

Mälardalen University Press Doctoral Thesis
No.250

A Model-driven Development Approach with Temporal Awareness for Vehicular Embedded Systems

Alessio Bucaioni

January 2018



MÄLARDALEN UNIVERSITY

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

Copyright © Alessio Bucaioni, 2018
ISSN 1651-4238
ISBN 978-91-7485-366-7
Printed by Mälardalen University, Västerås, Sweden

To Lorenzo and Gaia

Abstract

Considering the ubiquitousness of software in modern vehicles, its increased value and development cost, an efficient software development became of paramount importance for the vehicular domain. It has been identified that early verification of non functional properties of vehicular embedded software such as, timing, reliability and safety, is crucial to efficiency. However, early verification of non functional properties is hard to achieve with traditional software development approaches due to the abstraction and the lack of automation of these methodologies.

This doctoral thesis aims at improving efficiency in vehicular embedded software development by minimising the need for late, expensive and time consuming software modifications with early design changes, identified through timing verification, which usually are cheaper and faster. To this end, we introduce a novel model-driven approach which exploits the interplay of two automotive-specific modelling languages for the representation of functional and execution models and defines a suite of model transformations for their automatic integration. Starting from a functional model (expressed by means of EAST-ADL), all the execution models (expressed by means of the Rubus Component Model) entailing unique timing configurations are derived. Schedulability analysis selects the set of the feasible execution models with respect to specified timing requirements. Eventually, a reference to the selected execution models along with their analysis results is automatically created in the related functional model to allow the engineer to investigate them.

The main scientific contributions of this doctoral thesis are i) a metamodel definition for the Rubus Component Model, ii) an automatic mechanism for the generation of Rubus models from EAST-ADL, iii) an automatic mechanism for the selection and back-propagation of the analysis results and related Rubus models to design level and iv) a compact notation for visualising the selected Rubus models by means of a single execution model.

Sammanfattning

Eftersom programvaran är allestädes närvarande i fordon är kostnadseffektiv mjukvaruutveckling för fordon avgörande. Det anses att tidig verifiering av icke-funktionella egenskaper hos fordonsmjukvaran, såsom till exempel timing, pålitlighet och säkerhet, som avgörande för kostnadseffektiv mjukvaruutveckling. Emellertid är tidig verifiering av icke-funktionella egenskaper svårt att uppnå med traditionella mjukvaruutvecklingsmetoder på grund av bristande abstraktion och automatisering.

Denna avhandling syftar till att förbättra effektiviteten hos mjukvaruutveckling för fordon genom att ersätta behovet av sena, dyra och tidskrävande mjukvaruändringar med tidiga, billiga och snabba designändringar som drivs av timingverifiering. Vi introducerar en modelldriven metod för utveckling av inbyggd mjukvara för fordon på plattformar med en eller flera kärnor. Metoden guidar utvecklaren till acceptabla lösningar som identifieras genom schemalägningsanalys.

Acknowledgements

Runners say that you can learn everything you ever wanted to know about yourself in 26.2 miles. In my case, I learnt it in 4 years. Standing by the finish line, I can not think of a more frightening yet terrific experience as the one I am about to finish. However, as cliché as it could sound, I could have not completed this journey without the support of several individuals. The following words are my humble attempt in thanking all of them, knowing that these few words will never suffice the support these people provided me.

First and foremost, I would like to express my deepest gratitude and esteem to my supervisors Mikael Sjödin, Antonio Cicchetti, Federico Ciccozzi and Saad Mubeen whose guidance has been invaluable in my journey. I am especially grateful as they became friends besides being mentors and colleagues. I would like to thank Kurt-Lennart Lundbäck on behalf of Arcticus Systems for providing me with the best workplace a young researcher can wish for. I would like to thank my opponent Professor Matthias Tichy and the examining committee members Dr Henrik Lönn, Associate Professor De-Jiu Chen and Associate Professor Tomas Bures for dedicating me some of their precious time. The workplace at MDH is unique under several aspects. However, the human side of this organisation is just unbeatable. For this, I would like to thank all my friends and co-workers from the department for having provided me with help, inspiration and fun. I will never thank my family enough for their unconditional love. It took me almost thirty years for realising my family is the best and most authentic part of me. I would like to thank Angelika for not giving up on me. I would like to thank my friends, the family I chose, for always standing by my side without ever doubting our friendship. I would like to thank my grandfathers Vincenzo and Terzilio, my grandmother Elisa, my aunts Ines and Nunziata and my uncle Pasquale for protecting me from above. Lastly, I would like to thank the One above us all for always answering my prayers and giving me the strength to not throw in the towel.

Alessio Bucaioni
Västerås, January, 2018

List of Publications

Publications Included in this Doctoral Thesis¹

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.* IEEE Access (impact factor: 3.244). December, 2016.

Paper B – 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) (acceptance rate: 41%) co-located with the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS). Ottawa, Canada. September, 2015.

Paper C – Handling Uncertainty in Automatically Generated Implementation Models in the Automotive Domain. *Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Saad Mubeen, Alfonso Pierantonio, Mikael Sjödin.* 42nd Euromicro Conference Series on Software Engineering and Advanced Application (SEAA) (acceptance rate: 36%). Limassol, Cyprus. September, 2016.

Paper D – Technology-preserving Transition from Single-core to Multi-core in Modelling Vehicular Systems. *Alessio Bucaioni, Saad Mubeen, Federico Ciccozzi, Antonio Cicchetti, Mikael Sjödin.* 13th European Conference on Modelling Foundations and Applications (ECMFA) (acceptance rate: 38%). Marburg, Germany. July, 2017.

¹The included publications are reformatted to comply with the doctoral thesis printing format

Paper E – A Model-based Approach for Vehicular Systems. *Alessio Bucaioni, Lorenzo Addazi, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, Mikael Sjödin.* MRTC Report MDH-MRTC-321/2017-1-SE. Västerås, Sweden. December, 2017. Submitted for journal publication.

Additional Publications not Included in the Thesis

Demonstrating Model- and Component-based Development of Vehicular Real-time Systems. *Alessio Bucaioni, Saad Mubeen, Mikael Sjödin, John Lundbäck, Mattias Gålnander, Kurt-Lennart Lundbäck.* Open Demo Session of Real-time Systems (RTSS@Work) at Real Time Systems Symposium (RTSS). Paris, France. December, 2017.

Modeling of Vehicular Distributed Embedded Systems: Transition from Single-core to Multi-core. *Saad Mubeen, Alessio Bucaioni.* 14th International Conference on Information Technology : New Generations (ITNG). Las Vegas, USA. April, 2017.

Early Timing Analysis of Vehicular Systems: the Road from Single-core to Multi-core. *Alessio Bucaioni.* Doctoral Symposium at the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (Models). Saint-Malo, France. October, 2016.

Provisioning of Deterministic and Non-deterministic Services for Vehicles: The Rubus Approach. *Harold Lawson, Saad Mubeen, Alessio Bucaioni, Jukka Mäki-Turja, John Lundbäck, Mattias Gålnander, Kurt-Lennart Lundbäck, Mikael Sjödin.* 4th International Workshop on Critical Automotive Applications: Robustness & Safety (CARS-2016). Gothenburg, Sweden. September, 2016.

Towards Design-Space Exploration of Component Chains in Vehicle Software. *Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Saad Mubeen, Alfonso Pierantonio, Mikael Sjödin.* Work in Progress at the 42nd Euromicro Conference Series on Software Engineering and Advanced Application (SEAA). Lymassol, Cyprus. September, 2016.

