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (Models), Valencia, Spain, September, 2014.

OSLC Tool Integration and Systems Engineering - The Relationship Between The Two Worlds, *Mehrdad Saadatmand, Alessio Bucaioni*, 40th Euromicro Conference on Software Engineering and Advanced Applications, Verona, August, 2014.

Demonstrator for modeling and development of component-based distributed real-time systems with Rubus-ICE, *Alessio Bucaioni, Saad Mubeen, John Lundbäck, Kurt-Lennart Lundbäck, Jukka Mäki-Turja, Mikael Sjödin*, Open Demo Session of Real-Time Systems (RTSS@Work) at Real Time Systems Symposium (RTSS), Vancouver, Canada, December, 2013.

Other Publications

Understanding bidirectional transformations with TGGs and JTL, *Alessio Bucaioni, Romina Eramo*, 2nd International Workshop on Bidirectional Transformations (BX) at European Joint Conferences on Theory and Practice of Software (ETAPS), Roma, Italy, March, 2013.

A Model-Based Testing Framework for Automotive Embedded Systems, *Raluca Marinescu, Mehrdad Saadatmand, Alessio Bucaioni, Cristina Seceleanu, Paul Pettersson*, 40th Euromicro Conference on Software Engineering and Advanced Applications, Verona, August, 2014.

EAST-ADL Tailored Testing: From System Models to Executable Test Cases, *Raluca Marinescu, Mehrdad Saadatmand, Cristina Seceleanu, Paul Pettersson, Alessio Bucaioni*, MRTC Report MDH-MRTC-278/2013-1-SE, September, 2013.

To my princesses Chiara and Elisa

Contents

I

Thesis

Chapter 1

Introduction

Nowadays, embedded systems play a prevailing role in everyday life as they are widely employed in most electronic and electrical products, from microwave ovens to trains and cars. In the specific case of the automotive domain, embedded systems replace many of the hydraulic and mechanical parts of a vehicle, improving the driving experience, the comfort of the passengers and the safety of the vehicle. The growing complexity of software running on embedded systems results in an increasing complexity of its development, which in turn negatively affects the development costs and time to market [1].

The software engineering community has agreed on three instruments when dealing with the increasing complexity of software and its development: i) *abstraction*, ii) *separation of concerns* and iii) *automation*. In the midst of the many methodologies advocating these three instruments when developing software systems, Model-Driven Engineering (MDE) has progressively gained recognition and industrial attention in the last 15 years [2].

MDE is a paradigm which aims at raising the level of abstraction of software development by shifting the focus from coding to modeling activities. In this context, *models* and *model manipulations* are promoted as first-class citizens. A model represents an abstraction of the software system, from a particular point of view [3]. Models promote separation of concerns by describing the software system by means of different models each of which highlighting different concerns corresponding to different views. The set of rules and constraints for the construction of valid models are specified in the so-called *metamodel* [3]. The relation between a model and its metamodel is called conformance [3]. According to the MDE paradigm, a software system is developed

by means of model manipulations, where abstract models are refined into more detailed ones, until code is automatically generated. Model manipulations are performed by means of *model transformations* [4] which automatically translate a source model into a target model while ensuring their conformance to their respective metamodels.

In the automotive domain, the adoption of MDE resulted in the standardization of an architectural description language, EAST-ADL [5], which is used for modeling product-lines of vehicular embedded systems. EAST-ADL proposes a view over the software development process composed by four different *abstraction levels*, which implicitly ensure separation of concerns through the different engineering phases. Each abstraction level is described by means of metamodeling constructs and aims at hiding unnecessary information from higher abstraction levels. EAST-ADL defines a set of activities to perform for each abstraction level, based on the expressible concepts. Figure 1.1 shows the EAST-ADL abstraction levels together with the related languages and activities.

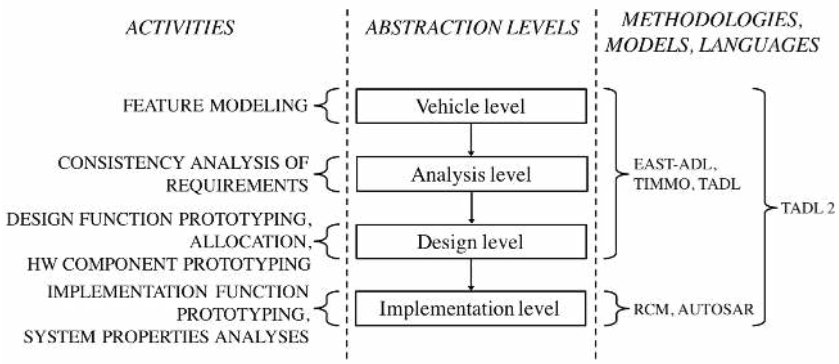


Figure 1.1: EAST-ADL Abstraction Levels Together With The Related Languages and Activities

The highest abstraction level is represented by the *vehicle level*, which captures information regarding the system's functionality. At the *analysis level*, by using formal notations, vehicle functions are expressed in terms of behaviors and interfaces. Yet, design and implementation details are omitted. At this stage, high level analysis for consistency checking of the requirements can be performed. At the *design level*, analysis-level artifacts are refined with design-oriented details, such as software, middleware and hardware separation as well

as allocation of software. At the *implementation level*, artifacts introduced at the design level are refined with implementation details for enabling system properties analyses, e.g., end-to-end delay timing analysis. At this stage, component models[1] (e.g., Rubus Component Model (RCM) or AUTOSAR) are used to model the system in terms of components and their interactions. The output of this level is a complete software architecture which can be used for code generation.

In this thesis, we consider RCM as the language for the implementation level [7]. RCM is a component model for the development of resource-constrained embedded real-time software systems. It is developed by Arcticus Systems AB in collaboration with Mälardalen University and it is currently adopted by several international companies, e.g., as alternative to AUTOSAR. Considering the importance of timing analysis for vehicular embedded systems [8] [9], RCM implements state-of-the-art end-to-end delay timing analysis [10] [11]. End-to-end delay timing analysis is used for providing evidence that behaviors of the software system meet a specific set of timing requirements. Within EAST-ADL, end-to-end delay timing analysis gives meaningful results only if run on implementation models which are currently manually defined starting from design models. Unfortunately, when dealing with systems of industrial size, this manual process becomes soon overwhelming, leading to the creation of a very limited subset of implementation models only . By having automation support among the different EAST-ADL abstraction levels, and in our specific case between EAST-ADL design and implementation levels, it would be possible to enable swift transitions avoiding error-prone and tedious manual activities (model manipulations). Also, it would be possible to leverage end-to-end delay timing analysis results for driving design decisions avoiding late discoveries of unacceptable quality of service with respect to timing requirements [8].

1.1 Thesis Contribution

In this licentiate thesis, we leverage MDE for realizing a semi-automatic approach which allows the developer to perform timing analysis on EAST-ADL design models without having to manually specify their timing elements. More precisely, starting from an EAST-ADL design model of an automotive software

[1] A *software component* is a software element which i) conforms to a component model, ii) can be deployed independently and iii) can be composed according to a composition standard [6]. A *component model* specifies i) the properties of software components and ii) how software components can be composed [6].

functionality, the proposed approach automatically generates a set of RCM models enriched with timing elements. End-to-end delay timing analysis is run on the generated RCM implementation models and, based on the results, the approach supports the selection of the generated RCM implementation model (or set of models) which better meets a specific set of timing requirements. To this end, the following contributions are identified:

1. *RCM metamodel*. Model transformations are required for achieving a full-fledged MDE approach for leveraging end-to-end delay timing analysis at design level. Moreover, model transformations are specified on the involved metamodels. While EAST-ADL provides a metamodel definition for the design level, RCM has not been described by metamodeling means. This contribution provides a metamodel definition for RCM.

2. *Model transformation between EAST-ADL design level metamodel and RCM metamodel*. This contribution allows the automatic translation from an EAST-ADL design model into a set of meaningful RCM implementation models that can be used as input for running end-to-end delay timing analysis.

3. *Selection mechanism*. Based on end-to-end delay timing analysis results, this contribution supports the selection of the RCM implementation model (or set of models) which better meets a specific, non-empty set of timing requirements.

1.2 Terminology

In this section we introduce the terms that we use in the remainder of this licentiate thesis.

- *End-to-end delay timing analysis*. Schedulability analyses are *a priori* analysis techniques used for ensuring that the software system meets its timing requirements. End-to-end delay timing analysis are well-established schedulability analyses which calculate upper bounds on the response time and delays of event chains distributed over several nodes or in the system. We will refer to *end-to-end delay timing analysis* simply as *timing analysis*.

- *EAST-ADL design level*. We will refer to *EAST-ADL design level* simply as *design level*.

- *EAST-ADL implementation level*. We will refer to *EAST-ADL implementation level* simply as *implementation level*.
- *EAST-ADL design level model*. We will refer to *EAST-ADL design level model* simply as *design model*.
- *EAST-ADL implementation level model*. We will refer to *EAST-ADL implementation level model* simply as *implementation model*.
- *RCM implementation model*. Different component models can be used at the implementation level. As aforesaid, we decided to use RCM as the target language for the implementation level. With the term *RCM implementation model* we refer to a RCM model used at the implementation level.
- *Timing requirement*. We will refer to *timing requirement* as the required timing performance specified on the vehicle functionality. A typical example would be: "the time between the request from the driver and the response of the physical system shall be lower than 10 milliseconds".
- *Timing property and timing value*. *Timing property* is a property which concerns the timing behavior of the software. A typical example of a timing property is the worst-case execution time of a function. *Timing value* is the actual value of a timing property.
- *Timing element*. We refer to *timing element* as a modeling element which represents a timing property or requirement.
- *Non-bijective model transformation*. A *non-bijective model transformation* is a model transformation that translates a single source model into multiple target models.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 describes the research plan in terms of research goals, challenges and contributions. Chapter 3 discusses conclusions and future directions. The second part of the thesis consists of Chapter 4 through Chapter 7 and describes the research contributions in terms of research publications.

Chapter 2

Research Plan

This chapter discusses the adopted research plan in terms of research goal, research challenges (RCs), research methodology and research contributions (RCOs).

2.1 Research Goal

Timing requirements are crucial in the design of the software running on vehicular embedded systems [9] [8]. Timing analysis is a primary means by which timing requirements are verified. However, design decisions are usually not driven by timing analysis results as timing analysis is usually performed after the design activities [12]. To this end, we believe that anticipating timing analysis at design level can mitigate software development issues (e.g., cost, time-to-market) as it would avoid late discoveries (i.e., during the testing activities) that the system delivers services of unacceptable quality with respect to timing [10] [8]. Within an EAST-ADL based methodology, the way towards early timing analysis is hampered by the weak linkage between the modeling language used at the implementation level (where timing analysis is usually performed) and the language used at the design level.

The goal of this research work is to enable timing analysis at design level for supporting design decisions. More specifically, we aim at providing an approach which gives automation means for seamlessly linking design and implementation level.

2.2 Research Challenges

Considering the research goal, the following RCs have been formulated and used as main drivers for this research work.

RC 1. Definition of a metamodel for RCM.
While EAST-ADL does not fully embrace the MDE paradigm, it still provides metamodel definitions for the languages adopted at each abstraction level. Consequently, MDE seems to be the natural choice for automating an EAST-ADL based methodology.

According to the MDE paradigm, metamodels and model transformations are crucial for providing automation within the software development: the former serves for regulating the specification of models while the latter for automating their manipulations. In particular, it is very beneficial that all the modeling languages involved in the development process are provided with a metamodel definition. Considering that we want to provide automation between the design and implementation levels, both the languages used at these levels should be provided with metamodel definitions. While for the design level a metamodel definition exists, our challenge is the definition of a metamodel for RCM, the language we use at the implementation level. In particular, the metamodel should be defined bearing in mind the following:

1. *Backward compatibility:* the metamodel should not hinder a migration of legacy RCM artifacts.

2. *Maintainability:* the metamodel should be easy to update and refine.

RC 2. Definition of a mapping between EAST-ADL design level metamodel and RCM metamodel.
Separation of concerns and abstraction are the pillars on which EAST-ADL relies and the EAST-ADL abstraction levels are designed to ensure these principles. Within EAST-ADL, timing analysis can only be performed at the implementation level, since timing properties are not entailed in higher abstraction levels.

One way to leverage timing analysis results at design level is the definition of a transparent and automatic process able to translate design models to implementation models, i.e., RCM implementation models, on which timing analysis can be performed. In fact, RCM implementation models contain timing elements, e.g., clocks, control-flow ports, to mention a few, that can not be modeled at the design level. However, these elements represent variability points in the transition from design to implementation level, meaning that

more than one RCM implementation model can be a valid translation of a given design model [13].

The challenge is to define and implement a semi-automatic translation able to produce a set of RCM implementation models for a given design model avoiding any manual, error-prone and time-consuming model manipulations.

RC 3. Definition of a mechanism for the selection of the best RCM model for a specific, non empty set of timing requirements.
This represents the last step in the process of leveraging timing analysis at design level for enabling guided design decisions. After the RCM implementation models are generated, timing analysis is run. Based on the timing analysis results, the challenge is to define a mechanism able to select the RCM implementation model which better meets a specific, non empty set of timing requirements. The mechanism should be able to select multiple equally good RCM implementation models. Once the RCM implementation model (or set of models) is identified, its analysis results should be propagated back to design level and made accessible to the developer. In case no RCM implementation model satisfies the set of timing requirements, refinements on the design model are required.

2.3 Research Contributions

The main contribution of this licentiate thesis is the definition of an approach for seamlessly linking design and implementation level in order to enable timing analysis at design level. This is needed to allow timing analysis results to drive design decisions. Figure 2.1 provides a graphical representation of the proposed approach.

Starting from a *Design Model* representing an automotive software functionality, the approach is able to generate a set of corresponding *RCM Model(s)*. The generated set contains all the *RCM Model(s)* which are meaningful for the considered *Timing Analysis*. *RCM Model(s)* are equipped with timing elements. While the generation of these elements is fully automated, their completion with timing properties is entrusted to the developer. That is to say, the developer drives the automatic generation of all the meaningful combinations of timing elements by inserting timing properties, via configuration files, only once per element instead of having to manually edit all the generated *RCM Model(s)*. At this point, *Timing Analysis* is run on the generated *RCM Model(s)* resulting in a set of *Analysis Result(s)*. These results are checked against a non-

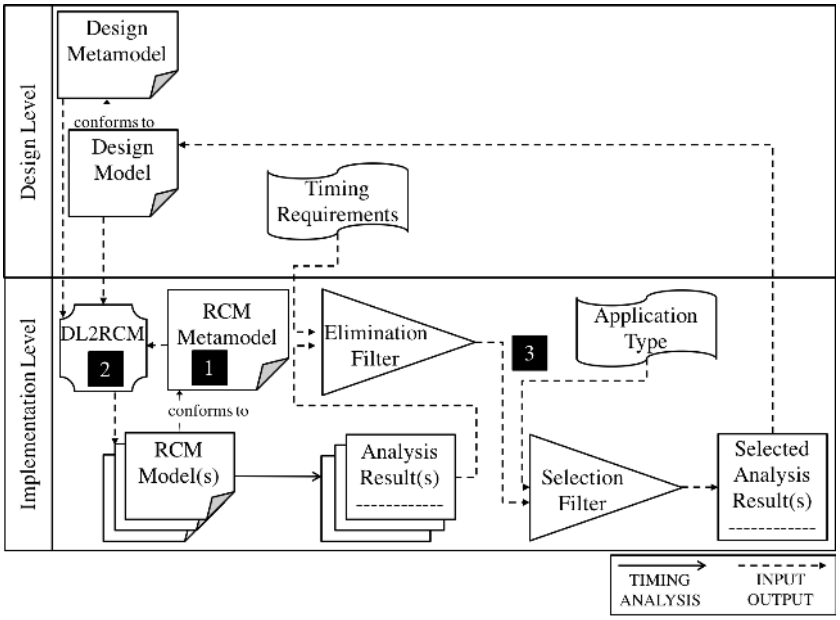


Figure 2.1: Research Contributions

empty set of *Timing Requirements* and the *RCM Model* which better satisfies the *Timing Requirements* is selected; note that multiple *RCM Models* might be equally good and thereby selected. Eventually, the corresponding analysis results, i.e., *Selected Analysis Result(s)*, are propagated back to the developer by means of annotations to the design model. Figure 2.1 provides a breakdown of the main contribution in specific RCOs.

RCO 1 - RCM metamodel. This contribution, marked as 1 in Figure 2.1, provides a metamodel definition for RCM as the first step in the process of providing automation means. The RCM metamodel has been realized within the Eclipse Modeling Framework[1] (EMF) as an EMF model, and it has been defined with particular attention to backward compatibility and maintainability.

In order to address the first goal, i.e., backward compatibility, we reverse-engineered the internal representation of RCM into the Rubus-ICE for polishing redundancies and optimizing model traversals. This resulted in the addi-

[1] http://www.eclipse.org

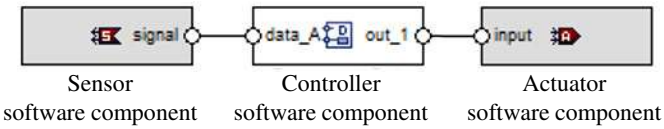
tion of 6 modeling elements (e.g., connectors) and the refinement of element hierarchies (e.g., ports and data element hierarchies). With respect to maintainability, building-up the development environment on the RCM metamodel allows to separate the modeling elements from their rendering and features of the development environment.

RCO 2 - DL2RCM transformation. This contribution, marked with 2 in Figure 2.1, provides a model-to-model transformation between design models and RCM implementation models (DL2RCM).

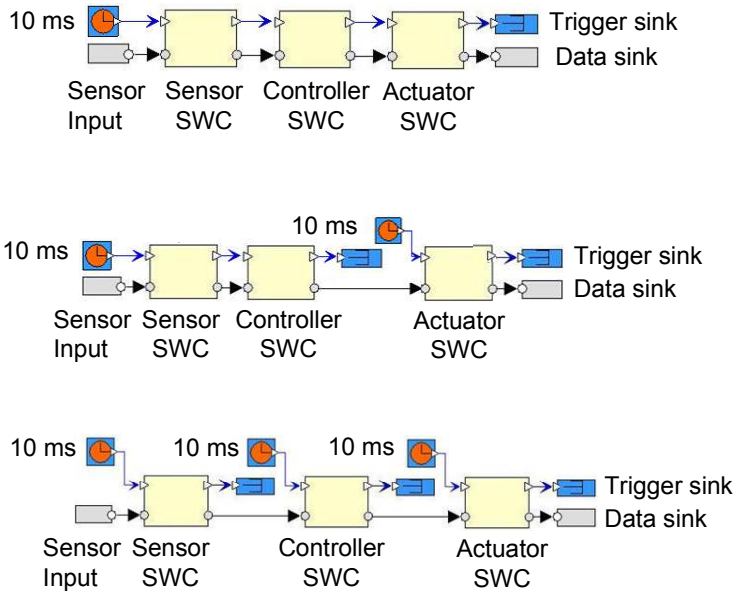
The DL2RCM transformation has been implemented by means of a bidirectional model transformation language, namely Janus Transformation Language (JTL) [14]. JTL is a constraint-based bidirectional model transformation language specifically tailored to support non-bijectivity by generating all the possible models, at once. JTL adopts a QVTr-like syntax [15], supports object pattern matching, and implicitly creates traces to record what occurs during the execution of a model transformation. The JTL implementation relies on the Answer Set Programming (ASP) [16], which is a declarative programming language based on the *answer set* (model) semantics of logic programming. The ASP solver, by means of a deductive process, finds and generates in a single execution all the possible models which are consistent with the transformation rules.

The DL2RCM transformation consists of 28 rules mapping design elements to corresponding RCM implementation elements. The contribution brought by the DL2RCM transformation is two-fold. On the one hand, it allows the automatic translation of EAST-ADL design models to RCM implementation models. On the other hand, it is able to generate the set of all meaningful RCM implementation models for a given design model. That is to say, given the EAST-ADL design model depicted in Figure 2.2a and considering the generation of clocks in the RCM implementation models, the DL2RCM transformation produces 8 RCM implementation models each of which has a unique combination of clocks. 3 of the 8 possible models are depicted in Figure 2.2b.

It is worth to note that JTL supports the specification of logic constraints which can be used for reducing the number of generated models and tailoring their generation for specific purposes. In the specific case of the DL2RCM transformation, we employed logic constraints for forcing the injectivity on the design elements not affected from the specified timing constraints. Paper A (Section 4) discusses an initial version of the transformation which produces one single RCM model from a design model. In Paper D (Section 7) we provide an enhanced version which is able to generate all the meaningful RCM



(a) Example of a design model



(b) 3 of the 8 RCM implementation models for the model in Figure 2.2a

Figure 2.2

models for the leveraged timing analysis.

RCO 3 - Filtering mechanism. This contribution, marked as 3 in Figure 2.1, provides a conceptual mechanism supporting the selection of the best RCM implementation models for a specific, non-empty, set of timing requirements. This represents the last step in the process of leveraging timing analysis at design level for driving design decisions. The proposed filtering mechanism consists of two cascaded filters, i.e., the *elimination filter* and the *selection filter*, as shown in Figure 2.1. After timing analysis is run on the generated RCM implementation models, the RCM implementation models and their analysis results are provided as input to the elimination filter together with the non-empty set of timing requirements specified on the vehicle functionality. The elimination filter discards all the RCM implementation models whose analysis results violate the set of timing requirements. The remaining RCM implementation models, along with their analysis results, are provided as input to the selection filter. This filter further refines the selection by considering the type of component chains required from the vehicle software functionality, i.e., *single-rate chain* or *multi-rate chain* [2], also received as an input. If the selection mechanism fails in identifying a suitable RCM implementation model, architectural refinements at the design model may be needed and the developer will be notified with a message in the console. Similarly, the developer will be notified if one (or a set of) RCM implementation model is selected. Paper B (Section 5) and Paper C (Section 6) present and discuss an initial version of the selection mechanism consisting of the elimination filter. Paper D (Section 7) describes the enhanced version of the selection mechanism consisting of the two cascade filters. Table 2.1 summarizes the relations between RCOs and RCs: RC 1 and RC 2 are addressed by the RCO 1 and RCO2, while RCO 3 addresses RC 3.