Raising Abstraction of Timing Analysis through Model-driven Engineering. *Alessio Bucaioni.* Licentiate Thesis. Västerås, Sweden. December, 2015.

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 (ICSS). New York, USA. August, 2015.

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.**

Exploring Timing Model Extractions at EAST-ADL Design-level Using Model Transformations. *Alessio Bucaioni, Saad Mubeen, Antonio Cicchetti, Mikael Sjödin*. 12th International Conference on Information Technology : New Generations (ITNG). Las Vegas, USA. April, 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, Italy. August, 2014.

From Modeling to Deployment of Component-based Vehicular Distributed Real-time Systems. *Alessio Bucaioni, Saad Mubeen, John Lundbäck, Kurt-Lennart Lundbäck, Jukka Mäki-Turja, Mikael Sjödin*. 11th International Conference on Information Technology : New Generations (ITNG). Las Vegas, USA. April, 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.

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, Italy. 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. Västerås, Sweden. September, 2013.

Contents

| | | |
|----------|--|-----------|
| I | Thesis | 1 |
| 1 | Introduction | 3 |
| 1.1 | Thesis Contribution | 5 |
| 1.2 | Thesis Outline | 5 |
| 2 | Preliminaries | 7 |
| 2.1 | Embedded Systems | 7 |
| 2.2 | Schedulability Analysis | 7 |
| 2.3 | Model Driven Engineering | 8 |
| 2.4 | EAST-ADL | 8 |
| 2.5 | Rubus Component Model | 9 |
| 2.6 | Uncertainty | 10 |
| 3 | Research Goal, Challenges and Contributions | 11 |
| 3.1 | Research Goal | 11 |
| 3.2 | Research Challenges | 12 |
| 3.3 | Research Contributions | 13 |
| 3.4 | Papers Contribution | 22 |
| 3.4.1 | Paper A | 23 |
| 3.4.2 | Paper B | 23 |
| 3.4.3 | Paper C | 24 |
| 3.4.4 | Paper D | 25 |
| 3.4.5 | Paper E | 26 |
| 4 | Research Methodology and Validation | 27 |
| 4.1 | Research Methodology | 27 |
| 4.2 | Validation | 28 |

| | | |
|-----------|--|-----------|
| 5 | Related Works | 31 |
| 6 | Conclusions and Future Works | 35 |
| | Bibliography | 37 |
| II | Included Papers | 43 |
| 7 | Paper A: | |
| | A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation from EAST-ADL | 45 |
| 7.1 | Introduction | 47 |
| 7.2 | Background and related work | 49 |
| 7.2.1 | MDE and CBSE in the Automotive Domain | 49 |
| 7.2.2 | End-to-end timing models and analyses | 52 |
| 7.2.3 | Paper contributions | 54 |
| 7.3 | Providing a metamodel for RCM | 55 |
| 7.4 | DL2RCM model transformation | 60 |
| 7.5 | Application to the steer-by-wire system | 67 |
| 7.6 | Evaluation and discussion | 69 |
| 7.7 | Conclusions and future work | 73 |
| | Bibliography | 75 |
| 8 | Paper B: | |
| | Anticipating Implementation-Level Timing Analysis for Driving Design-Level Decisions in EAST-ADL | 79 |
| 8.1 | Introduction | 81 |
| 8.2 | Related Work | 82 |
| 8.3 | A Running Example: the Steer-by-wire System | 83 |
| 8.4 | Applying the methodology | 84 |
| 8.4.1 | Transformation Phase | 85 |
| 8.4.2 | End-to-end Delay Analysis Phase | 89 |
| 8.4.3 | Filtering and Propagation Phases | 90 |
| 8.5 | Discussion | 91 |
| 8.6 | Conclusion | 92 |
| | Bibliography | 93 |

| | |
|---|------------|
| 9 Paper C: | |
| Handling Uncertainty in Automatically Generated Implementation Models in the Automotive Domain | 97 |
| 9.1 Introduction | 99 |
| 9.2 Background | 100 |
| 9.3 Motivating Scenario | 102 |
| 9.4 μ -Rubus | 105 |
| 9.5 Discussion | 109 |
| 9.6 Related Work | 111 |
| 9.7 Conclusion and Future Work | 114 |
| Bibliography | 115 |
| 10 Paper D: | |
| Technology-preserving transition from single-core to multi-core in modelling vehicular systems | 119 |
| 10.1 Introduction | 121 |
| 10.2 The Rubus Component Model | 123 |
| 10.3 Related Work | 124 |
| 10.4 Extending Rubus Component Model for Multi-core | 125 |
| 10.5 Modelling the Brake-by-wire System | 129 |
| 10.6 Lesson Learned | 133 |
| 10.7 Conclusion and Future Work | 134 |
| Bibliography | 137 |
| 11 Paper E: | |
| A Model-based Approach for Vehicular Systems | 143 |
| 11.1 Introduction | 145 |
| 11.2 Background | 147 |
| 11.2.1 EAST-ADL | 148 |
| 11.2.2 RCM | 150 |
| 11.2.3 Timing Analysis | 153 |
| 11.2.4 Paper Contributions in Relation with Authors' Previous Work | 155 |
| 11.3 The MoVES Methodology: why? | 156 |
| 11.4 MoVES for Timing | 158 |
| 11.4.1 FDA2RCM | 160 |
| 11.4.2 HDA2RCM | 163 |
| 11.4.3 MERGE and A2RCM | 165 |

| | |
|--|-----|
| 11.4.4 ALLOCATION | 168 |
| 11.4.5 BP | 169 |
| 11.5 Case Study | 171 |
| 11.6 Discussion and Validation | 179 |
| 11.7 Related Work | 183 |
| 11.7.1 Development of vehicular embedded systems | 183 |
| 11.7.2 Development of embedded systems | 184 |
| 11.7.3 Support for design-space exploration | 185 |
| 11.8 Conclusion and Future Work | 187 |
| Bibliography | 189 |

I

Thesis

Chapter 1

Introduction

In modern vehicles, more than 80% of innovation comes from the use of special purpose computers [1] which comprise of embedded software executing on embedded processors (embedded systems). Considering the growing complexity of embedded systems, their development cost and lead-time, effective software development methodologies is of paramount importance in the vehicular domain [1] [2]. Researchers and practitioners agreed that abstraction and automation, the founding pillars of Model Driven Engineering (MDE), could be game changers towards the achievement of such a goal [3]. MDE is an engineering paradigm which aims at improving the software development using models and model transformations. Models allow to focus on specific aspects of the software using concepts pertaining to the problem domain rather than constructs pertaining to the underlying technology [3]. Model transformations offer automation in the form of model manipulation (e.g., code-generation) [4]. In the last decade, MDE has been increasingly adopted in the vehicular domain bringing the introduction of several domain-specific modelling languages both from industry and academia. Currently, vehicular embedded software can be described by means of functional models from which execution models¹ are derived. Functional models provide a structured representation of the vehicle's functions in terms of software functions and interaction among them. Often, they are expressed by means of architectural languages such as the Electronics Architecture and Software Technology Architecture Description Language (EAST-ADL) [5]. Execution models enrich functional models with platform-

¹In the remainder of this thesis we refer to terms *execution models* and *implementation models* as synonyms.

and execution- information such as control flows and worst-case execution times and they are used as the base for verification of non functional properties such as timing. Generally, execution models are derived from functional models and expressed by means of domain-specific modelling languages or component models such as the AUTomotive Open System ARchitecture (AUTOSAR) [6] or the Rubus Component Model (RCM) [7]. However, existing approaches to support models integration in the development of vehicular embedded systems are still immature and the translation between functional and execution models is mainly performed manually. This lack of automation makes the translation of execution models cumbersome thus defers the verification of non functional properties to the last stages of the development process when modification on the software can be 40 times more expensive than the same modifications during the design of the software [8]. In this scenario, providing automation to support model integration would enable an early verification of the non functional properties of the vehicular embedded software. This would allow the engineer to take evidence-based decision during the design of the software when modifications generally require less effort and expense than the same modifications on an almost ready-to-deliver software.