		Research Challenges		
		RC 1	RC 2	RC 3
Research Contributions	RCO 1	X	X	
	RCO 2		X	
	RCO 3			X

Table 2.1: Research Contributions in relation to the Research Challenges

[2] In the body electronics domain, the applications are modeled with single-rate chains whereas in the control systems domain the applications are modeled with multi-rate chains.

2.4 Papers Contribution

Table 2.2 shows the relation between the included papers and the RCOs.

		Research Contributions		
		RCO 1	RCO 2	RCO 3
Included papers	A	X	X	
	B		X	X
	C		X	X
	D		X	X

Table 2.2: Included papers in relation to the Research Contributions

2.4.1 Paper A

A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation from EAST-ADL, *Alessio Bucaioni, Saad Mubeen, Federico Ciccozzi, Antonio Cicchetti, Mikael Sjödin.*

Abstract **–**According to the model-driven engineering paradigm, one of the entry requirements when realizing a seamless tool chain for the development of software is the definition of metamodels, to regulate the specification of models, and model transformations, for automating manipulations of models. In this context, we present the metamodel for an industrial component model, the Rubus component model, which is used by several international companies for the development of vehicular embedded systems. The metamodel includes the definition of structural elements as well as concepts for describing timing information. In order to show how, using model-driven engineering, the integration between models can be automatized, we present a model-to-model transformation between models conforming to the automotive domain-specific architecture description language EAST-ADL and models described with the Rubus component model. We also conduct an automotive-application case study to show the applicability of the Rubus component model metamodel and the model transformation.

Status. Conditionally accepted at the Journal of Systems and Software (JSS).

Personal Contribution. The research work presented in this paper was done in collaboration with all the authors. I am the main contributor and driver.

2.4.2 Paper B

Exploring Timing Model Extractions at EAST-ADL Design-level Using Model Transformations, *Alessio Bucaioni, Saad Mubeen, Antonio Cicchetti, Mikael Sjödin*, Proceedings of the 12th International Conference Information Technology: New Generations (ITNG), IEEE, Las Vegas, Nevada (USA), April, 2015.

Abstract – We discuss the problem of extracting control and data flows from vehicular distributed embedded systems at higher abstraction levels during their development. Unambiguous extraction of control and data flows is vital part of the end-to-end timing model which is used as input by the end-to-end timing analysis engines. The goal is to support end-to-end timing analysis at higher abstraction levels. In order to address the problem, we propose a two-phase methodology that exploits the principles of model driven engineering and component based software engineering. Using this methodology, the software architecture at a higher level is automatically transformed to all legal implementation-level models. The end-to-end timing analysis is performed on each generated implementation-level model and the analysis results are fed back to the design-level model. This activity supports design space exploration, model refinement and/or remodeling at higher abstraction levels for tuning the timing behavior of the system.

Status. Published.

Personal Contribution. The research work presented in this paper was done in collaboration with all the authors. I am the main contributor and driver.

2.4.3 Paper C

Raising Abstraction in Timing Analysis for Vehicular Embedded Systems through Model-Driven Engineering, *Alessio Bucaioni*, Proceedings of the Doctoral Symposium at Software Technologies: Applications and Foundations (STAF), L'Aquila, Italy, July, 2015. *Best paper award.*

Abstract – The complexity of vehicular embedded systems is continuously increasing and this can negatively affect their development cost and time to market. One way to alleviate these issues is to anticipate analysis of system properties at design time for early architectural refinements. In this paper, we

present a licentiate work which aims at contributing to this effort. In particular, considering the importance of timing constraints typical of vehicular embedded systems, we leverage model-driven engineering for realizing an automatic approach which allows the developer to perform timing analysis on design models, without having to manually specify timing elements. The proposed approach, starting from a high-level model of the vehicular embedded application, generates a set of candidate models enriched with timing elements in a semi-automatic manner. Timing analysis is run on the generated models and, based on its results, the approach supports the selection of the best candidate model for a specific, non-empty, set of timing constraints.

Status. Published.

Personal Contribution. I am the main contributor and driver.

2.4.4 Paper D

Anticipating Implementation-Level Timing Analysis for Driving Design-Level Decisions in EAST-ADL, *Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, Mikael Sjödin*, Proceedings of the 1st International Workshop on Modelling in Automotive Software Engineering (MASE) at ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (Models), Ottawa, Canada, September, 2015.

Abstract – The adoption of model-driven engineering in the automotive domain resulted in the standardization of a layered architectural description language, namely EAST-ADL, which provides means for enforcing abstraction and separation of concerns, but no support for automation among its abstraction levels. This support is particularly helpful when manual transitions among levels are tedious and error-prone. This is the case of design and implementation levels. Certain fundamental analyses (e.g., timing), which have a significant impact on design decisions, give precise results only if performed on implementation-level models, which are currently created manually by the developer. Dealing with complex systems, this task becomes soon overwhelming leading to the creation of a subset of models based on the developer?s experience; relevant implementation-level models may therefore be missed. In this work, we describe means for automation between EAST-ADL design and implementation levels to anticipate end-to-end delay analysis at design level for driving design decisions.

Status. Published.

Personal Contribution. The research work presented in this paper was done in collaboration with all the authors. I am the main contributor and driver.

2.5 Research Methodology

Software engineering research is often stimulated by problems arising from the development and usage of software in the real-life [17]. Collaborative research between industry and academia is a great example of this phenomenon. The

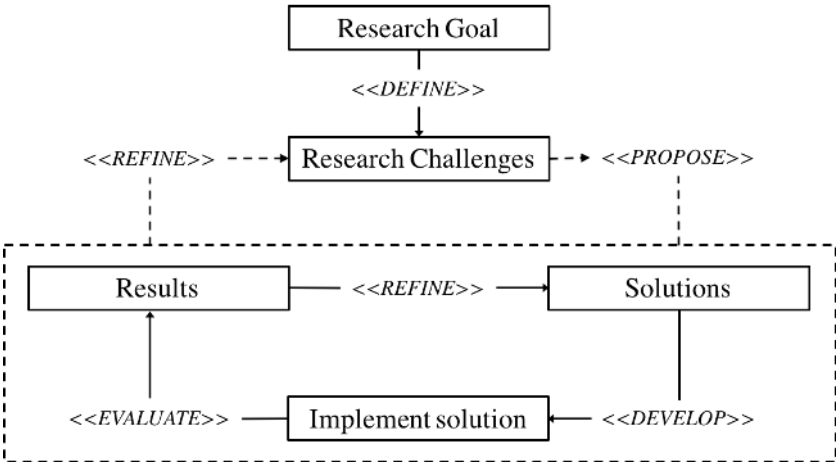


Figure 2.3: Research Methodology

research methodology applied to this licentiate thesis follows the definition of the *engineering method* given in [18]. That is, *"observe existing solutions, propose better solutions, build/develop, measure and analyze, and repeat the process until no more improvements appear possible"*.

Figure 2.3 shows a refinement of this engineering method, adopted for this licentiate thesis work. The industrial demands coming from our industrial partners were used for driving an initial investigation of the related state-of-the-art and the state-of-the-practice. The underlying goal of such an investigation was the definition of a clear research goal. Eventually, starting from the elicited re-

search goal, we defined a set of set of research challenges. For each of them, we conducted an investigation of the related state-of-the-art and the state-of-the-practice with the aim of proposing a possible solution if none existed. After a solution was developed, we evaluated its applicability upon industrial case studies (or case studies mimicking industrial scenarios). Based on the evaluation results, the proposed solution could be refined. This was the case of the model transformation described in RCO 2. In fact, the initial model transformation presented in Paper B (Section 5) has been enhanced in Paper C (Section 7) and Paper E (Section 7), considering the collected results. Eventually, the findings acquired during the elicitation, implementation and evaluation of the solutions, were used for refining the set of research challenges. An example of this iterative process is the selection mechanism described in RC 3. Such a challenge arose only when the enhanced version of the model transformation was developed. In fact, the need of a selection mechanism would be negligible if the model transformation would produce only one single RCM model per design model.

Chapter 3

Conclusions and Future Works

The complexity of software running on embedded systems is constantly increasing, negatively affecting its development costs and time to market. Among the many software development methodologies advocating i) abstraction, ii) separation of concerns, and iii) automation, when dealing with the software development, MDE has progressively gained academic and industrial recognition. Within the automotive domain, the adoption of MDE resulted in the standardization of the layered architecture description language EAST-ADL. While EAST-ADL provides means for abstraction and separation of concerns it does not support automation for transformation between its abstraction levels.

With this licentiate thesis, we introduced an approach which provides automation means for seamlessly linking EAST-ADL design and implementation levels. The underlying goal is to leverage timing analysis at design level such that analysis results can drive design decisions. The overall contribution of the thesis can be broken down into three main research contributions: i) a metamodel definition for the Rubus Component Model (RCM) which is one posisble implementation model for EAST-ADL, ii) automated model transformation from design model to RCM implementation models (called DL2RCM) and iii) a mechanism for the selection of the RCM implementation model which better meets a specific set of timing requirements. The formalization of the RCM metamodel serves as base for leveraging the proposed approach. The DL2RCM transformation provides automation means for seamlessly linking

design and implementation levels, avoiding error-prone and time-consuming activities. By doing so, it allows to leverage timing analysis at design level avoiding the problem of manually identifying a suitable RCM implementation models, in terms of timing characteristics. Eventually, the selection mechanism serves as last step for automatically identifying the best RCM implementation model (or set of models) and propagating back the corresponding analysis results.

Leveraging timing analysis at design level can bring multiple advantages. First of all, it helps the designer in taking design decisions based on much more precise feedback than common design level analysis, which often provides just estimations based on guessed values. We believe that, through our approach it is possible to disclose the opportunity of employing expensive software development resources (e.g., developers, timing experts) more efficiently, therefore to reduce software development costs and time to market. In fact, the proposed approach allows the developer to focus only on design activities exploiting timing analysis without having to investigate nor manually edit implementation models.

The DL2RCM may suffer from scalability issues when dealing with complex design models. Therefore, as a possible continuation of this research work towards a doctoral thesis, we plan to improve the scalability of the generation process. In particular, we will investigate the definition of smarter transformation rules which could avoid the generation of less relevant RCM implementation models by considering preliminary/run time knowledge. Also, we are planning to improve the Janus Transformation Language such that it could consider heuristics for the generation of classes of models. Our experience has shown that, besides timing, other relevant system properties need to be dealt with at design level. Therefore, another interesting research direction could be to investigate the possibility to leverage our approach to combine the optimization of multiple system properties, e.g., memory demands. Eventually, together with our industrial partners we will investigate challenges as well as possible benefits and drawbacks of extending the proposed approach to higher abstraction levels in EAST-ADL. We would also like to conduct empirical evaluations of the proposed technique in a real industrial context.

Bibliography

[1] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: the state of the practice. *Software, IEEE*, 20(6):61–69, 2003.

[2] Stuart Kent. Model driven engineering. In *Integrated formal methods*, pages 286–298. Springer, 2002.

[3] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, 2001.

[4] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *Software, IEEE*, 20(5):42–45, 2003.

[5] EAST-ADL Domain Model Specification, Deliverable D4.1.1, 2010. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[6] George T. Heineman and William T Councill. Component-based software engineering: putting the pieces together. *Component-based software engineering: putting the pieces together*, pages 33–48, 2001.

[7] Kaj Hänninen, Jukka Mäki-Turja, Mikael Sjödin, Mats Lindberg, John Lundbäck, and Kurt-Lennart Lundbäck. The rubus component model for resource constrained real-time systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[8] Bran Selic and Leo Motus. Using models in real-time software design. *Control Systems, IEEE*, 23(3):31–42, June 2003.

[9] Philippe Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, Mark-Oliver Reiser, David Servat, R Tavakoli Koligari, and DeJiu Chen. Developing automotive products using the east-adl2, an autosar compliant architecture description language. In *Embedded Real-Time Software Conference*. Citeseer, 2008.

[10] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *Proceedings of the IEEE Real-Time System Symposium ? Workshop on Compositional Theory and Technology for Real-Time Embedded Systems,*, 2008.

[11] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2):117–134, 1994.

[12] Ian Sommerville, Wendy Boggs, Michael Boggs, Bernd Bruegge, Allen H Dutoit, Wendy Boggs, and Michael Boggs. Software engineering 7 th ed. 2002.

[13] Alessio Bucaioni and Romina Eramo. Understanding bidirectional transformations with tggs and jtl. In *Second International Workshop on Bidirectional Transformations*, March 2013.

[14] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Jtl: A bidirectional and change propagating transformation language. In *Software Language Engineering*, volume 6563, pages 183–202. 2011.

[15] Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT). OMG Group.

[16] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. volume 88, pages 1070–1080. MIT Press, 1988.

[17] Mary Shaw. The coming-of-age of software architecture research. In *Proceedings of the 23rd International Conference on Software Engineering*, page 656. IEEE Computer Society, 2001.

[18] Victor R. Basili. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 1–12. Springer, 1993.

II

Included Papers

Chapter 4

Paper A:
A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation from EAST-ADL

Alessio Bucaioni, Saad Mubeen, Federico Ciccozzi, Antonio Cicchetti and Mikael Sjödin

Conditionally accepted at the Journal of Systems and Software (JSS)

Abstract

According to the Model-Driven Engineering paradigm, one of the entry requirements when realizing a seamless tool chain for the development of software is the definition of metamodels, to regulate the specification of models, and model transformations, for automating manipulations of models. In this context, we present the metamodel for an industrial component model, the Rubus Component Model, which is used by several international companies for the development of vehicular embedded systems. The metamodel includes the definition of structural elements as well as elements for describing timing information. In order to show how, using Model-Driven Engineering, the integration between models can be automate, we present a model-to-model transformation between models conforming to the automotive domain-specific architecture description language EAST-ADL and models described with the Rubus Component Model. We also conduct an automotive-application case study to show the applicability of the Rubus Component Model metamodel and the model transformation.

4.1 Introduction

During the last decades, industrial demands on vehicular embedded systems have been constantly evolving causing an increment of the related software complexity. It has been estimated that current vehicles can have more than 70 embedded systems running up to 100 million lines of code [1]. On the one hand, industry needs efficient processes to cope with the size of these systems for optimizing software development cost and time-to-market. On the other hand, most of vehicular embedded systems have extra-functional properties, e.g., timing requirements and constraints, which have to be taken into account from the early stages of the development. In fact vehicular embedded systems are real-time systems [2], meaning that they must deliver their functionality within their timing deadlines. Consequently, timing requirements are crucial for these systems. In this context, traditional software development processes have shown strong limitations.

Component Based Software Engineering (CBSE) has been acknowledged as an effective practice to deal with the increasing complexity of modern embedded software [3] by promoting software development at a higher level of abstraction relying on the definition and reuse of atomic units of composition, i.e., software components. Additionally, CBSE allows to express timing properties, e.g., by annotating the software components with properties and constraints (e.g., worst-case execution time) thus enabling timing analysis, e.g., end-to-end response time and delay analysis [4].

AUTOSAR [5] and the Rubus Component Model (RCM) [6], to name a few, are examples of component models (CMs) used within the vehicular domain. Lately, AUTOSAR has become part of the EAST-ADL initiative [7]. EAST-ADL is an architecture description language (ADL) which provides concepts and methods for managing and organizing the various artifacts produced along the software development of vehicular embedded systems [8]. It promotes the separation of concerns through a top-down software development process relying on four different abstraction levels, i.e., vehicle, analysis, design and implementation level. In the latter level, EAST-ADL makes use of AUTOSAR. Both EAST-ADL and AUTOSAR, embracing the Model-Driven Engineering (MDE), have been provided with metamodel definitions. MDE is a paradigm that intends software development as the process of designing and refining models, starting from higher and moving towards lower levels of abstraction, via the so-called model transformations.

While EAST-ADL has been proven successful in coping with the software complexity and size of industrial embedded software, it still provides

limited support for dealing with timing requirements. In fact, by employing AUTOSAR at implementation level, most of the timing, implementation and communication details are neglected. This information is necessary for building software timing models used for verifying timing requirements.

In this context, an increasing number of vehicular manufacturers are using RCM as a complementary technology in EAST-ADL based processes. In this case, in order to allow a smooth interplay between different languages in these processes, proper automation is needed for the translation among the various artifacts specified using, e.g., RCM and EAST-ADL. This aspect is even more crucial when considering that, in practice, manual translations are not only tedious, time consuming and error-prone, but even unfeasible in most of the cases due to the size and complexity of the models involved. To this end, MDE has been proven effective in reducing the software development cost and time to market [9] while automating the whole development process. In order to embrace the MDE paradigm and benefit from its advantages, it is necessary to define proper metamodels for any modeling language (e.g., component models) involved in the development process together with proper model transformations devoted manipulations of models, taking into account the need of modeling both functional and extra-functional concepts.

In this paper, we define a metamodel for RCM, that is used for the software development of vehicular real-time embedded systems. The metamodel is defining according to the following main goals:

backward compatibility: the metamodel should allow an easy migration of legacy RCM artifacts into the new modeling environment;

maintainability: the metamodel should enable a better management of RCM updates and refinements;

extensibility: the metamodel should disclose the opportunity to integrate in a smooth way RCM modeling environment in a typical automotive application development chain.

To this end, we first present the definition of metamodeling elements representing the software architecture. Then, we extend the metamodel with concepts representing timing constraints and properties for different type of delays in event chains. Instead of discussing the complete timing package in RCM, we focus on the elements representing the latest timing constraints (and corresponding timing information and analyses) introduced and practically used within the automotive industrial domain, i.e., the age and reaction delays [10, 7, 11, 4]. Moreover, we show how the integration between EAST-ADL and

RCM can be automated using the MDE paradigm, by presenting a model-to-model transformation from the EAST-ADL Design level to RCM (DL2RCM) together with a case study which mimics a typical industrial scenario.

The rest of the article is organized as follows. Section 4.2 presents the context of this work together with its related works. Section 4.3 introduces the RCM metamodel and extensions for timing elements. Section 4.4 shows the DL2RCM transformation while Section 4.5 discusses its applicability on a case study. Finally, Section 4.6 highlights the benefits of having a proper metamodel for RCM while Section 4.7 draws conclusions and discusses future works.

4.2 Background and related work

4.2.1 MDE and CBSE in the Automotive Domain

MDE is a paradigm which aims at raising the level of abstraction of software development by focusing on modeling activities rather than coding. In this context, MDE promotes *models* and *model transformations* as first-class citizens.

Models represents an abstraction of the system under development, from a particular point of view [12]. The set of rules and constraints needed for building a valid model is specified in the so-called *metamodel*. Formally, a metamodel defines the abstract syntax of a given model, or set of models; the relation between metamodel and models is called conformance. Model transformations represent the means of refinement by which models are manipulated [13]. In fact, model transformations translate a source model into a target model keeping their conformance to the respective metamodel intact.

According to the MDE paradigm, starting from a model and by means of model transformations it is possible to automatically obtain a variety of artifacts, such as new models, code, etc. In this context, the entire software development can be seen as a transformation process where low level abstraction models are automatically obtained by means of model transformations from higher-level abstraction models.

Within the automotive domain, the adoption of MDE and CBSE paradigms led to the standardization of an architectural description language, namely EAST-ADL [7]. EAST-ADL proposes a view over the development process composed by four different abstraction levels. Figure 4.1 shows the abstraction levels together with methodologies and languages used at each one of them.

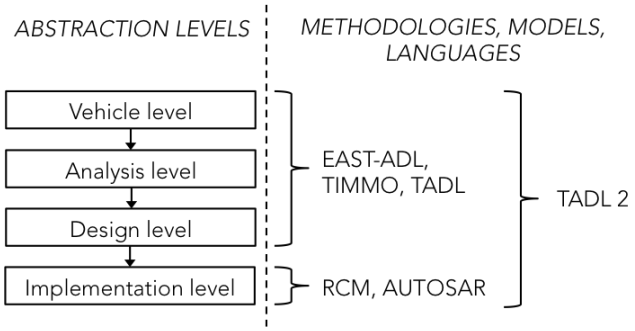


Figure 4.1: EAST-ADL abstraction levels

The *vehicle level* is the highest abstraction level and captures information regarding the system's functionality. In this level, feature models can be used for showing what the system provides in terms of functionality. Also, these models are decorated with requirements. The vehicle level is also known as end-to-end level as it serves to capture requirements and features on the end-to-end vehicle functionality.

At the *analysis level*, vehicle functions are expressed, using formal notations, in terms of behaviors and interfaces. Yet, design and implementation details are omitted. The artifact developed at this level is called Functional Analysis Architecture. At this stage, high level analysis for functional verification can be performed.

At the *design level*, the analysis-level artifacts are refined with design-oriented details: while the analysis level does not differentiate among software, middleware and hardware, the design level explicitly separates them. Allocation of software functions to hardware nodes is expressed at this level too. The artifacts developed at this level include Functional Design Architecture and Hardware Design Architecture.

At the *implementation level*, artifacts introduced at the design level are refined with implementation details. The output of this level is a complete software architecture which can be used for code generation. At this stage component models, e.g., RCM, AUTOSAR, are used to model the system in terms of components and interactions among them.

AUTOSAR is an industrial initiative to provide standardized software architecture for the software development of vehicular embedded systems. Within AUTOSAR, the software architecture is defined in terms of *software compo-*

nents (SWCs) and *Virtual Function Bus* (VFB). VFB handles the virtual integration and communication among SWCs, hiding low-level implementation details. AUTOSAR describes the software at a high level of abstraction focusing on the functional and structural aspects of the architecture. Also, it does not distinguish between data and control flow, as well as between inter- and intra-node communication.

Lately, AUTOSAR has been provided with a timing model within the two EU research projects TIMMO [14] and TIMMO-2-USE [15], respectively. To this end, TIMMO provides a predictable methodology and language, called TADL [16] for expressing timing requirements and constraints. TADL is inspired by MARTE [17], which is a UML profile for modeling and analysis of real-time and embedded systems. The TIMMO methodology makes use of EAST-ADL and AUTOSAR interplay, where the former is used for the software structural modeling, while the latter is used for the implementation. TIMMO-2-USE [15], a follow up project, presents a major redefinition of TADL in TADL2 [10]. The purpose of this project is to include new functionality for supporting the AUTOSAR extensions regarding timing model. Although both TIMMO and TIMMO-2-USE attempt to annotate AUTOSAR with a timing model, AUTOSAR is still not expressive enough for representing timing, implementation and communication information of the software architecture.

In this context, an increasing number of vehicular manufacturers are considering RCM as an alternative to AUTOSAR within the EAST-ADL based methodology. RCM supports both model- and component-based development. The main goal of RCM is to express the software architecture in terms of software functions and interactions among them. In RCM, the basic entity is the so-called *software circuit* (SWC) which represents the lowest-level hierarchical element in RCM and encapsulates basic software functions. Each SWC is defined by its *behavior* and *interface*. Interfaces manage the interactions among SWCs via ports. RCM distinguishes between data and control flow. Therefore, the interfaces have two types of ports: *data ports* for the data flow and *trigger ports* for the control flow. The SWC is characterized by run-to-completion semantics meaning that, upon triggering, it reads data from the data input ports, executes its behavior and writes data on the data output ports.

SWCs can be grouped and organized in *assemblies*, decomposing the system in a hierarchical manner. *Modes* are used to represent different configurations of the software architecture. *Target* entities are used for grouping modes that are deployed on the same Electronic Control Unit (ECU). Moreover, they provide details regarding hardware and operating system. *Node* is a hardware

and operating-system independent abstraction of a target entity. Finally, *System* is the top-level hierarchical entity, which describes software architecture for the complete vehicular system.

RCM facilitates analysis and reuse of components in different contexts by separating functional code from the infrastructure that implements the execution environment. Compared to AUTOSAR, RCM allows the developer to specify and handle timing information at design time. It also distinguishes between data and control flow as well as inter- and intra-node communication. To this end, RCM has been recently extended with special network interface components for modeling the inter-node communication [18]. The RCM pipe-and-filter communication mechanism is very similar to the AUTOSAR sender-receiver communication mechanism. In short, RCM was specifically designed for expressing all the low-level information needed for extracting the timing model from the software architecture.

4.2.2 End-to-end timing models and analyses

End-to-end timing analysis is a key activity for the verification and validation of vehicular real-time systems. Therefore, a tool chain that is used for the model- and component-based development of vehicular systems shall support such an analysis. In turn, to support timing analysis an appropriate system view, called end-to-end timing model, should be extracted from the software architecture. In particular, an end-to-end timing model comprises of timing properties, requirements, dependencies and linking information concerning all tasks, messages and task chains in a distributed embedded system under analysis[1]. It is mainly composed of two models namely a timing model and a linking model. In order to elaborate this, consider a task chain distributed over three nodes connected by a network as shown in Figure 4.2. The system timing model captures all the timing information about the three nodes and the network. Whereas the linking model includes all the linking information in the task chains, including the control and data flows.

The analysis engines use these models for performing end-to-end timing analyses. We refer the reader to [4] for further details about the end-to-end timing analyses. The analysis results consist of response time of tasks and messages as well as system utilization. Also, the analysis calculates end-to-end response times and delays. The end-to-end response time of a task chain is equal to the elapsed time between the arrival of a stimulus, e.g., the brake pedal

[1] We refer the reader to [18] for details about the timing model.

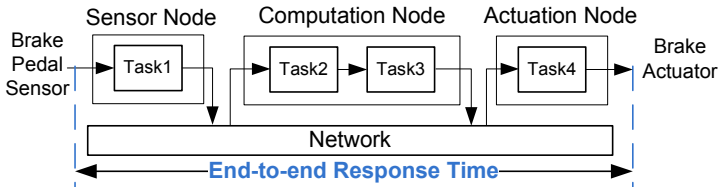


Figure 4.2: Example showing end-to-end response time

sensor input in the sensor node, and the response to it, e.g., the brake actuation signal in the actuation node as shown in Figure 4.2.

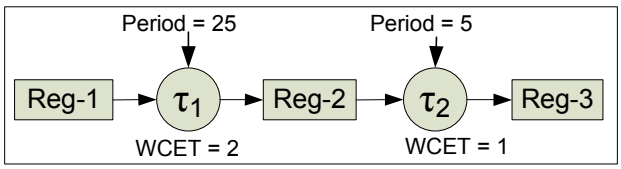


Figure 4.3: A task chain with independent activation of tasks

Within a task chain, if the tasks are triggered by independent sources, then it is important to calculate different types of delays such as age and reaction. An *age delay* corresponds to the freshness of data. It finds its importance in control systems used in the vehicles. Whereas, the *reaction delay* corresponds to the first reaction for a given stimulus. This delay finds its application in body electronics in the vehicles.

In order to explain these delays, consider a task chain in a single-node system as shown in Figure 4.3. There are two tasks in the chain denoted by τ_1 and τ_2. The tasks are triggered by independent clocks of periods 25ms and 5ms respectively. Let the Worst-Case Execution Times (WCETs) of these tasks be 2ms and 1ms respectively. τ_1 reads data from the register Reg-1 and writes data to Reg-2. Similarly, τ_2 reads data from the Reg-2 and writes data to Reg-3. Since, the tasks are activated independently with different clocks, there can be multiple outputs (Reg-3) corresponding to one input (Reg-1) to the chain as shown by several uni-directional arrows in Figure 4.4. The age and reaction delays are also identified in Figure 4.4. These delays are equally important in distributed embedded systems.

We consider the end-to-end timing model that corresponds to the holistic schedulability analysis for distributed embedded systems [19]. Stappert et

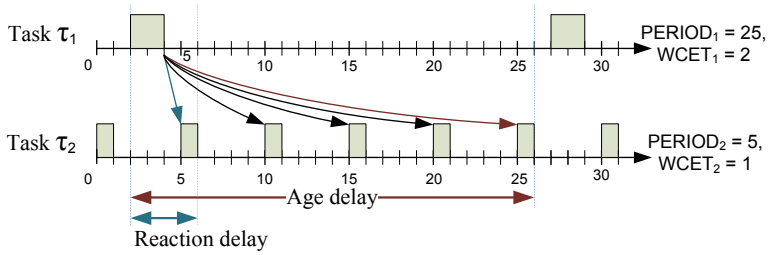


Figure 4.4: Example showing end-to-end delays

al. [20] described end-to-end timing constrains for multi-rate automotive embedded systems. In [11], Feiertag et al. presented a framework (developed as part of the TIMMO project) for the calculations of end-to-end delays. A scalable technique, based on model checking, for the computation of end-to-end latencies is described in [21].

4.2.3 Paper contributions

Compared with RCM, AUTOSAR describes the software architecture at higher level of abstraction; it focuses on the functional and structural aspects of the software architecture hiding low-level timing information, such as control flow. Neglect timing, implementation and communication information hampers end-to-end timing analysis. In [22] we propose RCM as an alternative to AUTOSAR in an EAST-ADL based methodology and we discuss its use for enabling end-to-end timing analysis. Moreover, in [4], we provide a method for extracting timing models and perform end-to-end timing analysis of component-based vehicular embedded systems.

This paper extends our previous work [23] where we present the metamodel definition of the architectural elements in RCM and discusses the transformation process from RCM to AUTOSAR [5]. However, the definition of metamodeling elements for representing timing properties and constrains is missing. In this paper we complement our previous work by including metamodeling elements representing timing properties and constraints for different types of delays that can be specified in single-node as well as distributed embedded systems. We provide a model-to-model transformation from the design-level models described using EAST-ADL to RCM model exploiting the extended metamodel. In addition, we conduct an automotive application case study to show the applicability of the extended metamodel as well as the transforma-

tion process. Eventually, we validate the metamodel expressiveness by means of several real-life automotive use-cases.

4.3 Providing a metamodel for RCM

In this section, we describe the RCM metamodel. For the sake of readability, we divide and present the metamodel in four fragments. However, the four fragments can be combined for an holistic view of the metamodel by matching metaclasses with the same names.

Figure 4.5 shows the metamodel's backbone. The top metaclass is *System*, which acts as a container for the whole architecture. System, as all the metaclasses in the metamodel, inherits from the abstract metaclass *NamedElement*. A System element contains one or more elements of type *Node*. A Node element is a hardware and operating-system independent abstraction of a *Target* element; it groups all the software architecture elements which realize a specific function. Its reference *activeTarget* defines which target, among those specified, is active for a certain node. Target is a hardware and operating-system specific instance of a Node; it serves for modeling the deployment of the software architecture. This means that, it contains all the functions to be deployed on the same ECU. Consequently, a Node can be realized by different Target elements, depending on the hardware and the operating system used for the deployment, for example, PowerPC with Rubus Operating System, Simulated target with Windows operating system.

A Target element contains one or more elements of type *Mode*. A Mode represents a specific application of the software architecture as, for instance, start-up or low-power mode. A Mode element might contain elements of type *Circuit* and *Assembly*. A Circuit is the lowest-level hierarchical element which encapsulates basic functions. It contains an element of type *Interface* and one or more elements of type *Behavior*. An Interface groups all the data and control ports of a certain circuit while a Behavior contains the code to be executed from the specific Circuit. The reference *activeBehavior* is used for specifying which behavior is active for the related circuit. Assembly elements do not add any semantics to the architecture: they are used for grouping and organizing circuits and assemblies in a hierarchical fashion. Both the metaclasses *ConnectorData* and *ConnectorTrig* inherit from the abstract metaclass *Connector*. A Connector realizes the actual communication between two ports. ConnectorData and ConnectorTrig metaclasses are used for modeling the communication between data ports and control ports, respectively.

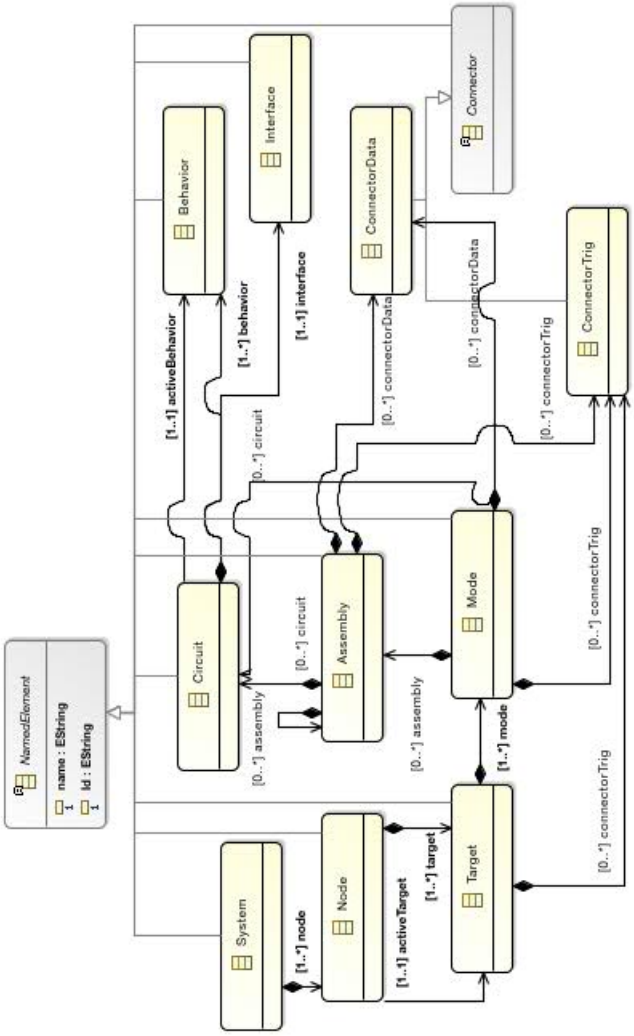


Figure 4.5: Fragment of the RCM metamodel representing the backbone elements

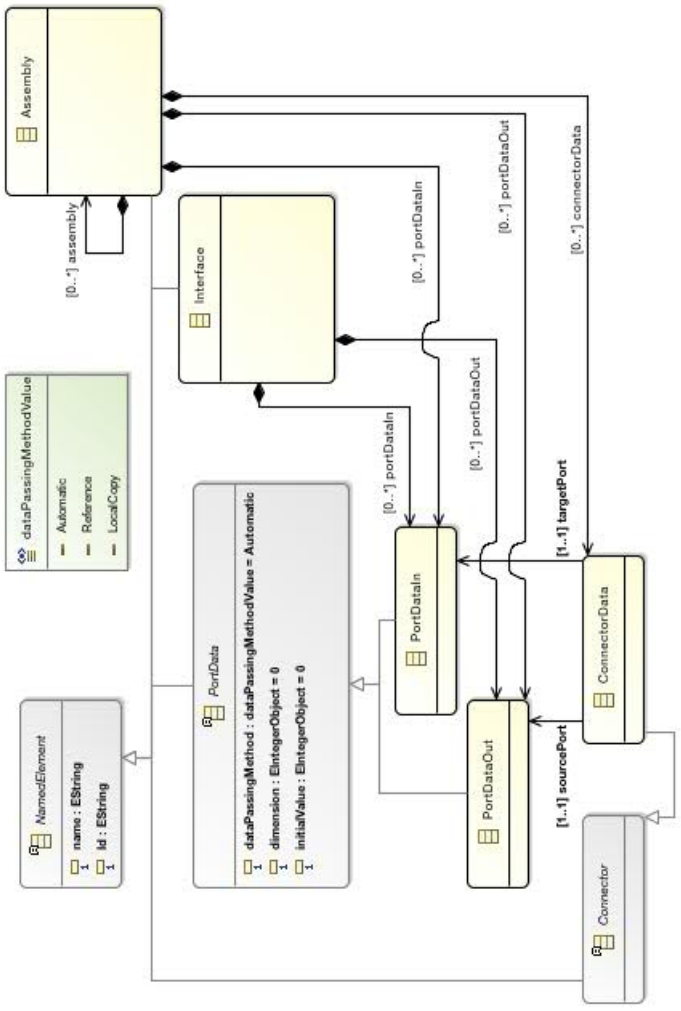


Figure 4.6: Fragment of the RCM metamodel representing the data flow elements

RCM explicitly separates data and control flow. Figure 4.6 shows the metamodel fragment containing the concepts used for modeling the data flow. The

abstract metaclass *PortData* models a generic data port. It has three attributes: *dataPassingMethod* specifies how data is passed to the port, *dimension* expresses the size of the port while *initialValue* specifies its initial value. The metaclass PortData is specialized by the metaclasses *PortDataIn* and *PortDataOut*, which model an input and output data port, respectively. They are contained in the Interface and the Assembly metaclasses for modeling the data communication among circuits and assemblies, respectively. As aforesaid, the metaclass ConnectorData is used for modeling the actual communication between two data ports. In this respect, the references *sourcePort* and *targetPort* are used for specifying the ports involved in the communication.

Figure 4.8 shows the metamodel fragment containing the concepts that can be used to represent the control flow. The metaclasses *PortTrigIn* and *PortTrigOut* describe an input trigger port and an output trigger port, respectively. They both inherit from the metaclass *PortTrig*, which describes a generic trigger port. Modes, assemblies and interfaces are composed of input and output trigger ports for modeling the control flow among modes, assemblies and circuits, respectively. The *ConnectorTrig* metaclass inherits from the abstract metaclass Connector. It has two references, *sourcePort* and *targetPort*, used for modeling the actual communication between trigger ports. *Clock* and *Sink* elements are responsible to start and end the execution of a software circuit, respectively.

Figure 4.8 depicts an excerpt of the RCM metamodel containing the metalements representing timing constraints and properties for different types of delays in event chains. The notion of different delay types is meaningful in multi-rate systems where components in the event chain can be triggered with independent clocks. Hence, there can be multiple occurrences of response corresponding to a single occurrence of stimulus in the chain. In RCM, these constraints are specified by means of two model elements placed at the beginning and at the end of the event chain.

The metaclasses which represent the data reaction constraint are *DataReactionStart* and *DataReactionEnd*, while the metaclasses which model the data age constraint are the *DataAgeStart* and *DataAgeEnd*. DataAgeStart and DataReactionStart inherit from the abstract metaclass *DataStart*, while DataAgeEnd and DataReactionEnd inherit form the abstract metaclass *DataEnd*. The *deadline* attribute of the DataEnd metaclass specifies the maximum value for the related reaction along the enclosed chain. DataStart and DataEnd inherit from the abstract metaclass *Data*, which models a generic delay constraint. It contains a data input port and a data output port, meaning that the data traveling along the data chain must traverse the delay constraint for activating it.

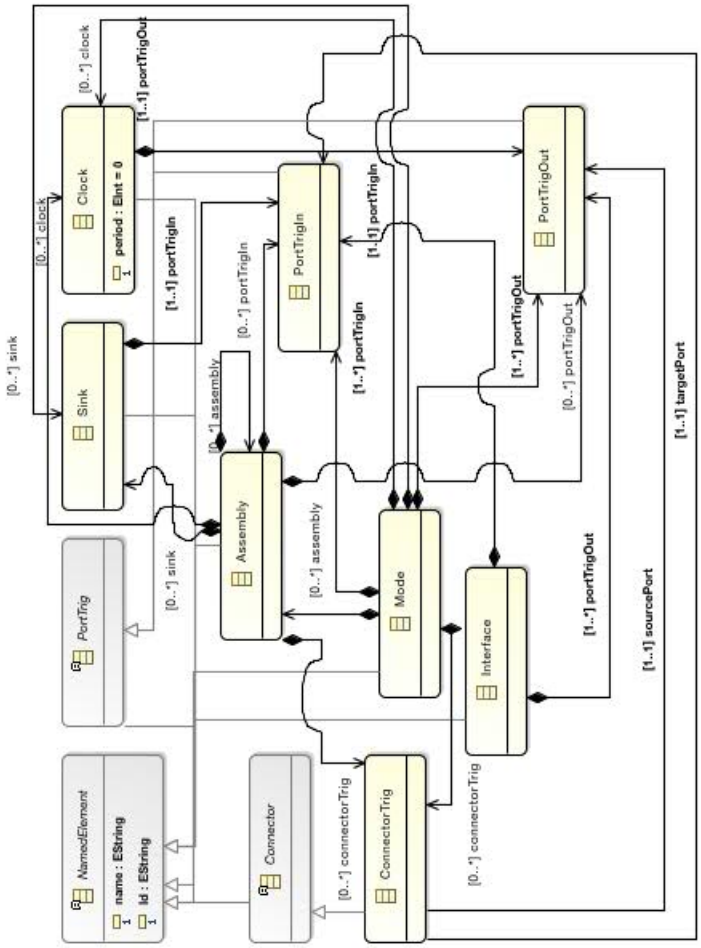


Figure 4.7: Fragment of the RCM metamodel representing the control flow elements

4.4 DL2RCM model transformation

In this section we present DL2RCM, a model-to-model transformation from the EAST-ADL Design Level metamodel to RCM. The intent is to show how,

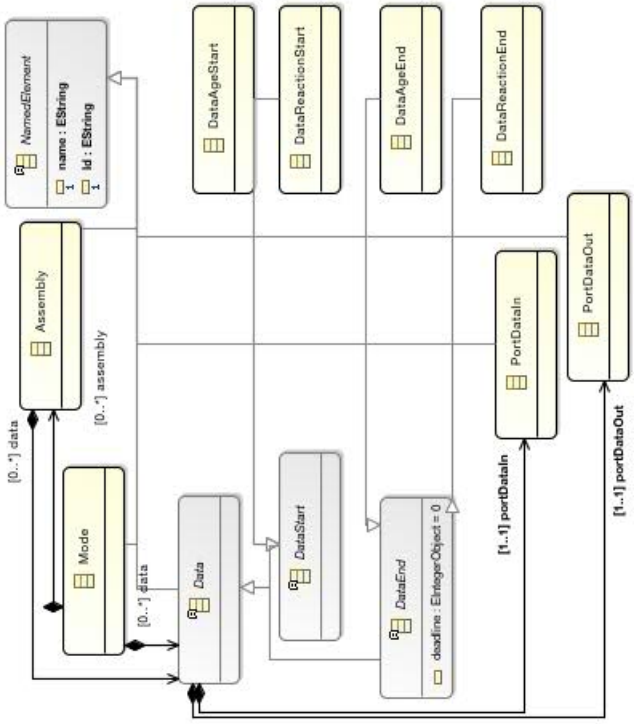


Figure 4.8: Fragment of the RCM metamodel representing the timing constraints and properties for different types of delay in event chains

having a proper metamodel for RCM, it is possible to realize a seamless thus complement EAST-ADL with the RCM's timing analysis capabilities. In Section 4.2, we showed how the EAST-ADL methodology (EAST-ADL complemented by AUTOSAR at the implementation level) uses the four abstraction levels for implementing a top-down development process. In this respect, we presented RCM and AUTOSAR to be technologies used at the last abstraction level, i.e., implementation level. In our previous work we proposed RCM as an alternative to AUTOSAR within an EAST-ADL development methodology. To this end, we believe it is crucial to show that RCM fully integrates within the EAST-ADL methodology. That is, considering the EAST-ADL four abstrac-

tion levels, it is possible to synthesize an EAST-ADL Design Level model to a RCM model.