In this doctoral thesis, we define a novel model-driven approach for vehicular embedded systems which supports the development and architectural exploration of system-designs with temporal awareness ensured by means of timing analysis. The proposed approach discloses the opportunity of improving the efficiency of the software development of vehicular embedded systems by replacing the need for late, expensive and time consuming software product modifications with early design changes, which are usually cheaper and faster. Starting from a functional model (expressed by means of EAST-ADL), model transformations generate a set of execution models (expressed by means of RCM). As there might be multiple ways to map elements between models, a source functional model can not be univocally translated into a single correspondent execution model [9]. While most of the current model transformations only consider one particular strategy out of the possible alternatives (of which developers have little or no control) [10], in the proposed approach, model transformations derive all the possible execution models entailing meaningful and unique configurations of modelling elements, from a timing perspective. We draw on existing schedulability analysis for evaluating the appropriateness of the generated execution models with respect to the specified timing requirements. Eventually, model transformations create a reference to the selected execution models along with their analysis results for enabling timing-aware design decisions. In order to ease the visualisation of

the selected execution models, we provide the engineer with a compact and intensional notation able to represent all of them by means of a single model. The proposed approach can be generally applied to non functional properties. However, we centred the approach on timing as it is one of the foremost concerns in the development of real-time systems as vehicular systems (let us think, for instance, to an untimely operation of an airbag or the anti-lock braking system which can cause the loss of lives). Moreover, timing-related issues are a perfect example of those usually discovered at late stages of the development and yet with a great impact on the system design.

1.1 Thesis Contribution

The main scientific contributions of this doctoral thesis are the following:

- A metamodel definition for RCM, called RubusMM, comprising modelling elements for representing software, timing constraints, occurrences and events, execution platform and software to hardware allocation.
- A mechanism for the automatic generation of the execution models, expressed using RubusMM, from a set of starting functional models and requirements, expressed using EAST-ADL.
-
- A mechanism that performs the analysis, selection and back-propagation of the RCM models which meet the specified set of timing requirements.
- A compact notation for visualising the set of the back-propagated RCM models by means of a single RCM model with uncertainty points.

1.2 Thesis Outline

The remainder of this thesis is organised as follows. Chapter 2 introduces the technical concepts used throughout the thesis. Chapter 3 describes the research goals, challenges and contributions of the thesis. Chapter 4 describes the research methodology and validation. Chapter 5 discusses the literature related to the work and contributions in this thesis. Chapter 6 draws conclusions and future directions. The second part of the thesis consists of Chapter 7 through Chapter 11 and describes the research contributions in terms of the included scientific publications.

Chapter 2

Preliminaries

In this section, we introduce the fundamental technical concepts used throughout this thesis.

2.1 Embedded Systems

An embedded system is a special-purpose computer system which is embedded in the system it controls [11]. Often, it interacts with its environment by means of sensors and actuators. Embedded systems are ubiquitous in electronic items, ranging from microwaves ovens to industrial process controllers. In modern vehicles, embedded systems replace or augment most of the vehicle's mechanical and hydraulic parts and implement many safety features, e.g., anti-lock braking system. Often, embedded systems have to meet real-time requirements as in the case of the collision avoidance systems. In this case, an embedded system is defined as real-time embedded system and it is expected to interact with its environment in a timely manner [12]. That is, its output is only acceptable when it is functionally correct and is delivered within the specified time.

2.2 Schedulability Analysis

Real-time embedded systems require evidences that their output will be delivered at the time that is suitable for the environment they interact with. Schedulability analysis is *a priori* timing analysis technique which provides evidence

on whether each function in the system is going to meet its timing requirements [13]. In this thesis, we leverage a mature schedulability analysis technique called end-to-end response-time and delay analysis [14]. The analysis calculates upper bounds on the end-to-end response times and delays of chains of tasks and messages in the system.

2.3 Model Driven Engineering

Model Driven Engineering is a software engineering paradigm which aims at raising the abstraction of the software development by shifting the focus from code to models [3]. To this end, MDE promotes models and model manipulations as first-class citizens in the development process. Models represent an abstraction of the system and help an expert to focus on system characteristics of interest, while hiding the others [3]. An example could be modelling functional behaviours, while hiding hardware-specific details. Valid models can be specified in accordance to the set of rules and constraints described by so-called metamodels [3]; valid models are said to conform to their respective metamodel. Within MDE, a software system can be developed by means of model manipulations. That is, abstract models are refined into more detailed models, until code is generated. Model manipulations are performed by means of model transformations. Automated model transformations are programs which automatically translate source models into target models while ensuring their conformance to their respective metamodels [4].

2.4 EAST-ADL

EAST-ADL is an architectural description language which captures the essentials of vehicular systems concerning their documentation, design, analysis and synthesis. EAST-ADL is composed of ten different packages, each of which addresses different aspects of these systems and their development. In this doctoral thesis, we leverage concepts from the structure, requirements and timing packages. The structure package provides for the specification of the software architecture in terms of basic elements and interactions among them. The structure package makes use of four abstraction levels which ensure separation of concerns through the development process. The abstraction levels are: vehicle, analysis, design and implementation. Such a separation is only conceptual and some modelling elements span over several abstraction levels. In this doctoral thesis, we specify the functional models by means of the functional

design architecture, hardware design architecture and allocation concepts from the EAST-ADL design level. The functional design architecture describes how the software functions interact. The hardware design architecture describes the physical architecture of the vehicular embedded system. The allocation describes the mappings between the elements of the functional design architecture and the hardware design architecture. The timing package provides for the modelling of the timing constraints stemming from the non functional requirements. In this doctoral thesis, we use the elements from the timing package for the specification of the timing events, occurrences and constraints within the functional models. The requirement package provides the means for describing the properties of a vehicular embedded system and their verification. In this doctoral thesis we make use of elements from the requirements package for the back propagation of the generated execution models and their schedulability analysis results to the related EAST-ADL model.

2.5 Rubus Component Model

Rubus Component Model is a modelling language for the predictable development of resource-constrained embedded real-time systems developed by Arcticus Systems AB¹ in collaboration with Mälardalen University. Currently, it is used by several OEM, Tier-1 and Tier-2 companies in the vehicular domain (e.g., Volvo Construction Equipment, BAE Systems Hagglands, Hoerbiger and Knorr Bremse) as the modelling language for representing execution models and in cooperation with architectural languages such as EAST-ADL. Currently, RCM supports the modelling of software architecture, execution platform, allocation information and timing properties of vehicular embedded systems [15].

Within RCM, the embedded software architecture is modelled by means of software circuit (SWC) elements and interactions among them. A SWC encapsulates a software function. SWCs can be grouped in composite elements called Assemblies. As the main goal of RCM is to support the predictable development of vehicular embedded systems, timing properties and constraints are pivotal in the language and they can be specified at different hierarchical levels. Within RCM, the execution platform is modelled by means of node, core and partition elements. Allocation information can be specified among any two elements of the software architecture and execution platform. In this doctoral thesis, we provide a canonical metamodel definition for RCM,

¹<https://www.arcticus-systems.com>

namely RubusMM, as part of our research contribution. Moreover, we employ RubusMM for the specification of execution models.