The DL2RCM transformation is used for performing such an integration automatically. The benefits of realizing this in an automatic manner become more visible when considering that the involved technologies, EAST-ADL and RCM, are used for representing complex architectures, for which manual translations are not only tedious, time consuming and error-prone, but they might even become unfeasible.

The DL2RCM transformation is a unidirectional model-to-model transformation from the EAST-ADL Design Level metamodel to the RCM metamodel. The latter has been presented in Section 4.3. The former has been described in [7] and implemented as a UML profile within the Eclipse Papyrus project [2]. Figure 4.9 shows the extract of the EAST-ADL metamodel containing the concepts entailed by the DL2RCM transformation[3].

The relation underneath the transformation is non-bijective meaning that the involved metamodels do not have the same expressiveness. In this respect, in order to preserve as much information as possible, assumptions are needed when defining the relations composing the transformations. Table 4.1 summarizes the main relations together with their assumptions. The interested reader can find a detailed discussion on the assumptions and the constraints used for defining the DL2RCM transformation in [22].

Algorithm 1 shows the DL2RCM transformation in pseudocode. The MODEL2SYSTEM function is the starting function of the transformation. It is responsible for translating an EAST-ADL Model element into a hierarchy of RCM elements consisting of System, Node, Target and Mode elements (line 2). This step can be skipped when considering all EAST-ADL abstraction levels, since the RCM elements would be translated from the equivalent EAST-ADL elements. In our case, since we are considering just the EAST-ADL design level, this step is needed to build a correct hierarchy in the RCM model, conforming to the RCM metamodel.

One of the major difficulties in defining the DL2RCM transformation is that EAST-ADL implements the *type-prototype* pattern: a *DesignFunctionPrototype* element is considered to be a specific instance of the *DesignFunctionType* element which in turn might contain other prototypes and connectors realizing its inner architecture (see Figure 4.9). This means that the inner architecture of a prototype is defined through its related type. Such a pattern, not lever-

[2] http://eclipse.org/papyrus/

[3] The explanation of theEAST-ADL metamodel is outside the scope of this work. The interested reader may refer to [7]

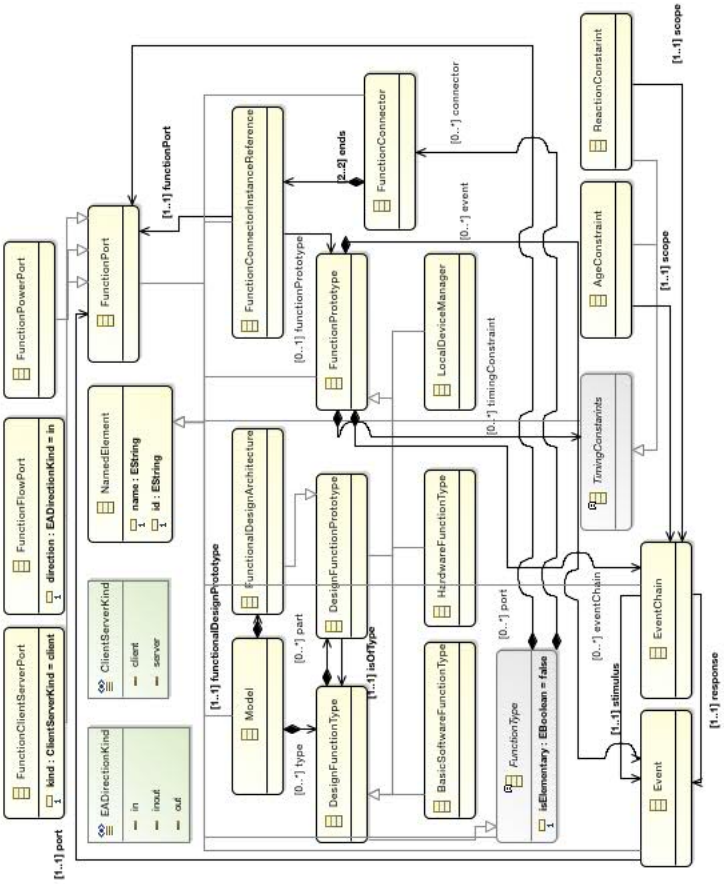


Figure 4.9: Fragment of the EAST-ADL metamodel for Function Modeling at the design level

aged in RCM, required additional effort in designing the transformation, as each DesignFunctionPrototype has to be checked against its type before to be transformed. These negligible low-level details are omitted from the pseudocode in Algorithm 1 for the sake of readability. For the same reason, in the pseudocode we make use of helper functions (e.g., CREATEHIERARCHY,

EAST-ADL elements	RCM elements	Conditions/Assumptions
Model	(hierarchy of) System, Node, Target, Mode	If we consider all the EAST-ADL abstraction levels then the hierarchy is not necessary
DesignFunctionPrototype	Assembly	If the associated DesignFunctionType is not elementary
DesignFunctionPrototype	(hierarchy of) Circuit, Interface	If the associated DesignFunctionType is not elementary
FunctionConnector	ConnectorData	
FunctionFlowPort	PortData	
AgeConstraint	DataAgeStart, DataAgeEnd	
ReactionConstraint	DataReactionStart, DataReactionEnd	
	ConnectorTrig, PortTrig, Clock, Sink	EAST-ADL does not explicitly model the control flow. Therefore these RCM elements are deducted considering the whole architecture

Table 4.1: Main relations holding in the DL2RCM transformation

CREATEASSEMBLY) which are responsible for the creation of the related elements and their inner architecture.

The *FDP* function is responsible for translating an EAST-ADL *DesignFunctionPrototype* element into a RCM *Assembly* or *Circuit* element depending on whether its related *DesignFunctionType* is an elementary element, meaning that it does not contain any other *DesignFunctionProtype* element. In the case it is not an elementary element (line 14), all the contained *DesignFunctionProtype* elements are transformed too. This translation is performed in two steps. First, FDP calls the *C2C* function on all its *FunctionConnector* elements (lines 10-12), for the translation of the elements connected via connectors. Afterwards, the FDP function calls *DP2A* or *DP2C* on its spare *DesignFunctionProtype* elements; if they are elementary elements then they are transformed into circuits by the DP2C function (line 17), otherwise they are transformed into assemblies through the DP2A function (line 15).

The C2C function translates an EAST-ADL *FunctionConnector* element into a RCM *DataConnector* element. More precisely, for each *FunctionConnector* element, the C2C function creates a *DataConnector* element (line 26) together with the connected *Assembly*/*Circuit* elements by calling the functions DP2A (line 29) and DP2C (line 41), respectively. *Port* elements are created and connected accordingly (lines 33, 36, 45, 48). Control flow information is not explicitly modeled at EAST-ADL design level. Therefore, we assume that each SWC is triggered independently. To this end, the C2C function generates the needed *Clock* (lines 32, 44) and *Sink* (lines 35, 47) elements together with the

Algorithm 1 DL2RCM transformation

```
1: function MODEL2SYSTEM(Model m)
2:     Mode mo = CREATEHIERARCHY(m);
3:     FDP(m.functionalDesignPrototype, mo)
4:     TC2TC(fdp, mo)
5: end function
6:
7: function FDP(FunctionalDesignPrototype fdp, Mode mo)
8:     if fdp is not elementary then
9:         Assembly a = CREATEASSEMBLY(fdp, mo);
10:        for connector in fdp do
11:            C2C(connector, a)
12:        end for
13:        for part in fdp do
14:            if part is not elementary then
15:                Assembly as = DP2A(part, a);
16:            else
17:                Circuit ci = DP2C(part, a);
18:            end if
19:        end for
20:    else
21:        Circuit c = CREATECIRCUIT(fdp, mo);
22:    end if
23: end function
24:
25: function C2C(FunctionConnector fc, Assembly a)
26:    ConnectorData con = CREATECONNECTORDATA(fc, a);
27:    for end in fc do
28:        if end.functionPrototype is not elementary then
```

ConnectorTrig elements.

With a logic similar to FDP, functions *DP2C* and *DP2A* translate an EAST-ADL DesignFunctionPrototype into RCM Circuit and RCM Assembly, respectively.

The function *TC2TC* is responsible for translating the timing (age and reaction) constraints. Starting from the outer *DesignFunctionPrototype*, it iterates on all the specified timing constraints (line 78). For each of them, it uses the start and end events (*stimulus* and *response* in Figure 4.9) for searching, within

Algorithm 1 DL2RCM transformation

```
29:             Assembly as = DP2A(end.functionPrototype, a);
30:             if end.functionPort is FunctionFlowPort then
31:                 if e.functionPort is in then
32:                     ConnectorTrig conTC =
33: CREATECONTROLFLOWIN(fc, a);
34:                     PortDataIn pdi =
35: CREATEPORTDATAIN(fc.functionPort, as);
36:                 else
37:                     ConnectorTrig conTS =
38: CREATECONTROLFLOWOUT(fc, a);
39:                     PortDataOut pdo =
40: CREATEPORTDATAOUT(fc.functionPort, as);
41:                 end if
42:             else
43:             end if
44:         else
45:             Circuit c = DP2C(e.functionPrototype, as);
46:             if end.functionPort is FunctionFlowPort then
47:                 if end.functionPort is in then
48:                     ConnectorTrig conTC =
49: CREATECONTROLFLOWIN(fc, a);
50:                     PortDataIn pdi =
51: CREATEPORTDATAIN(fc.functionPort, c);
52:                 else
53:                     ConnectorTrig conTS =
54: CREATECONTROLFLOWOUT(fc, a);
55:                     PortDataOut pdo =
56: CREATEPORTDATAOUT(fc.functionPort, c);
57:                 end if
```

the RCM model, the connector attached to the port and specified by the stimulus or response events (lines 79-82). After *DataAgeStart*, *DataReactionStart*, *DataAgeEnd* and*DataReactionEnd* elements are created (lines 84, 85, 90, 91), they are connected to the related data ports (lines 86, 92).

Algorithm 1 DL2RCM transformation

```
58:             else
59:             end if
60:         end if
61:     end for
62: end function
63: function DP2A(DesignFunctionPrototype dfp, Assembly a)
64:     Assembly as = CREATEASSEMBLY(dfp, a);
65:     for connector in dfp do
66:         C2C(connector, a)
67:     end for
68:     for part in dfp do
69:         if part is not elementary then
70:             Assembly as1 = DP2A(part, as);
71:         else
72:             Circuit c1 = DP2C(part, as);
73:         end if
74:     end for
75:     return as;
76: end function
77:
78: function DP2C(DesignFunctionPrototype dfp, Assembly a)
79:     Circuit c = CREATECIRCUIT(dfp, a);
80:     ConnectorTrig conTC = CREATECONTROLFLOWIN(dfp, c, a);
81:     ConnectorTrig conTS = CREATECONTROLFLOWOUT(dfp, c, a);
82:     return c;
83: end function
```

4.5 Application to the steer-by-wire system

In order to show the applicability of the DL2RCM transformation, we provide a part of the Steer-By-Wire (SBW) system case study. It is a vehicular feature that substitutes most of the mechanical and hydraulic components with electronic components in the steering system of a vehicle.

A partial architecture of the SBW system is shown in Figure 4.10. There are two ECUs (rest of the ECUs are not shown for simplicity) that are connected to a single Controller Area Network (CAN) bus. The Steering Control (SC) ECU receives inputs from steering angle, steering torque and vehicle speed

Algorithm 1 DL2RCM transformation

```
84: function TC2TC(FunctionalDesignPrototype fdp, Mode mo)
85:     for tc in fdp do
86:         Event startTC = tc.scope.stimulus;
87:         Event endTC = tc.scope.response;
88:         ConnectorData conS = FIND(mo.assembly, startTC);
89:         ConnectorData conE = FIND(mo.assembly, endTC);
90:         if tc is AgeConstarint then
91:             DataAgeStart startA = CREATEDATAAGESTART(tc);
92:             DataAgeEnd endA = CREATEDATAAGEEND(tc);
93:             ASSIGNPORTS(startA, endA, conS, conE)
94:         else
95:         end if
96:         if tc is ReactionConstarint then
97:             DataReactionStart startR = CREATEDATAREACTION-
    START(tc);
98:             DataReactionEnd endR = CREATEDATAREACTIO-
    NEND(tc);
99:             ASSIGNPORTS(startR, endR, conS, conE)
100:        else
101:        end if
102:    end for
103:    for part in fdp do
104:        TC2TC(part, mo.assembly)
105:    end for
106: end function
```

sensors. It also receives a CAN message from the Wheel Control (WC) ECU. It sends two CAN messages: one carries steer angle and torque signal; while the other carries feedback signals. Based on all the inputs, it calculates the feedback steering torque and sends it to the feedback torque actuator which is responsible for producing the turning effect of the steering. Similarly, the WC ECU receives inputs from wheel angle and torque sensors. Depending upon these signals and CAN messages received from the SC ECU, it calculates the wheel torque and produces actuation signals for the wheel actuators. It also sends one CAN message carrying wheel torque signal.

For the sake of simplicity and intuitive presentation of the transformation, the simplified internal software architecture of WC ECU is modeled with

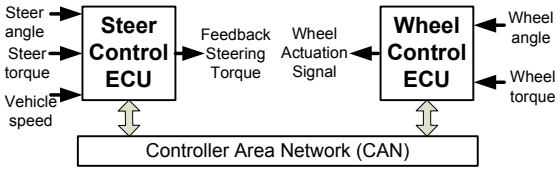


Figure 4.10: Architecture of the steer-by-wire system

EAST-ADL using EAST-ADL Rubus Designer[4] as shown in Figure 4.11.

There are four Software Components (SWCs)[5] in the simplified software architecture. We specify two timing constraints, namely age and reaction using TADL2. These constraints put a restriction of 20 ms on the time between the acquisition of sensor signals at the WC ECU and the production of wheel actuation signals by the actuator SWC. These constraints are internally referenced to the components on which they are specified. For convenience, the start and end points for these constraints are identified using the solid-line arrow, as shown in Figure 4.11.

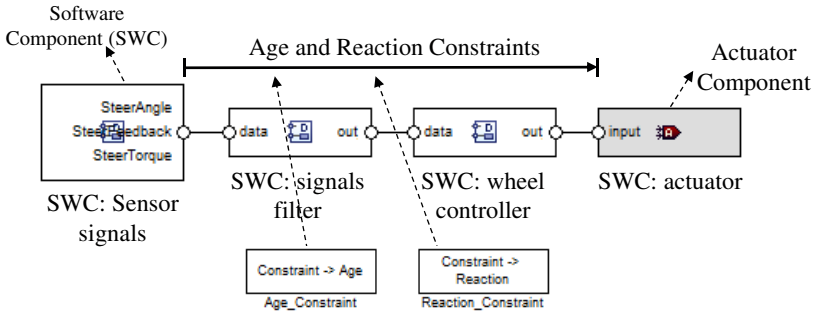


Figure 4.11: Software architecture of WC ECU modeled with EAST-ADL and TADL2

Figure 4.12 shows the Ecore model conforming to the EAST-ADL design level metamodel depicted in Figure 4.9. By applying the DL2RCM transformation presented in Section 4.4, the Ecore model in Figure 4.13 is obtained.

[4]http://www.arcticus-systems.com

[5]An SWC corresponds to a Software Component and a Software Circuit in EAST-ADL and RCM respectively.

Please note that, the model in Figure 4.13 is conforming to the RCM meta-model. Without going into the details of the transformation process, it can be easily noted how the RCM elements were translated from the related EAST-ADL elements. For instance, the RCM SWC SFN_FT it has been translated from the EAST-ADL DesignFunctionType SFN_FT by means of the C2C function. The same applies to all the RCM elements.

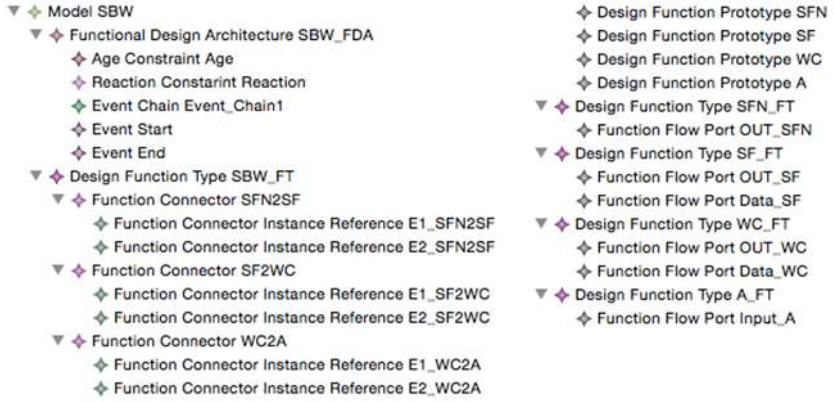


Figure 4.12: Serialized model of the EAST-ADL WC ECU architecture

A representation, given in Rubus Designer concrete syntax, of the model showed in Figure 4.13, is presented in Figure 4.14. The specified TADL2 timing constraints (i.e., Age and Reaction) in Figure 4.11 are also translated to RCM timing constraints shown by “Age Start”, “Age End”, “DR Start” and “DR End” objects in Figure 4.14.

We assume that all tasks corresponding to the four SWCs in Figure 4.14 have equal priorities. Moreover, we consider these tasks to be the highest priority tasks in the WC ECU. The worst-case execution times of these components are selected between the range 60 μs - 2000 μs. The analysis engines calculate the age and reaction delays for only those component chains (represented by task chains at runtime) on which the timing constraints are specified (there is only one component chain in Figure 4.14 on which these delays are specified). The calculated age and reaction delays are 5360 μs and 15360 μs respectively. A comparison between the specified constraints and calculated delays shows the system satisfies the specified timing constraints.

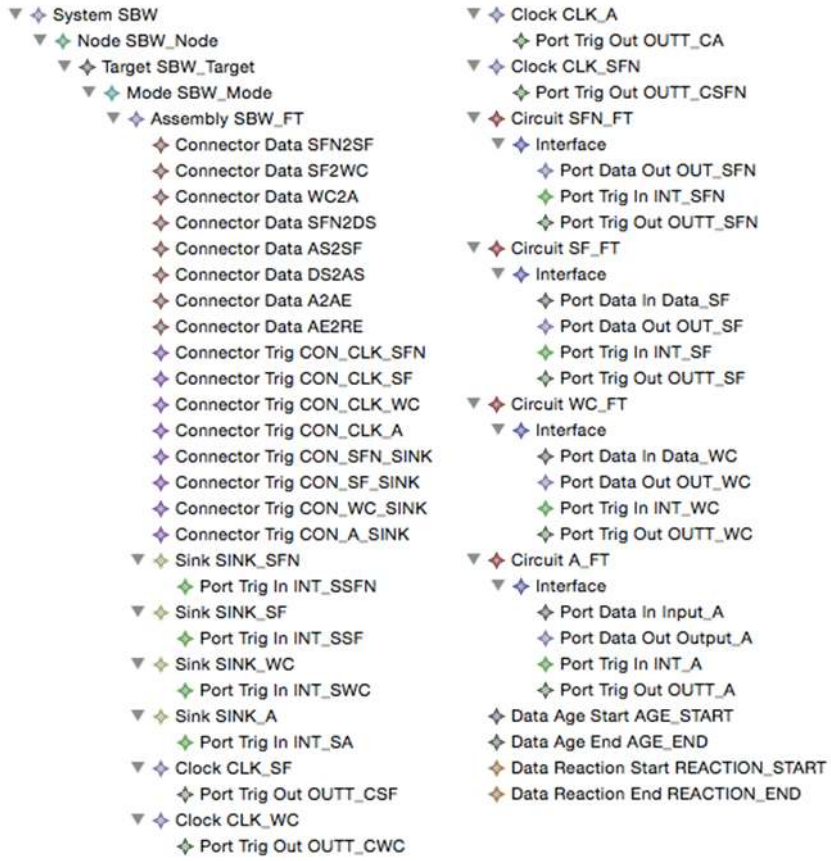


Figure 4.13: Serialized model of the RCM WC ECU architecture

4.6 Evaluation and discussion

The work discussed in this article finds its motivation and application context within the modernization efforts done by Articus Systems AB devoted to port RUBUS toolset into a proper model-driven development environment. In this respect, as mentioned so far, defining appropriate metamodels is a fundamental step towards enabling the implementation of MDE techniques. As a consequence, the RCM metamodel has been developed with precise goals in mind,

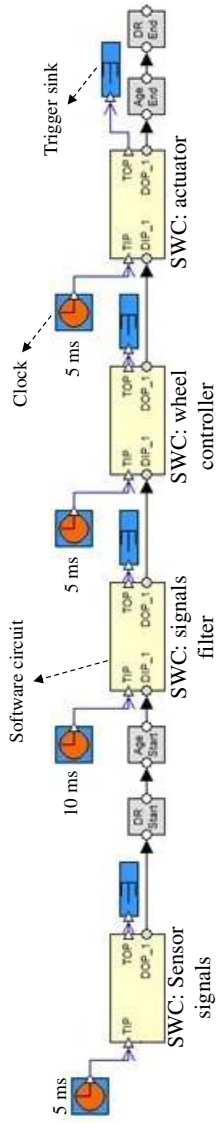


Figure 4.14: Translated software architecture of WC ECU in RCM

as follows:

backward compatibility: the metamodel should allow an easy migration of legacy RUBUS artifacts into the new modeling environment;

maintainability: the metamodel should enable a better management of RCM updates and refinements;

extensibility: the metamodel should disclose the opportunity to integrate in a smooth way RUBUS modeling environment in a typical automotive application development chain.

The first requirement has been addressed by reverse engineering the internal representation of RCM into the RUBUS tool, where the metamodeling activity both polished some redundancies due to the lower level of abstraction representation of modeling entities and optimized model traversals. These activities resulted in the addition of 6 elements and the refinement of 5 elements hierarchies. The metamodel illustrated in this paper has been tested and validated against several existing industrial system designs, e.g., modeling of i) Autonomous Cruise Control System that consists of 4 nodes (ECUs), 17 assemblies and 36 SWCs [4]; and ii) Intelligent Parking Assist System that consists of 2 nodes and 42 SWCs [24]. Moreover, ongoing work is addressing the incremental substitution of current features with replacements implemented through model-based mechanisms. In the long run, this migration will produce a new RUBUS modeling environment entirely based on metamodels, appropriate rendering of model entities, and model transformations.

With respect to the maintainability aspect, by building-up the development environment on the RCM metamodel allows to decouple modeling concepts from their rendering and the automated features provided as part of the tool. This means that extensions/refinements of RCM cause modifications of the current metamodel, which in turn trigger co-evolutions of interconnected artefacts [25]. Despite managing metamodel evolutions is not always straightforward, having an explicit link between RCM changes and metamodel manipulations allows to perform an impact analysis of the refinements and to precisely locate where changes will affect existing artefacts. Notably, especially in industrial contexts it is not unfrequent the need for local customisations of tools requiring *ad hoc* adaptations. On the one hand by using a higher level of abstraction approach allows e.g. to un/hide modelling elements, augment/reduce the number of modelling views, and so on. On the other hand, the need

for metamodel modifications limits the dangerous practice of hardcoding customisation directly on the implementation code of the modelling environment, which hinders its maintainability in the long run.

The last requirement targets the general trend of incrementally adopting higher abstraction level approaches to deal with the development of industrial systems (see also the discussion that follows in the remainder of this section). In particular, it requires RUBUS to be open enough to be integrated in a tool chain. The proposed metamodel-based solution supports tool integration contexts by permitting the definition of model transformations acting as import/-export utilities from a tool to another. The transformation from EAST-ADL to RCM and its application illustrated in Section 4.4 and 4.5, respectively, are a practical demonstration about the tool integration potentials disclosed by the adoption of a model-driven approach. Writing and testing the tool integration transformation is a one time effort; then the translation can be used for all the models produced by means of the same tools, as long as the metamodels are not modified. Also from a performance perspective, the transformation operates in negligible time. In fact, the transformation operates in 40ms, where 39ms are due to the loading of the involved models and 1ms is due to the transformation execution.

From a broader perspective, introducing higher level of abstraction approaches to the development of complex systems is an indisputable trend in modern software engineering practice. In this respect, industry is very often facing the issue of integrating new, task-specific tools, with the rest of legacy systems and development environments. In particular, if the constellation of adopted tools does not integrate in a seamless chain, manual effort is required to close the gap between tools and perform needed translations. Even if feasible, this practice can reveal as time-consuming and error-prone in the long run, especially when the size of the system grows and there are semantics aspects involved in the mapping. On the one hand, model transformations allow to automate the translation process; on the other hand, by using a transformation as tool integration solution provides traceability of translations. Traces not only allow to explicitly represent the correspondences between one tool and another, but they also enable the propagation of information from one domain-specific perspective to another. Notably, in the case study presented in Section 4.5 the forward transformation allows to get an RCM model from EAST-ADL, and carries by the rationale underlying the mapping across these two languages. Moreover, the trace links created during the transformation process allow, for example, to map timing analysis results back to EAST-ADL models.

4.7 Conclusions and future work

In the last twenty years, CBSE has enhanced the software development for vehicular embedded systems. Nevertheless, industry needs to move further towards a seamless development chain for reducing software development costs and time-to-market. Intertwining of MDE and CBSE has been proven to be effective towards this goal.

In this work, by exploiting the interplay between MDE and CBSE, we took initial steps towards the realization of the aforesaid seamless development chain. In details, we i) motivated the usage of RCM within the vehicular domain, by highlighting its unique features against existing CMs, ii) formalized a metamodel based on RCM comprising the concepts able to represent both the software architecture and the related timing constraints,, iii) presented a model-to-model transformation between EAST-ADL Design level and RCM and iv) discussed a case study which mimics a typical industrial scenario.

The formalization of the metamodel serves as base for embracing the MDE vision as well as for restoring the separation of concerns which has been lost during the evolution of the RCM. Due to space limitations, we did not discuss the complete RCM timing package, but we rather focused on the elements representing the most recent timing constraints, information and analyses introduced and practically used within the industrial automotive domain.

The DL2RCM transformation outlines the potential benefits gained in having a proper metamodel for RCM, in terms of compliance with the EASTADL based methodology.

As future works, we plan to minimize the assumptions needed in performing the transformation, by using model transformation languages able to fully and practically support non-bijective model transformations. Additionally, we will consider the possibility of using these non-bijective model transformations for design-space exploration. Finally we will, together with our industrial partners, cover the identification of additional languages used along the software development for the vehicular embedded systems, with the aim of formalizing their metamodels and hence enable model transformations for supporting a more extensive tool chain. The work in this paper is supported by the Swedish Knowledge Foundation (KKS) and Swedish Research Council (VR) within the projects FEMMVA and SynthSoft. We thank our industrial partners Arcticus Systems AB and Volvo CE, Sweden.

Bibliography

[1] R. N. Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.

[2] F. Liu, A. Narayanan, and Q. Bai. Real-time systems, 2000.

[3] I. Crnkovic and M. Larsson. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002.

[4] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems*, 10(1), 2013.

[5] AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct., 2013. http://autosar.org.

[6] K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J Lundback, and K.-L. Lundback. The Rubus Component Model for Resource Constrained Real-Time Systems. In *3rd IEEE International Symposium on Industrial Embedded Systems, 2008*, Jun. 2008.

[7] EAST-ADL Domain Model Specification, Deliverable D4.1.1, 2010. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[8] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Y. Papadopoulos, M. O. Reiser, A. Sandberg, D. Servat, R. T. Kolagari, M. Törngren, et al. 11 the east-adl architecture description language for automotive embedded software. In *Model-Based Engineering of Embedded Real-Time Systems*, pages 297–307. Springer, 2011.

[9] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson. Assessing the state-of-practice of model-based engineering in the embedded systems domain. In *Model-Driven Engineering Languages and Systems*, pages 166–182. Springer, 2014.

[10] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.

[11] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, dec. 2008.

[12] J. Bézivin and O. Gerbé. Towards a precise definition of the omg/mda framework. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, 2001.

[13] S. Sendall and W. Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42–45, 2003.

[14] TIMMO Methodology, Version 2, Deliverable 7, Oct. 2009.

[15] TIMMO-2-USE. https://itea3.org/project/timmo-2-use.html.

[16] TADL: Timing Augmented Description Language, Version 2, Deliverable 6, Oct. 2009. The TIMMO Consortium.

[17] The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, 2010. OMG Group, January 2010.

[18] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture*, 60(2):207–220, 2014.

[19] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40:117–134, April 1994.

[20] F. Stappert, J. Jonsson, M. Jürgen, and J. Rolf. A Design Framework for End-To-End Timing Constrained Automotive Applications. In *Embedded Real-Time Software and Systems (ERTS)*, 2010.

[21] A. C. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh. Schedulability and end-to-end latency in distributed ecu networks: formal modeling and precise estimation. In *Proceedings of the tenth ACM international conference on Embedded software*, EMSOFT '10, pages 129–138. ACM, 2010.

[22] Alessio Bucaioni, Saad Mubeen, Antonio Cicchetti, and Mikael Sjödin. Exploring Timing Model Extractions at EAST-ADL Design-level Using Model Transformations. In *International Conference on Information Technology: New Generations (ITNG)*, April 2015.

[23] A. Bucaioni, A. Cicchetti, and M. Sjödin. Towards a metamodel for the rubus component model. In *1st International Workshop on Model-Driven Engineering for Component-Based Software Systems, ModComp 2014, 29 September 2014*, pages 46–56, 2014.

[24] Alessio Bucaioni, Saad Mubeen, John Lundbäck, Kurt-Lennart Lundbäck, Jukka Mäki-Turja, and Mikael Sjödin. From modeling to deployment of component-based vehicular distributed real-time systems. In *International Conference on Information Technology: New Generations (ITNG)*, April 2014.

[25] Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. What is needed for managing co-evolution in mde? In *Proceedings of the 2Nd International Workshop on Model Comparison in Practice*, pages 30–38, 2011.

Chapter 5

Paper B: Exploring Timing Model Extractions at EAST-ADL Design-level Using Model Transformations

Alessio Bucaioni, Saad Mubeen, Antonio Cicchetti and Mikael Sjödin

In Proceedings of the 12th International Conference Information Technology: New Generations (ITNG), IEEE, Las Vegas, Nevada (USA), April, 2015

Abstract

We discuss the problem of extracting control and data flows from vehicular distributed embedded systems at higher abstraction levels during their development. Unambiguous extraction of control and data flows is vital part of the end-to-end timing model which is used as input by the end-to-end timing analysis engines. The goal is to support end-to-end timing analysis at higher abstraction levels. In order to address the problem, we propose a two-phase methodology that exploits the principles of Model Driven Engineering and Component Based Software Engineering. Using this methodology, the software architecture at a higher level is automatically transformed to all legal implementation-level models. The end-to-end timing analysis is performed on each generated implementation-level model and the analysis results are fed back to the design-level model. This activity supports design space exploration, model refinement and/or remodeling at higher abstraction levels for tuning the timing behavior of the system.

5.1 Introduction

The intrinsic complexity of vehicular embedded systems demands for development methodologies and technologies that are able to cope with it. In the last decades, Component-Based Software Engineering (CBSE) [1, 2], Model-Driven Engineering (MDE) [3] and their crosplay have gained acceptance due to their ability to both reduce the development complexity, by raising the abstraction level, and to cope with the most arduous aspects of these systems such as timing and safety requirements [2].

EAST-ADL [3] together with its development methodology has been getting closer and closer towards the status of de-facto standard within the automotive domain. It defines a top down development process promoting the separation of concerns through a four-level architecture, where each level is designed for hiding details pertaining to higher or lower levels. At the lowest level, i.e., implementation level, EAST-ADL makes use of AUTOSAR [5], which is an industrial initiative to provide standardized software architecture for the development of vehicular embedded systems. While EAST-ADL methodology has been successful in raising the software development abstraction level, it provides few means for coping with the timing requirements of such software systems. In the past few years, several initiatives such as TIMMO [4] and TIMMO2USE [5] and their outcomes including TADL [6] and TADL2 [10] languages, tried to provide AUTOSAR with a timing model. Nevertheless, they did not fully succeed with this goal at various abstraction levels because AUTOSAR explicitly hides some implementation-level information which is necessary for building a timing model from the software architecture.

Nowadays, automotive industry needs development methodologies and technologies able to cope with the timing requirements of such software systems. Nevertheless, current industrial needs push for having such end-to-end timing analysis earlier during the development process, i.e., at the design level. Industry is currently reusing most of the software architecture from previous projects, that means, some crude software architecture is already available in the early stages of the software development. In this context, it is beneficial to perform early timing analysis for Design Space Exploration (DSE) and software architecture refinements.

5.1.1 Paper Contribution

We target core challenges that are faced when end-to-end timing models are extracted to support end-to-end timing analysis at higher abstraction levels and

earlier stages of the software development of vehicular distributed embedded systems. These challenges include extraction of data and control paths at the implementation level from the design-level models; transformation of multiple implementation-level models from a single design-level model; and dealing with these transformed models from the timing analysis point of view. In order to deal with these challenges, we propose a two-phase methodology that exploits the principles of MDE and CBSE. In the first phase, the software architecture of the system at the EAST-ADL design level is automatically transformed to all legal implementation-level models, e.g., models that are build using the Rubus Component Model (RCM) [7]. Whereas in the second phase, the end-to-end timing analyses are performed on each generated implementation-level model. The analysis results of all or selected implementation-level models are fed back to the design-level model. Thus, the methodology provides a support for DSE and models refinement. Moreover, it supports remodeling at higher abstraction levels for the purpose of tuning the timing behavior of the system.

5.1.2 Relation with Authors' Previous Works

In [8], we provide a method to extract timing models and perform end-to-end timing analysis of component-based distributed embedded systems. In [9], RCM is presented as an alternative to AUTOSAR in the EAST-ADL development methodology and its usage is discussed for enabling end-to-end timing analysis at the lowest EAST-ADL abstraction level, i.e., implementation level. In [10], RCM is extended with a concrete meta-model definition. In [11], the translation of timing constraints from the design- to the implementation-level models is provided. However, the translation is done manually and is limited by the constraint such that it only considers that implementation-level model which results in worst-case response times and delays. In comparison with above works, this paper presents a novel two-phase methodology to automatically transform the software architecture of the system at the EAST-ADL design level to all legal implementation-level models (RCM models). The existing analysis engines in the Rubus analysis framework perform timing analysis on each generated implementation-level model. The analysis results are then fed back to the design-level model to support DSE and models refinement.

5.2 Background and Related Works

5.2.1 EAST-ADL Development Methodology

EAST-ADL defines a top-down development methodology that promotes the separation of concerns through the usage of four different abstraction levels, where each level provides a complete definition of the system under development for a specific perspective. Figure 5.1 shows the abstraction levels architecture together with the methodologies, models and languages used at each level.

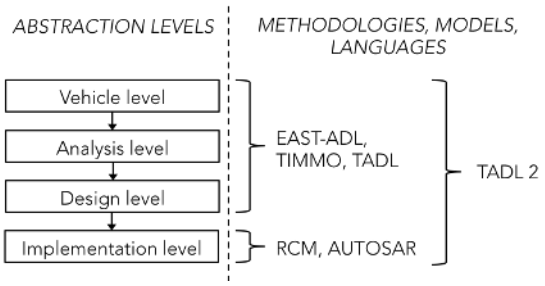


Figure 5.1: EAST-ADL abstraction levels

Vehicle level

The vehicle level, also known as end-to-end level, serves for capturing all the information regarding what the system is supposed to do, i.e., requirements and features on the end-to-end functionality of the vehicle. Feature models and requirements can be used for showing what the system provides and, eventually, how the product line is organized in terms of available assets.

Analysis level

At this level, the end-to-end functionalities are expressed using formal notations. Behaviors and interfaces are specified for each functionality. Yet, design and implementation details are omitted. At this stage, high-level analysis for functional verification can be performed.

Design level

At this level, the analysis-level artifacts are refined with more design-oriented details. The architecture of the system is redefined in terms of software, hardware and middleware architectures. Also, software functions to hardware allocation is expressed.

Implementation level

The design-level artifacts are enriched with implementation details. Component models are used to model the system in terms of components and their interconnections. The code for vehicle functions can be synthesized from the software component architecture.

5.2.2 The Rubus Component Model (RCM)

Rubus [1] is a collection of methods, theories and tools for model- and component-based development of resource-constrained embedded real-time systems. It is developed by Arcticus Systems in collaboration with Mälardalen University. Rubus is mainly used for development of vehicles control functionality by several international companies. The Rubus concept comprises of RCM and its development environment Rubus-ICE (Integrated Component development Environment), which includes modeling tools, code generators, analysis tools and run-time infrastructure. RCM has been recently extended with a concrete meta-model definition [10] for embracing the MDE vision and streamlining the modeling language.

RCM is used for expressing the software architecture in terms of software components and interconnections. A software component in RCM is called Software Circuit (SWC) and represents the lowest-level hierarchical element. Its purpose is to encapsulate basic functions. RCM distinguishes the SWCs interactions by separating the data flow from the control flow. The latter is defined by triggering objects, i.e., clocks and events. SWCs communicate with each other via data ports. RCM facilitates analysis and reuse of components in different contexts by separating functional code from the infrastructure that implements the execution environment. Within the context of above-mentioned abstraction levels in Figure 5.1, RCM is used at the implementation level.

[1] http://www.arcticus-systems.com

5.2.3 End-to-end Timing Models and Analyses

An end-to-end timing model consists of timing properties, requirements, dependencies and linking information of all tasks, messages and task chains in the distributed embedded system under analysis[2]. It can be divided into timing and linking models. For instance, consider a task chain distributed over three nodes connected by a network as shown in Figure 5.2. The system timing model contains all the timing information about the three nodes and the network. Whereas the system linking model contains all the linking information in the task chains, including the control and data paths.

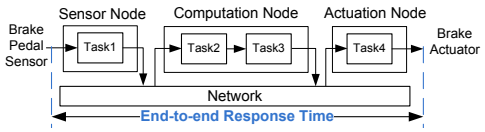


Figure 5.2: Example demonstrating end-to-end response time

The analysis engines [8] use these models for performing end-to-end timing analyses. The analyses results include response-time of tasks and messages as well as system utilization. Also, the analysis engines calculate the end-to-end response times and delays. The end-to-end response time of a task chain is equal to the elapsed time between the arrival of an event, e.g., the brake pedal sensor input in the sensor node and the response time of task, e.g., the brake actuation signal in the actuation node as shown in Figure 5.2.

Within a task chain, if the tasks are triggered by independent sources, then it is important to calculate different types of delays such as age and reaction. Such delays are crucial in control systems and body electronics domains, respectively. An age delay corresponds to the freshness of data, while the reaction delay corresponds to the first reaction for a given stimulus. In order to explain the meaning of reaction and age delays, consider a task chain in a single-node system as shown in Figure 5.3. There are two tasks in the chain denoted by τ_1 and τ_2 and triggered by independent clocks of periods 25ms and 5ms respectively. Let the Worst-Case Execution Times (WCETs) of these tasks be 2ms and 1ms respectively. τ_1 reads data from the register Reg-1 and writes data to Reg-2. Similarly, τ_2 reads data from the Reg-2 and writes data to Reg-3. Since, the tasks are activated independently with different clocks, there can be multiple outputs (Reg-3) corresponding to one input (Reg-1) to the chain as

[2]We refer the reader to [9] for details about the timing model.

shown by several uni-directional arrows in Figure 5.4. The age and reaction delays are also identified in the figure. These delays are equally important in distributed embedded systems.

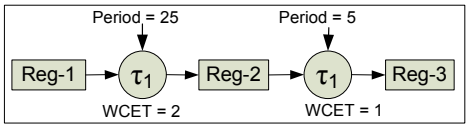


Figure 5.3: A task chain with independent activations of tasks

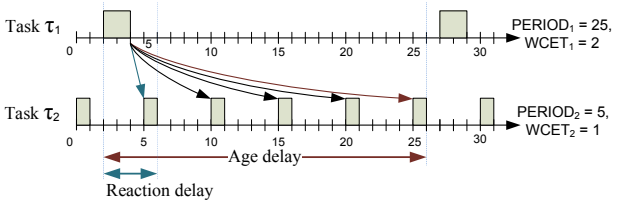


Figure 5.4: Example demonstrating end-to-end delays

5.2.4 Model Driven Engineering (MDE) and Janus Transformation Language (JTL)

MDE is a discipline which aims to abstract software development from the implementation technology by shifting the focus from the coding to the modeling phase. In this context, MDE promotes models and model transformations as first-class citizens. Models are seen as an abstraction of a real systems, built for a specific purpose [3]. Whereas, model transformations can be seen as a gluing mechanism among models [4]. Rules and constraints for the models' construction are specified in the so-called metamodels, i.e., a language definition to which a correct model must conform.

JTL [14] is a declarative model transformation language tailored to support bidirectionality and change propagation. The JTL transformation engine is implemented by means of the Answer Set Programming (ASP) [12], that is a form of declarative programming oriented towards difficult search problems and based on the stable model (answer set) semantics of logic programming. In JTL, a model transformation between a source and a target model, is specified

as a set of relations among models, which must hold for the transformation to be successful. The transformation engine considers such mapping rules for generating the set of all possible solutions. Then, it can refine the generated set by applying constraints on the generated target models, i.e., meta-model conformance rules.

5.2.5 MDE for DSE

During the last decades, MDE has been successfully employed for DSE. In [13], the author exploit JTL for implementing an automatic deployment exploration technique based on refinement transformations and platform-based design. The technique is validated upon an automotive case study using an AUTOSAR-like metamodel. [14] presents a pattern catalog for categorizing different MDE approaches for DSE. It demonstrates the usage of the identified patterns with a literature survey. The work in [15] defines a guided DSE approach based on selection and cut-off criteria defined using dependency analysis of transformation rules and an algebraic abstraction. Cutt-off criteria are used to identify dead-end states, while selection criteria are used to order applicable rules in a given state. The methodology has been effectively evaluated upon a cloud configuration problem.

5.3 Problem Statement

In order to support the end-to-end timing analysis at the design level, the end-to-end timing model should be extracted from the design-level model of the application. Consider the design-level model of a component chain consisting of three software components shown in Figure 5.5. Among other parameters, complete control (trigger) and data paths along component chains (task chains at run-time) must be unambiguously captured in the timing model. Unambiguous extraction of control and data paths from the system are vital for performing its timing analysis.

A control path captures the flow of triggers along the components chain, e.g., control path of the chain in Figure 5.6(b) can be expressed as $\{\{\text{Sensor} \rightarrow \text{Controller}\}, \{\text{Actuator}\}\}$. This means that Controller SWC is triggered by Sensor SWC, while Actuator SWC is triggered independently. Similarly, control paths of the chains shown in Figure 5.6(a) and Figure 5.6(c) can be expressed as $\{\{\text{Sensor} \rightarrow \text{Controller} \rightarrow \text{Actuator}\}\}$ and $\{\{\text{Sensor}\}, \{\text{Controller}\}, \{\text{Actuator}\}\}$ respectively. It should be noted that the three component chains

shown in Figure 5.6 are modeled at the implementation level using the Rubus-ICE tool suite.