2.6 Uncertainty

In software engineering, uncertainty is a meta-property caused by the lack of knowledge or unresolved decisions [16]. In this thesis, we adopt a language-centric approach for managing uncertainty, i.e., multiple models, which is able to generate at once and represent the entire solution space of the generated models in the intensive form of a model with uncertainty [9].

Chapter 3

Research Goal, Challenges and Contributions

This chapter discusses research goal, research challenges and research contributions.

3.1 Research Goal

Timing verification is essential and unavoidable for the development of real-time embedded systems such as vehicular embedded systems. However, researches show that timing verification is more efficient when it is performed earlier during the development process as modifications during the last stages of the development can be 40 times more expensive than the same modifications during the design of the software [8]. To this end, we believe that enabling timing verification at design level, by means of integration through model transformations, can improve the efficiency of the software development of vehicular embedded systems. In fact, timing verification results could be used for driving the design process and replacing the need for late, expensive and time-consuming software modifications with earlier design modifications, which are usually cheaper and faster. The overall goal of this research work is to improve the efficiency of the software development of vehicular embedded systems by supporting the development and architectural exploration of system-designs with temporal awareness. More precisely, we aim at providing automation for the generation of execution models, expressed by means

of RubusMM, starting from functional models, expressed by means of EAST-ADL. In addition, we aim at providing an automatic support for the selection of the generated RCM models that meet the specified timing requirements as well as for their back-propagation and visualisation at design level.

3.2 Research Challenges

Starting from the research goal, we derived the following research challenges (RCs) and used them as main drivers for the research work presented in this doctoral thesis.

RC 1. Definition of a metamodel for RCM. Currently, vehicular embedded software can be described through various modelling languages such as EAST-ADL and RCM. Consequently, MDE seems a natural choice for enabling the automatic integration among the languages. Metamodels and model transformations are the founding pillars of MDE and they serve for regulating the specification of models and for automating their manipulations, respectively. Therefore, in order to enable a full-fledged MDE approach, it is paramount to provide a metamodel definition to all the languages involved in the software development of vehicular embedded systems. Before this research effort, RCM did not have a canonical metamodel specification, but it rather relied on a textual language specification.

The challenge is the definition of a metamodel for RCM comprising modelling elements for representing software and the execution platform architectures, the timing constraints, occurrences and events of the vehicular embedded system and the software to hardware allocation. In particular, the metamodel should be defined bearing in mind backward compatibility with legacy RCM artefacts and should not entail any modification to the Rubus run-time layer.

RC 2. Definition of a mapping between EAST-ADL and RCM metamodels. Timing verification is crucial task in the development of vehicular embedded systems. However, it gives meaningful results only when applied on execution models as functional models do not entail detailed, e.g., timing, control and allocation information. One way to leverage timing verification results at design level, is the definition of an automatic and transparent process for the generation of RCM models from EAST-ADL models. However, due to the different levels of abstraction between EAST-ADL and RCM, there might be multiple ways to generate RCM from EAST-ADL models.

In this context, the challenge is two-fold. On the one hand, we need to define an automatic process able to generate the RCM models containing all the needed software architecture, timing, control and allocation information. On the other hand, this process should be able to generate all the possible RCM models entailing meaningful and unique timing as well as allocation configurations as opposed to considering only one particular generation strategy [9] [10].

RC 3. Definition of a mechanism for unveiling the feasible RCM models at design level. Once the RCM models have been generated and the schedulability analysis performed, the RCM models satisfying the specified timing requirements must be unveiled at design level for enabling timing-aware design decision. Here the challenge is two-fold. On the one hand, the generated RCM models must be compared against the specified timing requirements and back-propagated at design level. On the other hand, it is crucial that all the RCM models satisfying the specified timing requirements are back-propagated at design level and represented in a convenient notation, which highlights the models' commonalities and differences for aiding possible manual investigations. In fact, at this point, the selection can not be automated and it can only be made by manually investigating the set of selected RCM models considering perhaps additional non functional properties.

3.3 Research Contributions

Early verification of non functional requirements can positively affect the efficiency of the development of vehicular real-time embedded systems. Currently, early verification of non functional requirements is hard to achieve due to the lack of automation supporting models integration and analysis. For instance, let us consider a typical development process as described by the flowchart in Figure 3.1. In this setting, as meaningful non functional analysis (such as timing) must be run on execution models, the engineer is required to create one manually. The non functional analysis of interest is run on the manually created model and the result is verified against the given set of requirements. If the specified non functional requirements are not met, the engineer is has to iterate the process, modify or create a new execution model until a compliant one is found. Since the process of creating and verifying execution models is expensive, it is not leveraged early in the development process for having quick and early feedback on the design level models. To boost early verification, in this thesis we propose a novel model-driven approach for the devel-

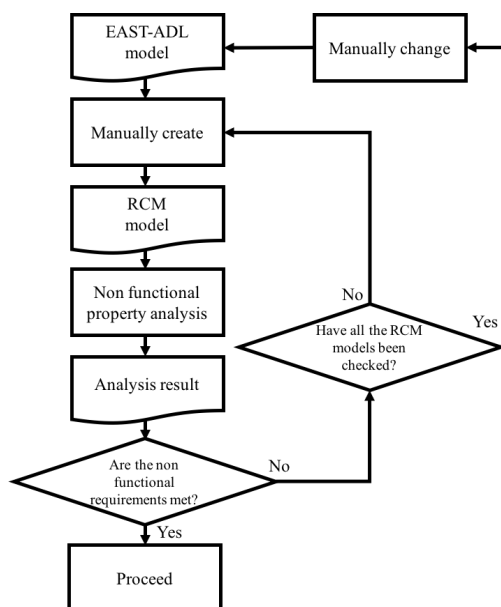


Figure 3.1: Development process without the proposed approach

opment of vehicular real-time embedded systems supporting early verification of non functional properties. Let us consider a development process equipped with the proposed approach as described by the flowchart in Figure 3.2. In this setting, all meaningful execution models are automatically generated from the design model and analysed by means of model transformations, at once. Considering a set of non functional requirements, model transformations are responsible for the selection and back propagation of the best execution model (or set of models), too. Besides relieving the engineer from the manual definition of execution models, the proposed approach enables early verification at design level. In addition, while in the manual process several iterations may be needed to reach a compliant execution model, the proposed approach generates all meaningful execution models and identifies the best one(s) automatically in one single iteration.

As timing requirements are crucial for our domain of interest and timing-related issues are typical problems arising very late in the development process, in this thesis we center the proposed approach on timing. In particular, the

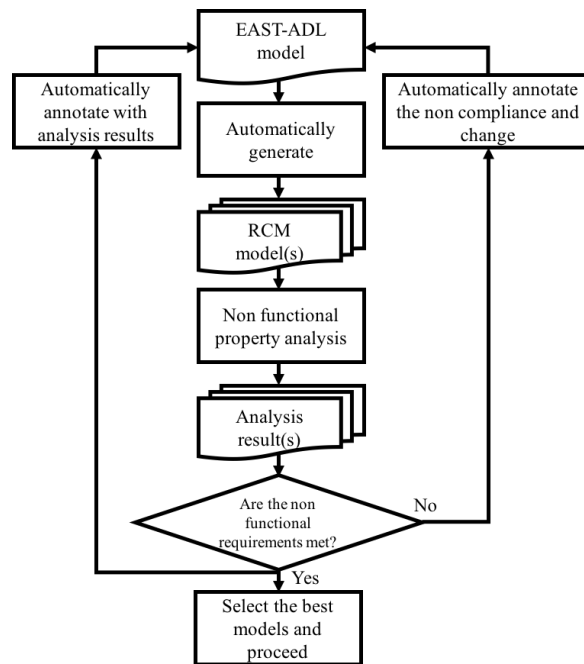


Figure 3.2: Development process equipped with the proposed approach

main contribution of this doctoral thesis is a model-driven approach supporting the development and architectural exploration of system-designs with temporal awareness ensured by means of schedulability analysis. Figure 3.3 provides a graphical representation of the proposed approach.