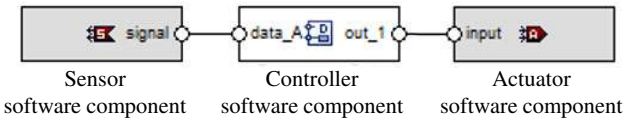


Figure 5.5: Design-level model of a component chain

The main challenge faced during the extraction of end-to-end timing models at the design level is the lack of clear separation between control and data paths. Although TADL2 augments EAST-ADL with some timing information at the design level, the support for clear separation and unambiguous extraction of control and data flows is still missing. At the implementation level, e.g. in RCM, these paths are clearly separated from each other by means of trigger and data ports as shown in Figure 5.6. A trigger output port of an SWC can only be connected to the trigger input port(s) of other SWC(s). Similarly, a data output port of an SWC can only be connected to the data input port(s) of other SWC(s). Hence, the trigger and data paths can be clearly identified and extracted in the timing model. Whereas at the design level, the components communicate by means of *flow ports* as shown in Figure 5.5. A flow port is an EAST-ADL object that is used to transfer data between components. It has a single buffer. The data contained in the port is non-consumable and overwritable. Since there is no other explicit information available about this object, it can be interpreted as a data or a trigger port at the implementation level. There is no support to specify explicit trigger paths at the design level. Moreover, a component can be triggered via specified timing constraints on events, modes, or internal behavior of the component. For example, consider again the design-level model of a component chain shown in Figure 5.5. Assume there is a periodic constraint of 10ms specified on this chain. There can be three model interpretations of this chain at the implementation level as shown in Figure 5.6. Consequently, there are three different control flows in these models. The data flow and control flow should be clearly and separately captured in the end-to-end timing model because the type of the timing analysis depends upon it. For example, it is not meaningful to perform end-to-end delay analysis on a trigger chain as shown in Figure 5.6(a) [8].

We have considered a very small part of a large system in the above example. In reality, distributed embedded systems may contain hundreds of soft-

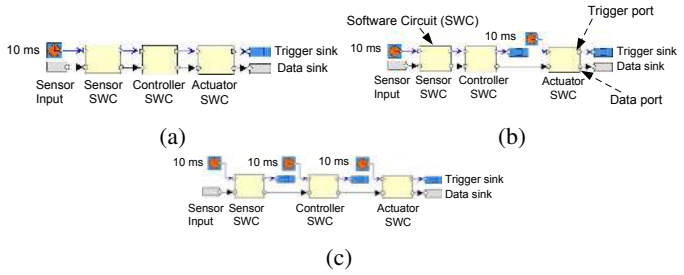


Figure 5.6: Implementation-level models of the design-level model of the component chain in Figure 5.5

ware components and component chains. The component chains, in turn, may be distributed over several nodes or Electronic Control Units (ECUs). Intuitively, there can be a large number of implementation-level model interpretations of the design-level model of a single distributed chain. To the best of our knowledge, RCM is the only model that intends to support high-precision end-to-end timing analysis at the design level[3]. However, it considers only that implementation-level model interpretation of the design-level model which produces worst-case response-times and delays. As a result, the calculated response-times and delays may be very pessimistic (considerably large compared to actual response times and delays). In order to be less pessimistic with the analysis results, the end-to-end timing analysis should be performed on all possible implementation-level model interpretations of a design-level model. The analysis results of all these models should be presented to the user. The user should be able to select the model with respect to the analysis results. This activity also helps in doing DSE and performing model refinements earlier during the development. There is a need for a methodology and corresponding automated model transformations to deal with this problem.

5.4 Proposed Solution and Methodology

In order to address the problem discussed in the previous section, we propose a solution methodology as shown in Figure 5.7. The input to the methodology is the EAST-ADL design-level software architecture of the system under development. Whereas, the output of the methodology consists of the end-to-end

[3]The solution is being prototyped.

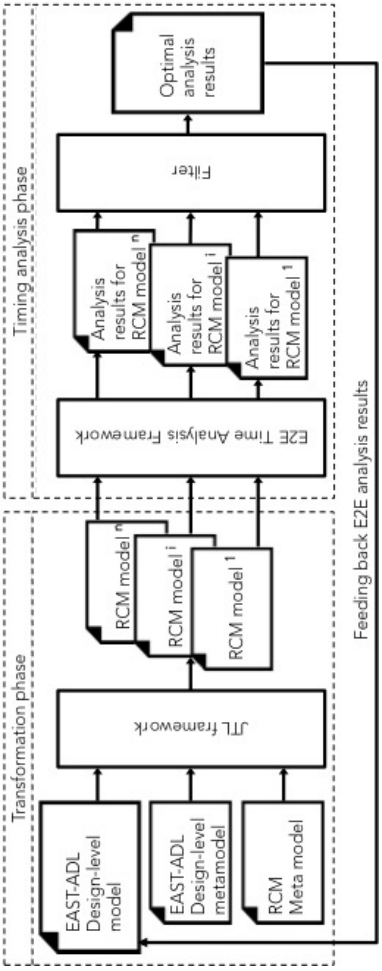


Figure 5.7: Methodology of the proposed solution

timing analysis results that are fed back to the design-level software architecture. The methodology comprises of two major phases (A) transformation phase and (B) timing analysis phase.

5.4.1 Transformation phase

The transformation phase is realized as a model-to-model transformation between EAST-ADL design-level and RCM models. The mapping relation between the related metamodels is a non-surjective relation. We select JTL to implement the transformation because it is able to deal with partial information, information loss and uncertainty [14]. To the best of our knowledge, JTL is the only transformation language with such characteristics. The JTL transformation requires the EAST-ADL design-level model and metamodel as well as the RCM metamodel as inputs. Exploiting the ASP engine, JTL produces, with a single execution, all the possible RCM models for the specified EAST-ADL design-level model. The transformation assumes a one-to-one mapping between each design- and implementation-level component. Although a design-level component can be mapped to more than one implementation-level components, our assumption of one-to-one mapping is based on common practice in industry, especially in the segment of construction-equipment vehicles domain. All the generated implementation-level models have same data flows but different control flows. For instance, consider that the EAST-ADL design-level model shown in Figure 5.5 along with the EAST-ADL and RCM metamodels are provided as input to the JTL framework. The corresponding transformation results into three implementation-level models as shown in Figure 5.6. For a complex embedded application, there can be many such transformations of a design-level model.

5.4.2 Timing analysis phase

In the timing analysis phase, our methodology exploits the end-to-end timing analysis framework of Rubus-ICE [8]. All the generated implementation-level models from the previous phase are provided as inputs to the analysis framework. It should be noted that the timing analysis framework operates on the implementation-level models which are annotated with complete timing information. However, in the generated models derived from the previous phase, some of the timing information required to do the timing analysis may be missing. In this respect, we make assumptions to compensate for the missing timing information. For example, if worst-, best- and average-case execution times are not specified at the design level, they can be estimated at the implementation level either using estimations by experts, reusing them from other projects or from previous iterations during the model refinement process. Further, we assume that the execution order of design-level components in a

chain is specified, otherwise we make implicit assumption about it. That is, each component is assumed to execute only after successful execution of preceding component in the chain, unless specified otherwise. This means, a data provider component is assumed to be always executed before the data receiver component. Since this assumption fixes the execution order, it is safe to assume the priorities of the components are equal within the component chain.

Eventually, the analysis framework performs end-to-end response-time and delay analyses on each implementation-level model separately. Once again, consider the three generated implementation-level models shown in Figure 5.6. We assume the WCET of each component to be equal to 1ms. Here we are interested in the end-to-end response times, reaction and age delays among all timing analysis results. These times for the three component chains are (a) 3ms, 3ms, 3ms; (b) 3ms, 10ms, 10ms; and (c) 3ms, 29ms, 19ms respectively. These analysis results are provided to the filter module which selects optimal result(s) depending upon the specified constraints (e.g. constraints on timing or constraints on activation of individual components in a chain, i.e., dependent or independent triggering). The filter can be considered as the designer who selects optimal implementation-level model interpretation of the design-level model based on the analysis results. The filter can also be a logical block making such decisions based on the specified constraints (the process of automating the filter is a future work).

The translation from the design- to the implementation-level models is automatic. Moreover, the translation is not limited by the constraint of considering that implementation-level model which results in worst-case timing behavior. For example, in the case of constrained translation, the design-level model in Figure 5.5 is only translated to implementation-level model of Figure 5.6 (c) because that chain results in worst-case delays. On the other hand, the timing analysis phase in our current methodology provides all possible implementation-level model interpretations of the design-level model. For example, the filter module can select the chain in Figure 5.6(a) or Figure 5.6(b) as optimal because of lower end-to-end delays and provide the corresponding analysis results back to the design level. Based on this feedback, better decisions can be made during DSE or the refinement of the system model. Moreover, the system can be remodeled or decisions can be made such that the timing analysis results in the next iteration are less pessimistic. This can help in fine tuning the timing behavior of the system.

5.4.3 Proof of concept

As a proof of concept we instantiate the above presented methodology within Rubus-ICE as depicted in Figure 5.8. In Rubus-ICE tool suite, Rubus-EAST tool supports modeling of applications with EAST-ADL. There are two options to start the modeling at the design level: i) model directly in Rubus-EAST, or ii) import XMI formats of EAST-ADL models of the application from any other EAST-ADL designer. The transformation phase of the methodology can be implemented as a plug-in for Rubus-ICE denoted as DL-JTL plug-in, where DL stands for Design Level. According to the proposed methodology, this plug-in calls the JTL framework that generates all feasible RCM models corresponding to the design-level model and provides them back to the plug-in. Consequently, the DL-JTL plug-in calls the HRTA and E2EDA plug-ins [8] and provides all generated implementation-level models to them. The HRTA and E2EDA plug-ins, in turn, perform end-to-end response time and delay analyses of all the input models and provide their analysis results back to the DL-JTL plug-in. Finally, the DL-JTL plug-in selects the optimal analysis results and feeds them back to the design-level model of the application in the Rubus-EAST tool. The sequence of above mentioned steps are identified in Figure 5.8.

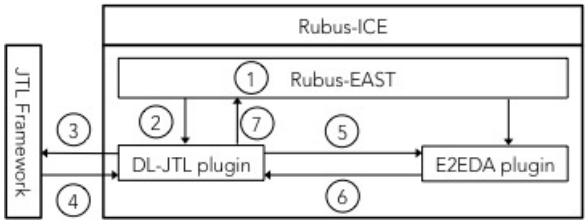


Figure 5.8: Methodology instantiated within Rubus-ICE

5.5 Conclusion

In this work we target core challenges arising when end-to-end timing models are extracted to support end-to-end timing analysis at the design level of the EAST-ADL development methodology. Towards such goal we propose a two-phase methodology that exploits MDE, CBSE and their crossplay. Within the proposed methodology, the design-level model of the system under development is automatically transformed to all possible implementation-level

models. Further, End-to-end timing analyses are performed on each generated implementation-level model; analyses results are filtered based on specified constraints and eventually the analysis results are fed back to the design-level model. Due to lack of needed information, timing model(s) can not be unambiguously extracted from a design-level model. More precisely, more than one timing model may correspond to a single design-level model, as shown in Section 5.3. One way to deal with this issue might be to consider a priori mapping between the design-level model and one of the feasible implementation-level model. In contrast, the proposed methodology is able to generate and manage all the feasible implementation-models (transformation phase) and it is able to choose the implementation-model which better meets the timing requirements, based on the timing analysis results (timing analysis phase). Such methodology naturally supports DSE and model refinements. As a proof of concept, we instantiate the proposed methodology within the Rubus-ICE industrial tool suite. As a future investigation direction, we will, together with our industrial partners, validate, and possibly refine, such methodology upon real industrial design-level models. In this context, it is important to evaluate the performance and scalability of the proposed methodology when the number of alternatives may grow remarkably.

Acknowledgement

This work is supported by the Swedish Research Council (VR) and the Swedish Knowledge Foundation (KKS) within the projects SynthSoft and FEMMVA respectively. The authors would like to thank the industrial partners Arcticus Systems and Volvo Construction Equipment, Sweden.

Bibliography

[1] Thomas A. Henzinger and Joseph Sifakis. The Embedded Systems Design Challenge. In *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[2] Ivica Crnkovic and Magnus Larsson. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002.

[3] EAST-ADL Domain Model Specification, Deliverable D4.1.1, 2010. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[4] TIMMO Methodology, Version 2, Deliverable 7, Oct. 2009.

[5] TIMMO-2-USE. http://www.timmo-2-use.org/.

[6] TADL: Timing Augmented Description Language, Version 2, Deliverable 6, October 2009. The TIMMO Consortium.

[7] K. Hänninen et.al. The Rubus Component Model for Resource Constrained Real-Time Systems. In *3rd IEEE International Symposium on Industrial Embedded Systems, 2008*, June 2008.

[8] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. In *Computer Science and Information Systems, vol. 10, no. 1, pp 453-482, January 2013. ISSN: 1361-1384*.

[9] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Towards extraction of interoperable timing models from component-based vehicular distributed embedded systems. In *International Conference on Information Technology: New Generations*. IEEE, April 2014.

[10] Alessio Bucaioni, Antonio Cicchetti, and Mikael Sjödin. Towards a metamodel for the rubus component model. In *1st International Workshop on Model-Driven Engineering for Component-Based Software Systems*, September 2014.

[11] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Translating timing constraints during vehicular distributed embedded systems development. In *1st International Workshop on Model-Driven Engineering for Component-Based Software Systems*, September 2014.

[12] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.

[13] Joachim Denil, Antonio Cicchetti, Matthias Biehl, Paul De Meulenaere, Romina Eramo, Serge Demeyer, and Hans Vangheluwe. Automatic deployment space exploration using refinement transformations. *Electronic Communications of the EASST*, Recent Advances in MPM(50), Jun. 2012.

[14] Ken Vanherpen, Joachim Denil, Paul De Meulenaere, and Hans Vangheluwe. Design-space exploration in model driven engineering. Technical report, SOCS-TR-2014.4, McGill University, 2014.

[15] Abel Hegedus, Akos Horvath, Istvan Rath, and Daniel Varro. A model-driven framework for guided design space exploration. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, 2011.

Chapter 6

Paper C: Raising Abstraction in Timing Analysis for Vehicular Embedded Systems through Model-Driven Engineering

Alessio Bucaioni

In Proceedings of the Doctoral Symposium at Software Technologies: Applications and Foundations (STAF), L'Aquila, Italy, July, 2015.
Best paper award at the Doctoral Symposium track.

Abstract

The complexity of vehicular embedded systems is continuously increasing and this can negatively affect their development cost and time to market. One way to alleviate these issues is to anticipate analysis of system properties at design time for early architectural refinements. In this paper, we present a licentiate work which aims at contributing to this effort. In particular, considering the importance of timing constraints typical of vehicular embedded systems, we leverage Model-Driven Engineering for realizing an automatic approach which allows the developer to perform timing analysis on design models, without having to manually specify timing elements. The proposed approach, starting from a high-level model of the vehicular embedded application, generates a set of candidate models enriched with timing elements in a semi-automatic manner. Timing analysis is run on the generated models and, based on its results, the approach supports the selection of the best candidate model for a specific, non-empty, set of timing constraints.

6.1 Introduction

During the last decades, ever-growing complexity of vehicular embedded systems development negatively affected their development cost and time-to-market [1]. To mitigate these issues, a common practice is to anticipate analysis of systems properties at design time to drive early architectural refinements. In this paper, we present a licentiate work[1] which aims at contributing to this effort. More precisely, we propose an approach to allow the developer to perform end-to-end and delay timing analysis[2] [3] on design models without having to manually specify their timing elements. Starting from a high-level model of a vehicular embedded application, we provide a semi-automated mechanism for generating a set of candidate models enriched with timing elements with the aim of enabling early timing analysis. Leveraging the timing analysis results, we support the selection of the best candidate model for a specific, non-empty, set of timing constraints.

6.1.1 Context

One of the first attempts in mitigating the increasing complexity of vehicular embedded systems development was the establishment of different views [4], each of which often exploiting a specific language, in the software development. As a result, the vehicular software development was characterized by a plethora of heterogeneous languages each targeting specific aspects related to a particular view. Nevertheless, the usage of heterogeneous languages introduced new challenges towards interoperability, e.g., integration between general purpose languages, e.g., UML, and domain-specific languages, e.g., AUTOSAR. In trying to solve to these challenges, the vehicular embedded research community developed a layered architectural language, namely EAST-ADL [5].

EAST-ADL proposes a top-down development process composed by four different abstraction levels, i.e., vehicle, analysis, design and implementation level. Within EAST-ADL, interoperability is ensured by well-defined relationships among elements in the different abstraction levels. Nevertheless, these relationships are not leveraged in any mechanism supporting the automatic

[1] Within the Swedish Higher Education Systems, the Degree of Licentiate is a third-cycle qualification formally equivalent to half of the Degree of Doctor.

[2] In the remainder of the paper we will refer to end-to-end and delay timing analysis simply as *timing analysis*.

[3] We refer the reader to [2] [3] for further details on timing analysis.

translation of different artifacts through the EAST-ADL abstraction levels. For this reason, automotive industry is currently pushing for having a closer linkage among the EAST-ADL abstraction levels for enabling a seamless development chain that takes into account relationships among different levels. Such a chain would improve the development of vehicular embedded software by providing automation of tedious and error-prone activities, e.g., transition from one level to another. Towards this goal, the vehicular embedded research community is considering the adoption of Model-Driven Engineering (MDE).

EAST-ADL.

EAST-ADL [5] is an architecture description language for modeling product-lines of vehicular embedded systems. Currently it is managed by the EAST-ADL Association together with the European FP7 MAENAD project. EAST-ADL proposes a view over the development process composed by four different abstraction levels, which implicitly ensure separation of concerns through the different engineering phases. Each abstraction level is described by means of metamodeling constructs. Figure 6.1 shows the abstraction levels together with methodologies and languages used at each one of them.

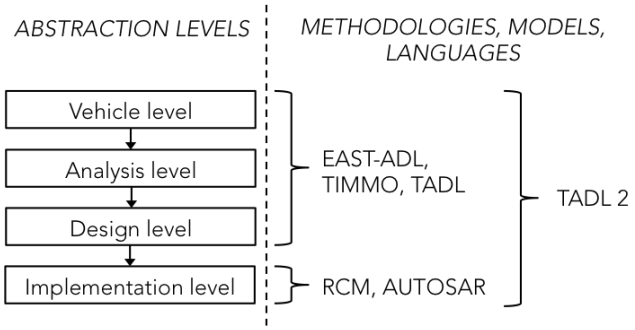


Figure 6.1: EAST-ADL abstraction levels

EAST-ADL does not provide modeling constructs for representing the software implementation architecture. Instead, for the last abstraction level, i.e., implementation level, EAST-ADL suggests the usage of existing modeling languages, e.g., AUTOSAR, the Rubus Component Model (RCM).
*Vehicle Level.*The highest abstraction level is represented by the vehicle level, which captures information regarding the system's functionality. Feature mod-

els can be used for showing what the system provides in terms of functionality. These models are decorated with requirements. The vehicle level is also known as end-to-end level as it serves to capture requirements and features on the end-to-end vehicle functionality.
Analysis level. At the analysis level, vehicle functions are expressed, using formal notations, in terms of behaviors and interfaces. Yet, design and implementation details are omitted. At this stage, high level analysis for functional verification can be performed.
Design level. At this abstraction level, the analysis-level artifacts are refined with design-oriented details: while the analysis level does not differentiate among software, middleware and hardware, the design level explicitly separates them. Allocation of software functions to hardware nodes is expressed at this level too.
Implementation level. At the implementation level, artifacts introduced at the design level are refined with implementation details. At this stage component models, e.g., RCM, AUTOSAR, can be used to model the system in terms of components and interactions among them. The output of this level is a complete software architecture which can be used for code generation.

Rubus Component Model

Rubus Component Model (RCM) is a component model for the development of resource-constrained embedded real-time systems. It is developed by Arcticus Systems AB in collaboration with Mälardalen University and it is currently adopted by several companies as alternative to, e.g., AUTOSAR. In fact, in contrast with its competitors, RCM offers high-precision timing analysis together with a well-established supporting framework.

In the context of EAST-ADL abstraction levels, RCM is used at the implementation level (Figure 6.1). Its main goal is to express the software architecture in terms of software functions and interactions among them. In RCM, the basic entity is the so-called *software circuit* (SWC) which represents the lowest-level hierarchical element in RCM and it encapsulates basic software functions. Each SWC is defined by its *behavior* and *interface*. Interfaces manage the interactions among SWCs via ports. RCM distinguishes between data and control flow. Therefore, the interfaces have two types of ports: *data ports* for the data flow and *trigger ports* for the control flow. SWCs are characterized by run-to-completion semantics meaning that a SWC, upon triggering, reads data from the data input ports, executes its behavior and writes data on the data output ports. SWCs can be grouped and organized in *assemblies*, for

decomposing the system in a hierarchical manner. *Modes* are used to distinguish among different states of the system. That is, each mode describes the architecture of the functions which are relevant for that mode. *Target* entities are used for grouping modes that are deployed on the same Electronic Control Unit (ECU). Moreover, they provide details regarding hardware and operating system. *Node* is a hardware and operating-system independent abstraction of a target entity. Finally, *System* is the top-level hierarchical entity, which describes software architecture for the complete vehicular system. RCM facilitates analysis and reuse of components in different contexts by separating functional code from the infrastructure that implements the execution environment.

The RCM metamodel definition is part of the intended research contributions of the Licentiate thesis.

6.1.2 Paper Outline

The rest of the paper is organized as follows. Section 6.2 describes the research problem. Section 6.3 presents the proposes solution and the related research contributions. Section 6.4 describes the work-to date and the current status. Finally, Section 6.5 and Section 6.6 discuss the validation methodologies and related work, respectively.

6.2 Problem Formulation

In this section, we discuss the research goal for the licentiate work, together with the research challenges to be tackled towards its achievement.

6.2.1 Research Goal

Given the timing constraints typical of vehicular embedded applications, anticipating timing analysis is a way for mitigating development issues, such as cost and time-to-market. Within the EAST-ADL development methodology, the way towards early timing analysis is hampered by the weak linkage between the modeling language used at the implementation level - where timing analysis is usually performed - and the design level. The goal of this licentiate research work is to provide a semi-automated support for allowing timing analysis on design models, without the need of manually adding their timing elements.