The proposed approach leverages the interplay of EAST-ADL and RCM as the modelling languages for expressing functional and execution models, respectively, and a suite of 6 model transformations. The first step of the proposed approach is the automatic generation of RCM models representing the software architecture and its timing properties and constraints from an EAST-ADL functional design architecture equipped with EAST-ADL timing constraint modelling elements. Such a generation is entrusted to the FDA2RCM model transformation. As there could be multiple ways of generating RCM models from an EAST-ADL functional design architecture, the FDA2RCM model transformation generates, in a single execution, all the RCM models

entailing unique timing and control flow information. The second step of the proposed approach is the automatic generation of an RCM model representing the execution platform from an EAST-ADL hardware design architecture and it is performed by the HDA2RCM model transformation. At this point, as RCM describes the execution platform at a different level of abstraction compared to EAST-ADL, manual refinements of the generated RCM model representing the execution platform may be needed in order to, e.g., specify cores and partitions in the case of vehicular embedded systems for multi-core. This is a necessary step as detailed execution platform models are pivotal for the specification of the software allocation information which, in turn, affects schedulability analysis. The next step of the proposed approach merges the generated RCM software and execution platform models to obtain a set of complete RCM models where the software allocation information can be specified. This step is performed by the MERGE model transformation. The specification of the allocation information on the merged RCM models is entrusted to two model transformations, namely A2RCM and ALLOCATION. The former is responsible for translating the allocation information from the EAST-ADL allocation model. The latter is responsible for generating RCM models entailing those allocation configurations that can not be directly derived from the EAST-ADL allocation model as in the case of, e.g., allocation of software to core and partition elements. As there could be multiple unique allocation configurations, the ALLOCATION model transformation generates, in a single execution, all the RCM models entailing unique allocation information. At this point, schedulability analysis is run on the generated RCM models. If none satisfies the set of specified timing requirements, the engineer is notified about the inability of the initial EAST-ADL model to satisfy its timing requirements. Otherwise, the RCM models satisfying the specified timing requirements are propagated back, together with their analysis results, at the design level by the BP model transformation and visualised as a single RCM model with uncertainty. Figure 3.3 provides a breakdown of the main contribution in specific research contributions (RCOs) while Table 3.1 shows the relation between them and the RCs.

RCO 1 - RubusMM. This contribution, marked as 1 in Figure 3.3, provides a metamodel definition for RCM as a needed step for enabling integration through model transformations. In fact, RCM was originally thought for providing modelling purposes, but it did not feature any model driven mechanism, i.e. automation in terms of model transformation. RubusMM has been defined through a two-step process. In the first step, we reverse-engineered the RCM specification with the aim of restoring the separation of concerns lost

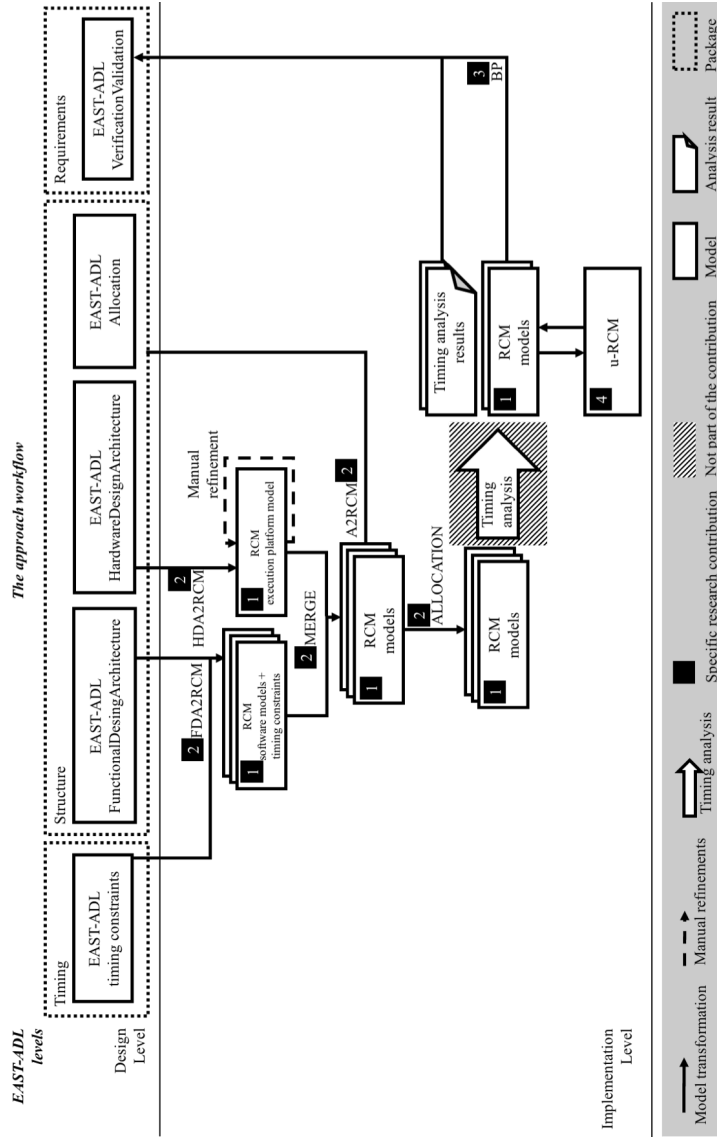


Figure 3.3: Research Contributions

during the evolution of the component model. As a side effect, this allowed us to maximise backward compatibility with legacy RCM artefacts. This activity resulted in the addition of modelling elements such as connectors and threads as well as in the refinement of hierarchical structures. In the second step, we extended RubusMM for the modelling of vehicular embedded systems on single- and multi-core platforms. This extension includes modelling elements for representing the execution platform and the software to hardware allocation information. It is important to note that the extension does not affect backward-compatibility as it does not modify any hierarchical structure. Currently, RubusMM is defined as an Ecore model, within the Eclipse Modeling Framework¹ (EMF), and comprises modelling elements for representing i) the software architecture and timing constraints, occurrences and events, ii) the execution platform and iii) the software to hardware allocation information.

This contribution provides a solution to RC 1. Paper A presents the reverse-engineered version of RubusMM while Paper D presents the extended RubusMM for single- and multi-core.

RCO 2 - Mechanism for the automatic generation of execution models.

This contribution, marked with 2 in Figure 3.3, provides an automatic mechanism for the generation of RCM models from EAST-ADL models. This is fundamental for enabling timing-aware design decision by integration through model transformation.

This contribution comprises a set of five model transformations namely FDA2RCM, HDA2RCM, MERGE, A2RCM and ALLOCATION. The contribution brought by them is two-fold. On the one hand, they provide automatic generation of RCM models, which are the input for schedulability analysis. On the other hand, they provide generation of all the meaningful RCM models from an initial EAST-ADL model.

The FDA2RCM transformation provides for the generation of RCM models representing the software architecture and its timing constraints from an EAST-ADL functional design architecture equipped with EAST-ADL timing constraints. In a nutshell, it translates the elements of the EAST-ADL functional design architecture to RCM software elements. Additionally, it provides automatic generation of all control flow and timing elements in the RCM models. Since such a translation can produce multiple RCM models entailing unique configurations of control flow and timing elements, FDA2RCM generates, in a single execution, all of them. This is possible thanks to a bidirectional

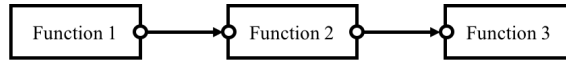
¹<http://www.eclipse.org>

model transformation language, namely the Janus Transformation Language (JTL) [17]. JTL is a constraint-based bidirectional model transformation language specifically tailored to support one-to-many model transformations by generating all the possible models, at once. JTL adopts a Query/View/Transformation (QVT) relation-like syntax [18] and relies on the Answer Set Programming (ASP) [19], 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 models that are consistent with the transformation rules. For instance, the application of the FDA2RCM transformation to the simplified EAST-ADL functional design architecture depicted in Figure 3.4a produces the four simplified RCM models depicted in Figure 3.4a .

It is worth to mention that JTL supports the specification of logic constraints, which can be used for narrowing the number of generated models and tailoring their generation for specific purposes. In the specific case of the FDA2RCM model transformation, we employed logic constraints for generating RCM models entailing valid configurations of control flow and timing elements, only. For instance, in the case of the simplified EAST-ADL functional design architecture in Figure 3.4a, the logic constraints prevent the generation of the four RCM models where software function *Function 1* was not triggered by an independent clock.

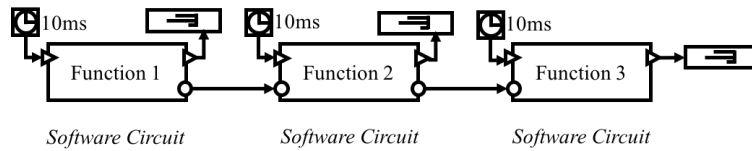
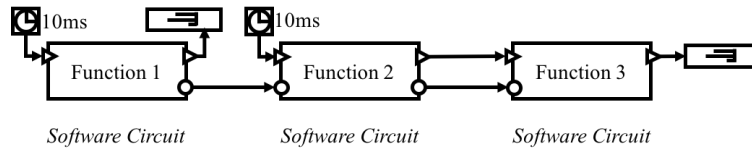
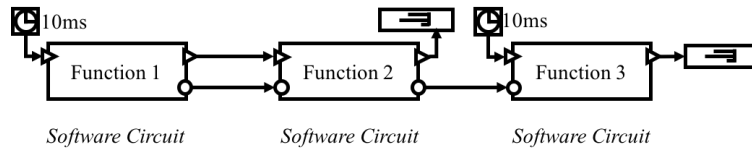
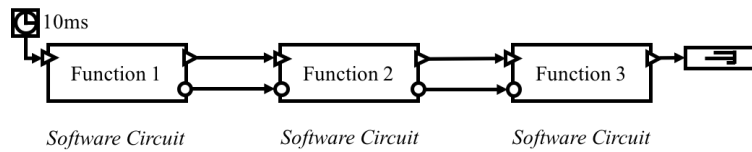
The HDA2RCM transformation provides generation of the RCM model representing the execution platform from an EAST-ADL hardware design architecture. It is implemented by means of JTL and translates the EAST-ADL node, connectors and port elements into corresponding elements in RCM. Moreover, in order to conform to the RCM hierarchy of execution platform elements, for each generated RCM Node element, a Core and a Partition element are created, too. In fact, compared to EAST-ADL, RCM describes the execution platform at a different level of abstraction by using core and partition concepts. Please note that, the engineer can still manually refine the generated RCM execution platform model by using the RCM *Core* and *Partition* elements.

The MERGE transformation merges the generated RCM models representing software architecture and execution platform for allowing the translation of the allocation information from EAST-ADL. MERGE is implemented as a QVT Operational (QVT-O) transformation which performs a weaving of the RCM models, where the modelling elements of the RCM execution platform model are linked to the *System* element of the RCM software model through its *Node* reference. The translation of the allocation information is entrusted to the A2RCM transformation. A2RCM is an in-place transformation writ-



DesignFunctionPrototype DesignFunctionPrototype DesignFunctionPrototype

(a) Example of a Functional Model



(b) 4 of the 8 RCM Execution Models for the EAST-ADL Model in Figure 3.4a

ten in QVT-O and that sets the reference *isAllocated* of the RCM Allocatable elements starting from the allocation information expressed by means of the EAST-ADL Function Allocation elements.

Due to the different level of abstraction between RCM and EAST-ADL, complete allocation information for the RCM models can not be directly derived from an EAST-ADL Allocation. In this context, the ALLOCATION transformation provides automation means for the generation of the allocation information in the RCM models when a direct translation from EAST-ADL is

not possible. The engineer is required to set which software elements must be allocated to which execution platform elements. (e.g., Assembly to Core, Assembly to Partition, SWC to Core). Based on the engineer choice, the ALLOCATION transformation automatically generates, in a single execution, all the RCM models which entail unique allocation configurations of RCM Allocatable to Allocator elements. Similar to FDA2RCM, this is implemented as a JTL model transformation. It is worth to note that logic constraints can be applied for reducing the number of generated RCM models when, e.g., allocation information is already available.

This contribution provides a solution to RC 2. Paper B provides an initial version of this automation mechanism consisting of the FDA2RCM transformation only. Paper E discusses this contribution in its complete version.

RCO 3 - Back-propagation of analysis results to design level. This contribution, marked as 3 in Figure 3.3, enables the selection of the generated RCM models satisfying the specified timing requirements and their back-propagation to design level. This represents the last step in the process of enabling timing-aware design decisions.

The contribution is embodied by BP, an in-place model transformation, which takes as input the generated RCM models, their schedulability analysis results and the set of specified timing requirements. First, BP compares the analysis results with the specified timing requirements and discards those not fulfilling the requirements. Afterwards, it adds to the initial EAST-ADL model the elements from the requirements package for the validation of the software. Finally, it enriches the added elements with the references to the folders containing the selected RCM models and their analysis results. Currently, BP is defined as a QVT-O transformation.

This contribution provides a solution to RC 3. Paper B and Paper E discuss the initial and enhanced version of this contribution, respectively.

RCO 4 - Compact visualisation of multiple Rubus models. Multiple RCM models can be selected and back propagated to design level and no further selection can be automated as all selected RCM models have equally good schedulability analysis results. This contribution, marked as 4 in Figure 3.3, provides a mechanism for the compact representation of all these equally good RCM models in terms of their commonalities and distinctions by means of a single RCM model with uncertainty points. The intent is to allow the engineer to deal with the set of selected RCM models as if they were a single model and enable further selection based on, e.g., architectural choices or other relevant

non functional properties. Such a representation is achieved by employing u-RubusMM, which is a revised version of RubusMM endowed with uncertainty elements. This is done by employing the metamodel-independent technique presented in [9]. More precisely, an automated model transformation defined in u-JTL [9] is responsible for the generation of u-RubusMM starting from RubusMM.

This contribution provides a solution to RC 3. Paper C provides further details about u-RubusMM.