6.2.2 Research Challenges

RC 1 – Definition of a metamodeling characterization of RCM.

Although EAST-ADL does not fully embrace the MDE paradigm, the languages defined for the abstraction levels are formalized by metamodeling. For this reason we want to leverage MDE for automating the development of the vehicle embedded software in EAST-ADL. To this end, all the involved languages have to be provided with a proper metamodel definition. In our case, the challenge is the definition of a metamodel for RCM, the language used at implementation level.

RC 2 – Definition of a mapping between EAST-ADL design level metamodel and RCM metamodel.

Due to the lack of timing information, high-precision timing analysis can not be performed at the EAST-ADL design level. One way to solve this issue, is the automatic translation of design level models to implementation level models, i.e., RCM models, on which timing analysis can be performed. RCM models contain in fact timing elements, e.g. clocks, control-flow ports, to mention a few, that cannot be modeled at the design level. These elements represent a variability points in the transition from design to implementation level, meaning that more than one RCM model can be a valid translation of a given EAST-ADL design level model. The challenge is to define and implement a semi-automatic translation such that all the possible RCM models for a given EAST-ADL design level model are produced without any error-prone and time consuming manual activity. Semi-automatic in the sense that, while timing elements (e.g., clock, control flow port) can be fully automatically specified, their completion with actual timing properties (e.g., clock period, wcet) is still guided by the developer.

RC 3 – Definition of a mechanism for the selection of the best RCM model for a set of timing constraints.

Starting from the generated RCM models, the challenge is to define a mechanism able to, based on the analysis results, select the RCM model which better meets the given set of timing constraints. In fact, at the end of the process, starting from a design level model, either an RCM model is chosen or refinements of the design level model are needed. This represents the last step for

exploiting timing analysis results at design level for early enabling software architecture refinements.

6.3 Proposed Solution and Intended Contributions

The contribution of the licentiate work presented in this paper, is the definition of a process which, from a design level model, generates a set of candidate implementation level models enriched with timing elements whose properties are set at generation time by the developer. That is to say, the developer drives the automatic generation of all the relevant combinations of timing elements by *inserting timing properties only once per element* instead of having to manually edit all the generated models. At this point timing analysis can be run and from its results, the best candidate implementation model, for a specific timing property or a set of them, is selected. Figure 6.2 depicts the proposed approach and it also provides a breakdown of the overall contribution in specific research contributions (RCOs).

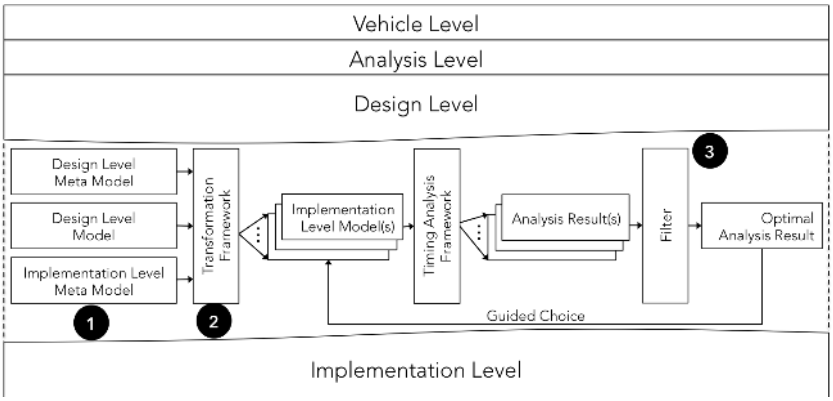


Figure 6.2: Research Contributions

RCO 1 – RCM metamodel.

This contribution, marked as 1 in Figure 6.2, provides a metamodel definition for RCM. The metamodel has been realized as an Ecore model, within the

Eclipse Modeling Framework [4] (EMF). The definition of the metamodel comprised the addition of some modeling elements, e.g., connectors, as well as the refinements of already existing elements relations, e.g., ports and data element hierarchies.

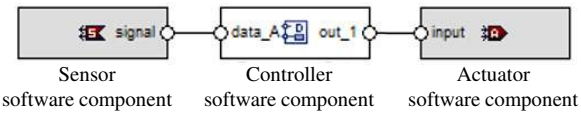
RCO 2 – DL2RCM transformation.

This contribution, marked with 2 in Figure 6.2, provides a model to model transformation between EAST-ADL design level metamodel and RCM metamodel (DL2RCM). The transformation has been implemented by means of a bidirectional model transformation language, namely Janus Transformation Language (JTL). To the best of our knowledge, JTL is the only transformation language able to deal with non-bijective transformations by possibly producing multiple results. The contribution of the DL2RCM transformation is two-fold. On the one hand, it allows the automatic translation of EAST-ADL design level models to RCM models. On the other hand, it does not impose restrictions on the generation of the RCM models, i.e., it is able to generate all the possible RCM models for a given EAST-ADL design level model. For instance, given the EAST-ADL design level model depicted in Figure 6.3a and considering the generation of clocks in the RCM models, the DL2RCM transformation will produce the RCM models depicted in Figure 6.3b.

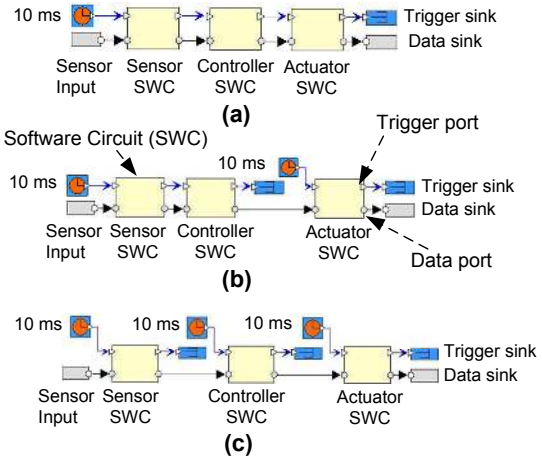
RCO 3 – Mechanism for the best RCM candidate selection.

This contribution, marked as 3 in Figure 6.2, provides a conceptual mechanism supporting the selection of the best RCM model for a specific, non-empty, set of timing constraints as the last step in the process of anticipating timing analysis at design level for enabling early architecture refinements. For each generated RCM model, timing analysis is applied. Analysis results together with a non-empty set of timing constraints, are the inputs of the mechanism that checks analysis results versus timing constraints to identify, possibly the best RCM candidate implementation model. If the mechanism fails in identifying a candidate, early architecture refinements at the EAST-ADL design level model may be needed.

[4] http://www.eclipse.org



(a) EAST-ADL Design level model of a component chain



(b) Possible RCM models of the EAST-ADL design level model of the component chain in Figure 6.3a

Figure 6.3: Source and target model examples for DL2RCM

6.4 Preliminary Work and Current Status

In [6], we present the metamodel definition for RCM which focuses on the definition of metamodeling elements representing the software architecture. A work describing an extension of the RCM metamodel including the definition of new structural elements and elements used for describing timing information is currently under review at the Journal of Systems and Software[5]. In the same work, we also demonstrate the applicability of the RCM metamodel by conducting an automotive application case study. In [7] we propose a two-phase methodology which supports the extraction of timing models from EAST-ADL design-level models with the aim of anticipate timing analysis at design level. Within the proposed methodology, the software architecture at de-

[5]http://www.journals.elsevier.com/journal-of-systems-and-software/

sign level is automatically transformed to all the meaningful implementation-level models. The end-to-end timing analysis is performed on each generated implementation-level model and the analysis results are fed back to the design-level model. The aforesaid methodology is based on a model-to-model transformation between models conforming to the EAST-ADL design-level metamodel and models conforming to RCM metamodel (DL2RCM). The DL2RCM transformation has been implemented within the Eclipse Modeling Framework[6] using JTL [8]. The proposed methodology is concretely integrated in an industrial tool used by automotive companies. We are planning to validate the methodology by means of an industrial case study and submit the results of this validation to the International Conference on Model Driven Engineering Languages and Systems[7].

6.5 Validation

The works presented in [6], has been evaluated upon an industrial case-study. In [6], we demonstrate the applicability of the RCM metamodel by defining a model-to-model transformation between models conforming to the RCM metamodel and models conforming to AUTOSAR [9]. The models describe a single-node real time system composed of three components. Similarly, in [7], in order to prove the applicability of the proposed methodology, we instantiate the methodology within an industrial tool-suite, namely Rubus-ICE[8] used for the development of vehicular embedded systems. Also we are planning to demonstrate the validity of the methodology with an industrial case-study mimicking the TIMMO2USE break-by-wire validator [10].

6.6 Related Work

This section discusses some literature related to our problem and contributions.

6.6.1 Modeling Languages for Vehicular Embedded Systems

In this section we discuss modeling languages and methodologies specifically tailored for the development of vehicular embedded systems.

[6] http://www.eclipse.org
[7] http://www.modelsconference.org
[8] https://www.arcticus-systems.com/products/rubus-ice/

AUTOSAR.

AUTOSAR [9] is an industrial initiative to provide standardized software architecture for the development of vehicular embedded systems. Within AUTOSAR, the software architecture is defined in terms of *software components* (SWCs) and *Virtual Function Bus* (VFB). VFB handles the virtual integration and communication among SWCs, hiding low-level implementation details. Unlike RCM, no particular focus was directed to specification and handling of timing-related details.

AUTOSAR metamodel describes the software development at a higher level of abstraction compared to RCM metamodel. Unlike RCM, it does not separate control and data flows among software components nor differentiate between the modeling of intra- and inter-node communication. Despite these differences, there are some similarities between AUTOSAR and RCM, e.g., the sender receiver communication mechanism in AUTOSAR is very similar to the pipe-and-filter communication mechanism in RCM. AUTOSAR is more focused on the functional and structural abstractions, hiding the implementation details about execution and communication. Whereas, RCM supports the modeling, analysis and synthesis of the execution environment of software functions. Essentially, AUTOSAR hides the details that RCM highlights.

TIMMO/TIMMO-2-USE.

TIMMO [11], a large EU research project, is an initiative to provide AUTOSAR with a timing model. To this end, it provides a predictable methodology and language, called TADL [12] for expressing timing requirements and constraints. TADL is inspired by MARTE [13] which is the UML profile for the model-driven development of real-time and embedded systems. TIMMO methodology makes use of the EAST-ADL and AUTOSAR interplay. Although the TIMMO project has been evaluated upon prototypes, to the best of our knowledge, there is no concrete industrial implementation of it.

TIMMO-2-USE [10], a follow up project, presents a major redefinition of TADL and new functionality for supporting the AUTOSAR extensions regarding timing model. Although both TIMMO and TIMMO-2-USE attempt to annotate AUTOSAR with a timing model, this may be hard to accomplish as AUTOSAR aims at hiding implementation details of execution environment and communication using the VFB.

TIMMO-2-USE proposes a methodology for the software development of vehicular embedded systems solving particular issues related to the timing modeling. The methodology is based on the EAST-ADL abstraction levels

and it introduces, for each level, a set of activities to perform in order to ensure the software timing requirements. Compared to the proposed solution, the TIMMO-2-USE methodology does not provide of automation nor supports early timing analysis.

6.6.2 Model-Driven Engineering for Vehicular Embedded Systems

In the last decades, several works investigated how to automatize the interplay between EAST-ADL and AUTOSAR through MDE. In this respect, in [14], the authors investigate the relationship between concepts of EAST-ADL and AUTOSAR. The authors use three case studies as drivers for the empirical establishment of these relationships. Nevertheless, the work only considers behavioral aspects and timing constraints from the control systems development view. Thus, it might be considered as a first step towards the automatic synthesis of an AUTOSAR architecture from EAST-ADL.

Similarly, in [15] the authors present a mapping between EAST-ADL and AUTOSAR artifacts. Nevertheless the mapping is limited only on few EAST-ADL elements and events. Furthermore, as EAST-ADL evolved, the mapping is no longer valid.

The work in [16] aims at achieving model synchronization between SysML and AUTOSAR using Triple Graph Grammars (TGG) [17]. The proposed approach is developed in an industrial project. The work represents one of the first attempts in using MDE for achieving model synchronization within the vehicular embedded domain. Nevertheless, compared to EAST-ADL, the adopted SysML profile is very generic and it does not cover most of the vehicular aspects of the software systems. On the other hand, having a more tailored language, such as EAST-ADL, would make the synchronization hard to achieve due to the underneath non-injective relations [18].

Although the work presented in this paper does not address design space exploration, the generation of candidate models and their evaluation are two of the four common steps for design space exploration. The work in [19] describes a MDE approach which allows the automatic selection of the most adequate modeling solution for application, platform, and mapping between application and platform. The approach is based on an iterative algorithm which evaluates a candidate model per iteration, until the optimum is found. With respect our solution, such an approach does not generate all the candidate models in a single execution; rather, by means of heuristics, it considers a model per iteration. Similarly to the work proposed in this paper, in [20], the authors

exploit JTL, for an automatic deployment exploration technique based on refinement transformations and platform-based design. More precisely, JTL is used for realizing the so-called refinement transformations responsible for the design space exploration. Differently from our approach, the refinements transformations are endogenous, i.e., the involved source and target metamodels are the same, as the source and target models both conform to the AUTOSAR metamodel.

Bibliography

[1] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: the state of the practice. *Software, IEEE*, 20(6):61–69, 2003.

[2] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *Procs of CRTS*, 2008.

[3] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems*, 10(1), 2013.

[4] Anthony Finkelstein, Jeff Kramer, Bashar Nuseibeh, Ludwik Finkelstein, and Michael Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(01):31–57, 1992.

[5] EAST-ADL Domain Model Specification, Deliverable D4.1.1, 2010. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[6] Alessio Bucaioni, Antonio Cicchetti, and Mikael Sjödin. Towards a metamodel for the rubus component model. In *1st International Workshop on Model-Driven Engineering for Component-Based Software Systems, ModComp 2014, 29 September 2014*, pages 46–56, 2014.

[7] Alessio Bucaioni, Saad Mubeen, Antonio Cicchetti, and Mikael Sjödin. Exploring timing model extractions at east-adl design-level using model transformations. In *12th International Conference on Information Technology : New Generations*, April 2015.

[8] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Jtl: a bidirectional and change propagating transformation language. In *Software Language Engineering*, pages 183–202. Springer, 2011.

[9] AUTOSAR Techincal Overview, Version 2.2.2. AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1, The AUTOSAR Consortium, Aug., 2008. http://autosar.org.

[10] TIMMO-2-USE. https://itea3.org/project/timmo-2-use.html.

[11] TIMMO Methodology, Version 2, Deliverable 7, Oct. 2009.

[12] TADL: Timing Augmented Description Language, Version 2, Deliverable 6, October 2009. The TIMMO Consortium.

[13] The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, 2010. OMG Group, January 2010.

[14] Tahir Naseer Qureshi, DeJiu Chen, Henrik Lönn, and Martin Törngren. From east-adl to autosar software architecture: a mapping scheme. In *Software Architecture*, pages 328–335. Springer, 2011.

[15] Philippe Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, Mark-Oliver Reiser, David Servat, R Tavakoli Koligari, and DeJiu Chen. Developing automotive products using the east-adl2, an autosar compliant architecture description language. In *Embedded Real-Time Software Conference*. Citeseer, 2008.

[16] Holger Giese, Stephan Hildebrandt, and Stefan Neumann. Towards integrating sysml and autosar modeling via bidirectional model synchronization. In *MBEES*, pages 155–164, 2009.

[17] Holger Giese and Robert Wagner. From model transformation to incremental bidirectional model synchronization. *Software & Systems Modeling*, 8(1):21–43, 2009.

[18] Romina Eramo and Alessio Bucaioni. Understanding bidirectional transformations with tggs and jtl. *Electronic Communications of the EASST*, 57, 2013.

[19] Marcio F. da S. Oliveira, Eduardo W Bri ao, Francisco A. Nascimento, and Flávio R. Wagner. Model driven engineering for mpsoc design space exploration. In *Proceedings of the 20th Annual Conference on Integrated Circuits and Systems Design*, SBCCI '07, pages 81–86. ACM, 2007.

[20] Joachim Denil, Antonio Cicchetti, Matthias Biehl, Paul De Meulenaere, Romina Eramo, Serge Demeyer, and Hans Vangheluwe. Automatic deployment space exploration using refinement transformations. *Electronic Communications of the EASST*, Recent Advances in MPM(50), Jun. 2012.

Chapter 7

Paper D: Anticipating Implementation-Level Timing Analysis for Driving Design -Level Decisions in EAST-ADL

Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, Mikael Sjödin

In Proceedings of the 1st International Workshop on Modelling in Automotive Software Engineering (MASE) at ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (Models), Ottawa, Canada, September, 2015

Abstract

The adoption of model-driven engineering in the automotive domain resulted in the standardization of a layered architectural description language, namely EAST-ADL, which provides means for enforcing abstraction and separation of concerns, but no support for automation among its abstraction levels. This support is particularly helpful when manual transitions among levels are tedious and error-prone. This is the case of design and implementation levels. Certain fundamental analyses (e.g., timing), which have a significant impact on design decisions, give precise results only if performed on implementation level models, which are currently created manually by the developer. Dealing with complex systems, this task becomes soon overwhelming leading to the creation of a subset of models based on the developers experience; relevant implementation level models may therefore be missed. In this work, we describe means for automation between EAST-ADL design and implementation levels to anticipate end-to-end delay analysis at design level for driving design decisions.

7.1 Introduction

The importance of software is growing in practically all industrial sectors. In the automotive domain, software is used, e.g., for improving the safety of the vehicle, the driving experience, and the comfort of the passengers. The electronic system of a modern car can be composed of more than 70 embedded systems running up to 100 million lines of code [1]. As a consequence, development of these systems is a daunting task. Especially painful is to make late discoveries, during testing, that the software system does not deliver a service of acceptable quality w.r.t. timing errors and delays that cause suboptimal performance of important systems such as engine- or stability-control. Thus, *early analysis* of expected timing-behaviors and feasibility of architectural decisions w.r.t. timing requirements would be very welcome as support for design decisions. In this paper we propose a technique to achieve early *timing*[1] analysis.

Among the many methodologies advocating abstraction, separation of concerns, and automation as powerful instruments for dealing with complexity of software development, Model-Driven Engineering (MDE) has progressively gained industrial attention in the past 15 years [2]. In automotive, the adoption of MDE resulted in the standardization of a layered architectural description language, namely EAST-ADL [3].

EAST-ADL proposes a top-down approach relying on four different abstraction levels, i.e., vehicle, analysis, design and implementation, and it provides abstraction and implicitly ensures separation of concerns through the different engineering phases[2]. Each abstraction level, except implementation, is equipped with a specific modeling language. At implementation level EAST-ADL proposes the adoption of existing modeling languages, e.g., AUTOSAR[3] or the Rubus Component Model (RCM) [4]. Due to its high precision timing analysis [5], we consider RCM as the reference modeling language exploited at implementation level. EAST-ADL provides mediums for achieving abstraction and separation of concerns, but it does not come with explicit support for automation among the different abstraction levels. The lack of this crucial means, imperative for a full-fledged MDE approach, leads to a scattered development process where consistency among artefacts is a burden for the developer to bear.

[1] Although other relevant extra-functional properties and related analyses exist, the focus of this work is on timing-related properties and analysis.

[2] In the remainder of the paper we will refer to design level models simply as *design models* and to implementation level models as *implementation models*.

[3] `http://www.autosar.org/`

Due to the lack of detailed timing information (e.g., control flow ports, clocks, to mention few) [5] at design level, timing analysis cannot be performed on design models, which indeed need to be translated to implementation models equipped with needed timing details (e.g., clocks). This translation is usually done manually, driven by the developer's experience and, due to size and complexity of the task, it often considers a one-to-one mapping only. This, besides being tedious and error-prone, may lead to the loss of relevant implementation-model candidates when dealing with complex industrial systems.

In this work, we discuss a methodology which provides automation means for seamlessly linking EAST-ADL design and implementation levels to enable end-to-end delay analysis at design level[4] for supporting design decisions. The importance of exploiting implementation level analysis for taking design decisions resides in the fact that it is *more accurate* than design level analysis, which usually provides estimations and does not suffice industrial needs. The initial idea was introduced in [6], while in this work we focus on its *enhancement*, *concrete implementation* and *deployment* in the automotive context.

The rest of the paper is organized as follows. In Section 7.2 we present related work documented in the literature. In Section 7.3 we describe a running example taken from the automotive domain, and in Section 7.4 we apply the proposed methodology to it. In Section 7.5 we discuss benefits and limitations of the proposed methodology and conclude the paper in Section 7.6.

7.2 Related Work

Model-based approaches supporting timing analyses can be distinguished between those detached from design models, e.g. [7], and those deriving (part of) the necessary information from the design, like [8, 5]. In general, the latter have the advantage of avoiding *discontinuities* due to the abstraction gap between design and analysis [9], even though they have to deal with the intrinsic issue of evaluating multiple implementation choices [10, 11]. Some approaches propose manual mappings to reduce uncertainty between architectural and intermediate models, which is tedious and error-prone when dealing with hundreds of implementation alternatives. Other approaches introduce automation by specifying a predefined one-to-one mapping between architectural and intermediate model elements, like [12] and in a broader way the refinement

[4]For *design level* we mean the EAST-ADL design level throughout the paper.

process prescribed by the Model-Driven Architecture standard[5]. Even though this alleviates time and error-proneness issues of manual approaches, it still relies on a predefined mapping, while in general different implementation alternatives, for the same design, should be evaluated [11].