Table 3.1 shows the relation between RCOs and RCs.

| | | Research Challenges | | |
|------------------------|-------|---------------------|------|------|
| | | RC 1 | RC 2 | RC 3 |
| Research Contributions | RCO 1 | X | X | |
| | RCO 2 | | X | X |
| | RCO 3 | | | X |
| | RCO 4 | | | X |

Table 3.1: Research Contributions in Relation to the Research Challenges

3.4 Papers Contribution

This section lists the papers included in the thesis and shows the relations between them and the RCOs, as discussed in Section 3.3, in Table 3.2 .

| | | Research Contributions | | | |
|--------|---|------------------------|-------|-------|-------|
| | | RCO 1 | RCO 2 | RCO 3 | RCO 4 |
| Papers | A | X | | | |
| | B | | X | X | |
| | C | | X | | X |
| | D | X | | | |
| | E | | X | X | X |

Table 3.2: Included papers in Relation to the Research Contributions

3.4.1 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.* IEEE Access (impact factor: 3.244). December, 2016.

Abstract –According to the Model-Driven Engineering paradigm, one of the entry requirements when realising 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 a metamodel definition for the Rubus Component Model, an industrial solution used 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 different modelling levels can be automated, we present a model-to-model transformation between models conforming to EAST-ADL and models described by means of the Rubus Component Model. To validate our solution, we exploit a set of industrial automotive applications to show the applicability of both the Rubus Component Model metamodel and the model transformation.

Status. Published.

Personal Contribution. The research work presented in this paper was done in collaboration with all the authors. However, I was the main contributor and driver. More specifically, I i) reverse-engineered the RCM language , ii) provided a canonical metamodel definition to RCM, called RubusMM and iii) extended RubusMM with the modelling elements for the integration with EAST-ADL.

3.4.2 Paper B

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) (acceptance rate: 41%) co-located with the 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.

Status. Published.

Personal Contribution. The research work presented in this paper was done in collaboration with all the authors. However, I was the main contributor and driver. More specifically, I i) defined the methodology, ii) implemented its composing tasks and iii) applied the solution to the running example.

3.4.3 Paper C

Handling Uncertainty in Automatically Generated Implementation Models in the Automotive Domain. *Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Saad Mubeen, Alfonso Pierantonio, Mikael Sjödin.* 42nd Euromicro Conference Series on Software Engineering and Advanced Application (SEAA) (acceptance rate: 36%). Limassol, Cyprus. September, 2016.

Abstract – Models and model transformations, the two core constituents of Model-Driven Engineering, aid in software development by automating, thus taming, errorprone of tedious engineering activities. In many cases, the result of these automated activities is an overwhelming amount of information. This is the case of one-to-many model transformations that, e.g. in model-based design-space exploration, can potentially generate a massive amount of

candidate models (i.e., solution space) from one single source model. In our scenario, from one design model we generate a set of possible implementation models on which timing analysis is run. The aim is to find the best model from a timing perspective. However, multiple implementation models can have equally good analysis results. Therefore, the engineer is expected to investigate the solution space for making a final decision, using criteria which fall outside the analysis' criteria themselves. Since candidate models can be many and very similar to each other, manually finding differences and commonalities is an impractical and error-prone task. In order to provide the engineer with an expressive representation of models' commonalities and differences, we propose the use of modelling with uncertainty. We achieve this by elevating the solution space to a first-class status, adopting a compact notation capable of representing the solution space by means of a single model with uncertainty. Commonalities and differences are thus represented by means of uncertainty points for the engineer to easily grasp them and consistently make her decision without manually inspecting each model individually.

Status. Published.

Personal Contribution. The research work presented in this paper was done in collaboration with all the authors. However, I was the main contributor and driver. More specifically, I i) provided RubusMM with the uncertainty notation and ii) applied the solution to the running example.

3.4.4 Paper D

Technology-preserving transition from single-core to multi-core in modelling vehicular systems. *Alessio Bucaioni, Saad Mubeen, Federico Ciccozzi, Antonio Cicchetti, Mikael Sjödin.* 13st European Conference on Modelling Foundations and Applications (ECMFA) (acceptance rate: 38%). Marburg, Germany. July, 2017.

Abstract – The vehicular industry has exploited model-based engineering for design, analysis and develop of single-core vehicular systems. Next generation of autonomous vehicles will require higher computational power, which can only be provided by multi-core platforms. Current model-based solutions and related modelling languages, originally conceived for single-core, can not effectively deal with multi-core specific challenges, such as core-interdependency and allocation of software to hardware. In this paper, we propose an extension

to the Rubus Component Model, core of the Rubus model-based approach, for the modelling, analysis and development of vehicular systems on multi-core. Our goal is to provide a lightweight transition of a model-based approach from single-core to multi-core, without disrupting the current technological assets in the vehicular domain.

Status. Published.

Personal Contribution. The research work presented in this paper was done in collaboration with all the authors. However, I was the main contributor and driver. More specifically, I i) extended RubusMM with the modelling elements for representing the execution platform and the software to hardware allocation information and ii) conducted the running example.

3.4.5 Paper E

A Model-based Approach for Vehicular Systems. *Alessio Bucaioni, Lorenzo Addazi, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, Mikael Sjödin.* MRTC Report MDH-MRTC-321/2017-1-SE. Västerås, Sweden. December, 2017. Submitted for journal publication.

Abstract – This paper introduces a novel model-based approach for the software development of vehicular embedded systems. The proposed approach discloses the opportunity of improving efficiency of the development process by providing support to identify viable design solutions with respect to selected non functional requirements. To this end, it leverages the interplay of two modelling languages for the vehicular domain whose integration is achieved by a suite of model transformations. An instantiation of the methodology is discussed for timing requirements, which are among the most critical ones for the development of vehicular systems. The applicability of the methodology is demonstrated as proof of concepts on industrial use cases performed in cooperation with our industrial partners.

Status. Under review.

Personal Contribution. The research work presented in this paper was done in collaboration with all the authors. However, I was the main contributor and driver. More specifically, I i) defined the methodology and iii) applied the solution to the running example.

Chapter 4

Research Methodology and Validation

This chapter discusses research methodology and validation.

4.1 Research Methodology

Collaborative research between industry and academia is a great example of how research in software engineering is often stimulated by problems arising from the use of software in the real life [20]. In this respect, the research presented in this thesis was conducted in a partnership between Mälardalen University and Arcticus Systems with the collaboration of Volvo Construction Equipment, Saab Avionics Systems and BAE Systems. For this research, we adopted a methodology being an adaptation of the model for technology transfer described in [21]. Figure 4.1 gives a graphical representation of the adopted research methodology. We began by assessing the state-of-the-art, the state-of-the-practice and the industrial demands with the aim of defining a research goal. During these stage, we identified several research challenges connected to the main research goal. For each elicited challenge, we investigated the state-of-the-art and practice with the aim of identifying a possible solution, if none existed. After performing the investigation, we defined a candidate solution. In this stage, the industrial partners played a crucial role as they provided early and quick feedbacks ensuring that the candidate solution was realistic and could fit current practices and industrial needs. The validation the of each