Our solution proposes to generate a set of possible implementations, each of which entailing (possibly) different timing characteristics. Then, end-to-end delay analysis is run to evaluate them in terms of their timing characteristics and to select the best candidate(s). In this way, relevant design decisions can be anticipated before the final implementation is reached. It is worth noting that a similar mechanism could be realized, notably, by adopting other non-bijective transformation languages, architectural languages (e.g., AADL [13]), and/or other model-based timing analyses approaches (e.g., Simulink[6] or MARTE[7]). However, some preconditions should hold: i) the transformation language should fully support non-bijectivitness; ii) the architectural language shall provide adequate support for timing information at design level of abstraction; iii) the timing analyses shall keep their reliability by relying on the sole design level information (plus the alternatives generated during the derivation process).

The mechanism of implementation models generation resembles the general concept of design-space exploration (DSE) [14], and in particular rule-based DSE [15]. Our approach performs an exhaustive generation of implementation models, enriched with timing details, as derivable from the system architecture designed through EAST-ADL, and constrained by domain-specific rules. Therefore, as opposed to typical DSE, the generation is not meant to provide optimization hints at architectural level [12], rather it shows the best (timing configuration) result given a certain system architecture as input. This procedure is technically identified as quality-driven model transformations [16, 17].

7.3 A Running Example: the Steer-by-wire System

A steering system in a vehicle employs mechanical and hydraulic components between wheels and steering wheel. The Steer-by-wire (SBW) system, which we leverage as running example, replaces most of these components with electronic ones.

[5] http://www.omg.org/mda/
[6] http://www.mathworks.com/products/simulink/
[7] http://www.omg.org/spec/MARTE/

We model the SBW system at the EAST-ADL design level with the help of the Rubus-ICE[8] tool suite. In the hierarchy of a design model, the leaf element is the so-called *design function prototype* (DFP). EAST-ADL implements the type-prototype mechanism, meaning that a DFP represents a specific instance of *design function type*, which defines the type. Within EAST-ADL, DFPs communicate through *function ports*, which are linked via *function connectors*.

It should be noted that one of the main goals of this example is to demonstrate the validity of the proposed methodology. Therefore, in order to better understand the transformation and corresponding selection process, we only consider the internal software architecture of the SC_ECU as depicted in Figure 7.1. The internal software architecture of the SC_ECU consists of six DFPs.

`Steer_Angle` is responsible for acquiring the steer angle sensor input. It passes the acquired values to `Steer_Angle_Preprocessing`. The preprocessed steer angle signal is passed to`Input_Processing`. which also receives the speed of the vehicle from `Vehicle_Speed`. `Input_Processing` passes the processed input data to `FB_Steer_Torque_Computation`, which in turn produces the feedback steering torque and passes it to `Steer_Sensation_Actuator`, which produces the signals for the steering actuator.

The WCETs specified on `Steer_Angle`, `Steer_Angle_Preprocessing`, `Input_Processing`, `Vehicle_Speed`, `FB_Steer_Torque_Computation` and `Steer_Sensation_Actuator` are 120, 200, 280, 120, 1200 and 100 μs, respectively. Since the implementation details are not available at the design level, the WCETs are estimated based on the expert's judgements. The following timing requirement is specified too:

- *"The calculated age and reaction delays shall not exceed 25 ms and 35 ms, respectively."*

Within EAST-ADL, timing requirements are specified by timing constraints [18]. Therefore, there are two end-to-end delay constraints, namely age and reaction, specified on the software architecture of the SC_ECU as shown in Figure 7.1.

The values of the age and reaction constraints are 25 ms and 35 ms respectively.

7.4 Applying the methodology

Design models do not contain the timing information (e.g., control flow) needed for running end-to-end delay analysis. Therefore, in order to leverage this analysis at design level, we propose to automatically translate design to implemen-

[8] `http://www.arcticus-systems.com`

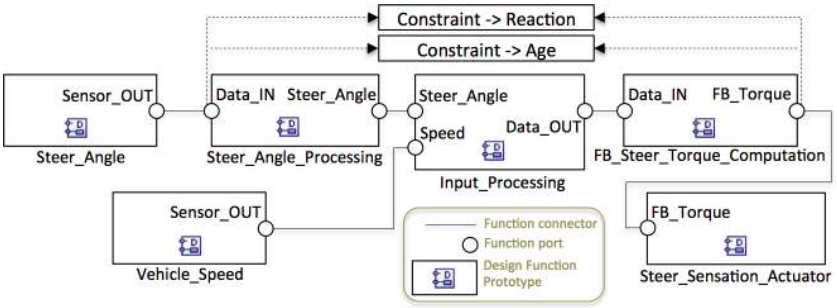


Figure 7.1: Internal software architecture of SC_ECU at design level.

tation models, which contain the needed timing information. Such a translation is non-bijective, meaning that multiple implementation models can be valid translations of a given design model. To this end, the proposed methodology generates all the meaningful (from an analysis perspective) implementation models.

The approach, depicted in Figure 7.2, leverages the interplay of model-driven techniques and model-based analysis and it consists of four main phases, namely *transformation*, *end-to-end delay analysis*, *filtering* and *propagation*. Starting from a design model of an automotive functionality, the approach generates a set of corresponding meaningful implementation models (*transformation phase*, 1 in Figure 7.2) enriched with timing elements whose values are set at generation time by the developer or via configuration files. At this point, end-to-end delay analysis is run on the generated models resulting in a set of analysis results (*end-to-end delay analysis phase*, 2 in Figure 7.2). These results are checked against a non-empty set of timing constraints derived from the timing requirements expressed on the vehicle functionality. The result which better meets the given timing constraints is selected (*filtering phase*, 3 in Figure 7.2); note that multiple results might be equally good and thereby selected. Eventually, the selected candidates are propagated back to the design level by means of annotations to the design model (*propagation phase*, 4 in Figure 7.2).

7.4.1 Transformation Phase

The *transformation phase* relies on a model-to-model transformation, called DL2RCM, between the EAST-ADL design level and RCM metamodels. DL-

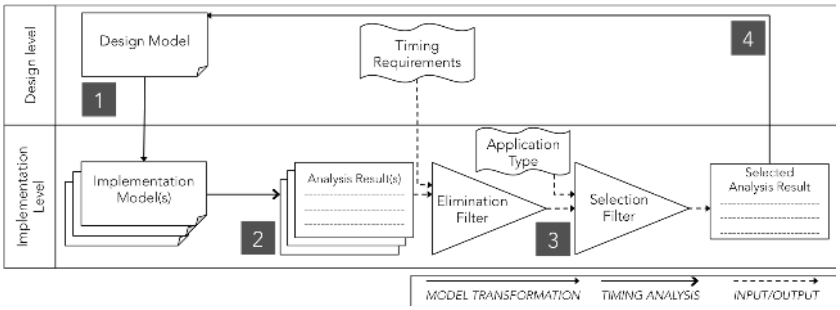


Figure 7.2: Methodology supporting delay analysis at design level.

2RCM is a non-bijective transformation realized within the Eclipse Modeling Framework (EMF)[9] using the Janus Transformation Language (JTL) [19].

JTL is a constraint-based bidirectional model transformation language specifically tailored to support non-bijectivity by generating all the possible solutions at once. It adopts a QVTr-like syntax and allows a declarative specification of relationships between MOF models. The language supports object pattern matching, and implicitly creates traces to record what occurred during a transformation execution. The JTL implementation relies on the Answer Set Programming (ASP) [20], which is a type of declarative programming able to address hard (primarily NP-hard) search problems and based on the model (answer set) semantics of logic programming. The ASP solver finds and generates, in a single execution, all the possible models which are consistent with the transformation rules by a deductive process.

The DL2RCM transformation consists of 28 rules mapping design elements to correspondent implementation elements. In the hierarchy of an RCM implementation model, which represents the transformation's output format, a *software circuit* (SWC) is the leaf element and encapsulates basic software functions. RCM distinguishes between data and control flow therefore a SWC has *data port* and *trigger port*. Within RCM, *Data connectors* link data ports while *Trigger connectors* link trigger ports. *Clocks* and *trigger sinks* are used to initiate and terminate the execution of a SWC, respectively.

Listing 7.1 depicts a fragment of the *DL2RCM* transformation[10], which is expressed in the textual concrete syntax of JTL and applied on models given by

[9] `http://www.eclipse.org/modeling/emf/`

[10] Implementation available at `http://jtl.di.univaq.it/downloads/DL2RCM.zip`

means of their Ecore representation in EMF. In particular, the following rules are defined:

- *C2C*, which maps a function connector to both a data and trigger connectors and triggers the transformation of the connected DFPs;
- *E2C*, which maps a DFP, connected via a function connector, to a SWC;
- *E2CCS*, which maps a DFP, connected via a function connector, to a SWC equipped with a clock and a sink.

The *when* and *where* clauses specify conditions on the relation. For instance, the *where* clause on Line 17 selects the *function ports* linked by the considered *function connector* and triggers the subsequent rules.

E2C and E2CCS define a non-bijective portion of the transformation. In fact, a DFP connected via a connector may be mapped to either a SWC or a SWC equipped with a clock and a sink. This means that, from one single design model, the transformation is able to generate multiple implementation models, each of which containing a unique control flow.

```
1  transformation DL2RCM(dl:designlevel, rcm:RCM) {
2    relation C2C {
3      name, id: String;
4      checkonly domain dl con : designlevel::FunctionConnector
           {
5        name=name,
6        id=id
7      };
8      enforce domain rcm a : RCM::Assembly {
9        connectorData = cd:RCM::ConnectorData {
10         name=name,
11         id=id+"_d",
12         sourcePort = RCM::PortDataOut { ... },
13         targetPort = RCM::PortDataIn { ... }
14       },
15       connectorTrig = ...
16     };
17     where { (con.ends->select(end |
           end.functionPort.oclIsKindOf(designlevel::
18       FunctionFlowPort) and
19       end.designFunctionPrototype.isOfType.isElementary=true)
20       ->forAll(end | E2C(end,a) and E2CCS(end,a) )); }
21   }
22   relation E2CCS {
23     name2, id2: String;
24     checkonly domain dl e :
           designlevel::FunctionConnectorInstanceReference {
25
26       designFunctionPrototype = dfp
```

```
            :designlevel::DesignFunctionPrototype {
27            name=name2,
28            id=id2
29          }};
30        enforce domain rcm a : RCM::Assembly {
31          clock = clk: RCM::Clock {
32            name=name2+'_clock',
33            name=id2+'_clock'
34          },
35          sink = snk: RCM::Sink {
36            name=name2+'_sink',
37            name=id2+'_sink'
38          },
39          circuit = cir :RCM::Circuit {
40            name=name2,
41            id=id2,
42            interface = int :RCM::Interface {
43              name=name2+'_interface',
44              id=id2+'_interface'
45            }}};
46        where { ... }}
47      relation E2C {
48        ...
49    }}
```

Listing 7.1: Fragment of the DL2RCM transformation in JTL.

The DL2RCM model transformation, applied to our design model in Figure 7.1, generates 64 implementation models [11] (one of them is depicted in Figure 7.3). However, considering the end-to-end delay analysis we want to perform, we are only interested in the combinations of those DFPs that are enclosed by the start and end points of the timing constraints.

To this end, we added an OCL logic constraint (shown in Listing 7.2) to the DL2RCM transformation for reducing the set of generated implementation models. It imposes the selection of the implementation model alternatives in which *Steer_Angle*, *Vehicle_Speed* and *Steering_Sensation_Actuator* are transformed by the *E2CCS* rule.

```
1 Circuit.allInstances()->excluding(self.getConstrainedSWC())
2 ->select(c:Circuit | c.getClock().oclIsUndefined()
3 and c.getSink().oclIsUndefined())
```

Listing 7.2: Logic constraint applied to the DL2RCM transformation.

[11] Each SWC can be transformed either via the E2C rule or via the E2CCS rule.

Therefore by enforcing the bijectivity on the *Steer_Angle*, *Vehicle_Speed* and *Steering_Sensation_Actuator*, the DL2RCM transformation generates 8 implementation models[12].

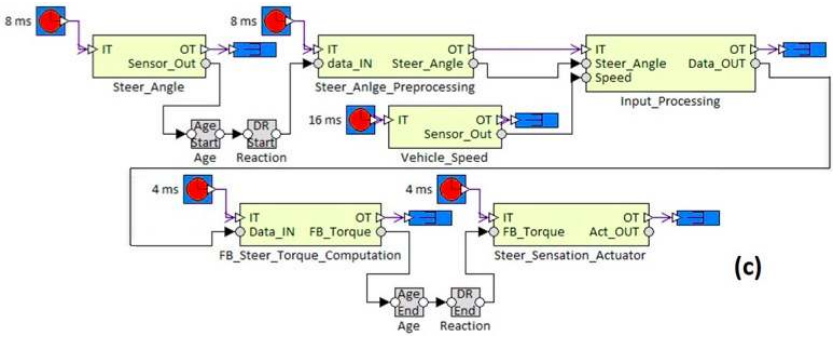


Figure 7.3: Generated implementation model example.

7.4.2 End-to-end Delay Analysis Phase

In this phase, we predict the timing behavior of each generated implementation model by performing the end-to-end delay analysis [21, 5]. We are interested in the calculations of two different delays, namely *age* and *reaction* [5]. Age delay is important in control applications where the interest lies in the freshness of received data. Reaction delay is used to determine the first reaction time for a given stimulus. Our focus is on the Controller Area Network (CAN) which is a event-triggered serial communication bus protocol. We do not use global time stamps (that require tracking of global chronological time) to predict the timing behavior. Instead we use response-time analysis and end-to-end delay analysis. We refer the reader to [21, 5] for the details about the calculations of age and reaction delays.

Once the analysis has been performed on each generated implementation model, the analysis results, which include calculated age and reaction delays for each individual implementation model as shown in Table 7.1, are forwarded to the filtering phase.

[12] All the combinations of the *Steer_Angle_Preprocessing*, *Input_Processing* and *FB_Steering_Torque_Computation* are generated by not enforcing bijectivity.

		Delay Analysis (μs)				Delay Analysis (μs)	
		Age Delay	*Reaction Delay*			*Age Delay*	*Reaction Delay*
Model	*(a)*	26020	30020	**Model**	*(e)*	26020	30020
	(b)	26020	42020		*(f)*	26020	42020
	(c)	18020	22010		*(g)*	18020	18020
	(d)	2020	10020		*(h)*	18020	18020

Table 7.1: Delay Analysis Result for the generated implementation models.

For calculating age and reaction delays, the methodology employs the timing analysis engines implemented in the Rubus-ICE.

7.4.3 Filtering and Propagation Phases

The filtering phase consists of two cascaded filters: the *elimination filter* and the *selection filter*. The timing analysis results are provided as input to the elimination filter together with the non-empty set of timing constraints. In our example, the elimination filter compares the analysis results of each implementation model with the specified age and reaction constraints of 25 and 35 ms respectively. The implementation models identified as (a), (b), (e) and (f) in Table 7.1 violate one or both timing constraints; hence, they are discarded. The remaining models, which satisfy the specified timing constraints (i.e., **(c)**, (d), (g) and (h)), are forwarded to the selection filter.

The selection filter selects the best implementation model based on the requirement concerning the type of application, also received as input. To this end, an application i) contains only single-rate chains, or ii) contains multi-rate chains. In our example, the system shall be developed using multi-rate chains. This means that the implementation models that contain single-rate chains between start and end points of the specified timing constraints are negligible. Therefore, the models identified in Table 7.1 as **(c)**, depicted in Figure 7.3, and (g) are selected[13]. Finally, the models and their analysis results are propagated back to the design model (as annotations done by text-to-model transformations).

[13]The selection filter selects the implementation model with shorter age and reaction delays. In our case two models have same analysis results, thus they are both selected.

7.5 Discussion

Running and leveraging implementation level analysis at higher abstraction levels (e.g., design) brings multiple advantages. First of all, it can help the designer in taking architectural decisions based on much more precise feedback than common design level analysis, which, being based on estimated or guessed properties, are usually just conceived as complementary to implementation level analysis in industrial settings. Moreover, it allows the developer to only focus on design activities exploiting implementation level analysis results without having to investigate nor manually edit implementation models, which are automatically produced and transparent to the developer.

We employ JTL to generate multiple implementation models from one design model by providing different combinations of implementation elements, derived from the design model, and timing elements, added by the transformation. Clearly, the generation of all possible combinations, besides being unnecessary in most scenarios, becomes soon unbearable from a scalability perspective when dealing with complex systems of industrial size. For this reason, we exploit JTL's capability of entailing ASP logic constraints for narrowing the generation space.

We provide a set of default constraints to prune solutions that are evidently meaningless for our analysis. This means that we can enable support for the generation of different classes of models by providing different default constraints. Nonetheless, default constraints do not prevent the generation of dimly meaningless solutions nor high transformation time in case of very complex design models. While the first issue can be solved through analysis and filtering mechanisms, the latter demands additional user-defined constraining based on the specific modeled functionality.

It is interesting to note that the methodology may propagate more than one generated implementation model, along with its timing analysis results, to the design model. This happens only when those results are equally good. In this case, the designer is given the possibility to select among them.

By considering the general development scenario, through our methodology it is possible to disclose the opportunity of shortening time-to-market and leverage expensive resources (e.g., architects, timing experts) more efficiently. More concretely, the simple software system illustrated in this work contains more than fifty components, seventeen in the SC_ECU and ten in each of the four WC_ECUs. This means that starting from such an architecture a designer willing to manually define a proper implementation model would face a space of 2^{57} possible alternatives. It becomes evident that having an auto-

mated mechanism that is able to derive those alternatives and select the best one(s) brings a gain in terms of time, costs and risks in the construction of the implementation.

7.6 Conclusion

The approach proposed in this paper tackles the problem of identifying a suitable implementation choice, in terms of timing characteristics, starting from the software architecture. In general this issue requires the consideration of a number of alternatives that grows exponentially with the number of software components in the architecture. We proposed to solve this by adopting a *quality-driven model transformation* approach and defining a precise mapping between EAST-ADL design and implementation models (defined in terms of the Rubus Component Model). Since in general the mapping of design to implementation models equipped with timing elements is non-bijective, we leveraged the properties of a constraint-based transformation language, JTL, to automatically derive all the meaningful implementation alternatives. Subsequently, generated implementation models are classified in terms of timing results enabling the selection of the best implementation model candidate(s) derivable from the input design model.

The experiment we conducted in collaboration with industrial partners in automotive showed promising results w.r.t. time gains and reduction of possible errors in the creation of a suitable implementation model. Despite the generation and selection processes are transparent to the developer, issues about scalability remain open. In particular, the size of the problem could reach a point such that the generation of implementation alternatives would be intractable. In this respect, a main future investigation direction encompasses the study of smarter generation rules. Another line of research will be devoted to the study of combining the optimisation of multiple system (especially extra-functional) properties.

Acknowledgement

This work is supported by ARTEMIS, the Swedish Research Council (VR), the Swedish Foundation for Strategic Research (SSF), and the Knowledge Foundation (KKS) with the projects CRYSTAL, SynthSoft, PRESS and SMARTCore. The authors would like to thank the industrial partners Arcticus Systems AB and Volvo AB, Sweden.

Bibliography

[1] Robert N. Charette. This Car Runs on Code. *Spectrum, IEEE*, 46(2), 2009.

[2] D.C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39:25–31, 2006.

[3] EAST-ADL Domain Model Specification, Deliverable D4.1.1, 2010. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[4] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K.-L. Lundback. The rubus component model for resource constrained real-time systems. In *Procs of SIES*, pages 177–183, June 2008.

[5] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems*, 10(1), 2013.

[6] A. Bucaioni, S. Mubeen, A. Cicchetti, and M. Sjödin. Exploring timing model extractions at east-adl design-level using model transformations. In *Procs of ITNG*, April 2015.

[7] M. Gonzalez Harbour, J.J. Gutierrez Garcia, J.C. Palencia Gutierrez, and J.M. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *Procs of ECRTS*, pages 125–134, 2001.

[8] S. Anssi, S. Tucci-Piergiovanni, C. Mraidha, A. Albinet, F. Terrier, and S. Gérard. Completing east-adl2 with marte for enabling scheduling analysis for automotive applications. In *Procs of ERTS*, 2010.

[9] B. Selic and L. Motus. Using models in real-time software design. *Control Systems, IEEE*, 23(3):31–42, June 2003.

[10] B. Schatz, F. Holzl, and T. Lundkvist. Design-space exploration through constraint-based model-transformation. In *Procs of ECBS*, pages 173–182, March 2010.

[11] J. Denil, A. Cicchetti, M. Biehl, P. De Meulenaere, R. Eramo, S. Demeyer, and H. Vangheluwe. Automatic deployment space exploration using refinement transformations. *EASST*, Recent Advances in MPM, 2012.

[12] M. Walker, M.-O. Reiser, S. Tucci-Piergiovanni, Y. Papadopoulos, H. Lnn, C. Mraidha, D. Parker, D. Chen, and D. Servat. Automatic optimisation of system architectures using east-adl. *Journal of Systems and Software*, 86(10):2467 – 2487, 2013.

[13] Peter H. Feiler, David P. Gluch, and John J. Hudak. The architecture analysis & design language (AADL): An introduction. Technical Report SEI Technical Note CMU/SEI-2006-TN-011, 2006.

[14] M Gries. Methods for evaluating and covering the design space during early design development. *Integr. VLSI J.*, 38(2):131–183, December 2004.

[15] A. Hegedus, A. Horvath, I. Rath, and D. Varro. A model-driven framework for guided design space exploration. In *Procs of ASE*, 2011.

[16] J. Merilinna. A Tool for Quality-Driven Architecture Model Transformation, 2005.

[17] M.L. Drago, C. Ghezzi, and R. Mirandola. Towards quality driven exploration of model transformation spaces. In *Procs of MoDELS*, pages 2–16. 2011.

[18] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.

[19] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Jtl: a bidirectional and change propagating transformation language. In *Procs of SLE*, pages 183–202. 2011.

[20] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Procs. of the ICLP 1988*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[21] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *Procs of CRTS*, 2008.