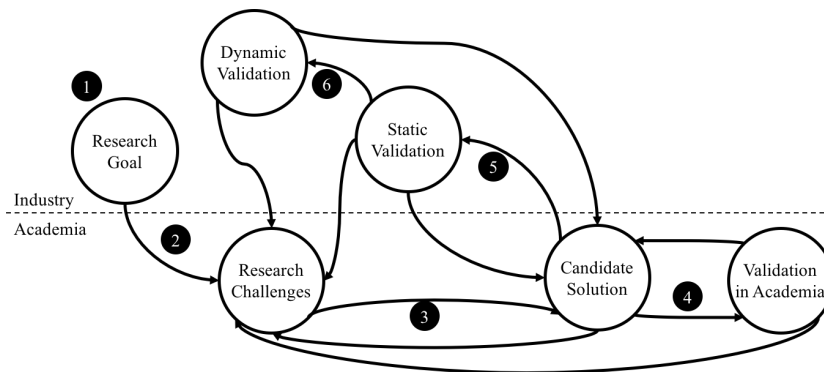


Figure 4.1: Research Methodology

candidate solution required three steps. During the academic validation, each solution was evaluated in the university by means of case study performed by researchers. Eventually, we used the finding acquired during the academic validation for refining the existing solutions or defining new research challenges. For instance, this was the case of RubusMM, whose definition described in Paper A was refined in the definition given in Paper D. During the static validation, we presented the candidate solutions to the industrial partners in a series of dedicated meetings and workshops. The aim of this step was to collect feedbacks regarding the usability and scalability of each solution. The feedbacks acquired during the static validation were used for refining the existing solutions or defining new research challenges as in the case of the visualisation mechanism described in RCO 4. In fact, the challenge of having an intensional and convenient notation for representing a multitude of models as one model with uncertainty arose only when the generation mechanism described in RCO 2 was able to generate a set of RCM models. Eventually, we performed the dynamic validation by means of industrial projects and experiments.

4.2 Validation

The work presented in this doctoral thesis and its contributions have been evaluated progressively as prescribed by the research methodology in Section 4.1.

With respect to RubusMM, we verified its consistency, expressiveness and applicability against several industrial system designs such the Brake-By-Wire

(BBW) [15], Steer-By-Wire (SBW) [22], Intelligent Parking Assist (IPA) [23], etc. Moreover, the industrial partners played a key role in providing feedbacks regarding the its industrial relevance.

Apart from RubusMM, the remaining contributions are implemented by means of model transformations. In this respect, the three validation steps described in the research methodology helped us in discussing some interesting properties of the model transformations, such as syntactic and semantic correctness, complexity, termination and performance [24]. With syntactic correctness, we refer to the ability of a transformation to produce valid target models when executed on valid source models [24]. Such a property holds for the transformations presented in this thesis and we demonstrated it by means of the case studies done during the academic and dynamic validation. With the term semantic correctness, we refer to the ability of a transformation to produce semantically valid target models [24]. Such a property holds for the transformations presented in this thesis and one way we entrusted it was to define the transformations by means of a precise and finite set of rules mapping EAST-ADL to RCM elements without altering, violating or colliding the structural hierarchies of the languages. Moreover, the semantic of the generated RCM models was validated by the practitioners during the static validation and by the leveraged schedulability analysis. We considered two dimensions for the transformations complexity which are the intricacy and the number of the generated RCM models. During the static validation, we conveyed that the generated RCM models have equal complexity of manually defined ones. Although some of the model transformations can theoretically produce multiple RCM models, during the academic and dynamic validation we were able to demonstrate that the transformations always terminate¹ in few seconds and produce only a limited number of RCM models. Moreover, the transformations could be refined by the engineer on the basis of the specific system and the solution space can be reduced by adding constraints that operate on the possible mapping policies.

We believe that the automation introduced by the proposed approach discloses the opportunity to improve the efficiency of the software development process by means reduced need for late modifications on the software. In particular, model transformations allow to cut the development time while ensuring the compliance with the non functional requirements of the vehicular embedded software. Without the proposed approach, in fact, the development would progress incrementally with team of engineers manually defining exe-

¹Please note that, providing a formal proof on the transformations' termination is outside the scope of this thesis.

cution models until a suitable one, from a non functional perspective, is found. On the contrary, with the proposed approach, the execution models are automatically generated and non functional requirements verified at once allowing the engineers to focus and reason only on the compliant models. By enabling early verification, the proposed approach discloses the opportunity of reducing late modifications on the vehicular software, which empirical studies showed to be generally more expensive than modifications during the design of the software [8]. In fact, by adopting the proposed approach, the engineer is either notified on the non compliance of the starting EAST-ADL model to the set of the considered non functional requirements or notified with the set of the compliant RCM models with which proceed for the development. In the former scenario, late modifications are prevented while in the latter they are not needed.

In this thesis, given the importance of timing properties during the design and development of vehicular real-time embedded systems, we centred the proposed approach on timing. However, we recognise that further non functional properties such as memory usage, energy efficiency, and so forth, play an important role during the development of these systems. In this respect, it is worth to note that the proposed approach proposed can be instantiated to consider further properties, as long as they are measurable and comparable at the EAST-ADL and RCM levels of detail. Additionally, other properties can be exploited for selecting multiple RCM models having equally good timing performance or can be considered from the initial stages of the generation process of the possible solutions. In both cases, the proposed approach would need to be extended only in terms of specific model transformations for the generation of the related non functional properties of interest.

Chapter 5

Related Works

This doctoral thesis deals with the research problem of supporting a timing-aware model driven development of distributed vehicular embedded systems on single- or multi-core platforms. Hereafter, a number of relevant similar approaches are discussed.

AUTOSAR [6] is an industrial initiative to provide a standardised software architecture for the development of vehicular embedded systems. In the TIMMO and TIMMO2USE projects, AUTOSAR was provided with a timing model [25]. However, the AUTOSAR timing model does not distinguish between the control and the data flows at the application software level, a distinction that is fundamental for providing early timing verification [26]. A framework to specify the end-to-end timing constraints and analyse the corresponding end-to-end delays [27, 28, 29] was proposed in the aforementioned projects too.

There are some works [30] [31] that aim at supporting end-to-end timing analysis of the higher-level abstraction models. These works heavily rely on the availability of legacy models and favour the bottom-up development approach when the existing systems are extended. Unlike our methodology, these works are not applicable to the top-down development approach. Another work [26] refines timing requirements between the design and implementation levels to support the timing analysis at the implementation level. Unlike our methodology, this work performs manual translation of the software architecture from the design level to the implementation level. However, this work can be used complementary to our work by running the timing analysis of the implementation level models that are automatically generated by our methodology.

Often, AUTOSAR is used in cooperation with EAST-ADL as a modelling language for the implementation level. Even if EAST-ADL entails abstraction and separation-of-concerns, there is no specific automation support for interconnecting the abstraction levels in the structure package, e.g., between design and implementation levels. As a consequence, schedulability analysis and their results have to be tackled and tracked back manually by the engineer due to the abstraction gaps between the different levels. These tasks can be time-consuming and error-prone, especially when considering the complexity of modern vehicular systems [32]. On the contrary, in this doctoral thesis we propose to leverage automation through model transformations to keep the consistency between the different abstraction levels. The abstraction gaps naturally introduce multiple choices, which are managed by an appropriate transformation language (JTL). Moreover, AUTOSAR does not provide means for modelling the execution platform [33].

CHESS is a cross-domain framework for the design of component-based embedded systems, including vehicular systems [34]. CHESS features a specific UML profile obtained as a combination of different languages, e.g., MARTE and SysML. The framework provides modelling of embedded software for early analysis, such as dependability and schedulability, as well as for code generation, monitoring and back-propagation. Currently, CHESS does not support the design space generation and does not provide means for representing uncertainty in the development process.

Vehicular embedded systems, especially when considering autonomous driving and networks of vehicles, are often referred to as cyber-physical systems (CPS) [35]. In the recent years, several approaches dealing with CPS development by adopting multi-paradigm modelling techniques and leveraging simulation mechanisms to perform early analysis of systems have been proposed [36, 37]. Although the work presented in this doctoral thesis does not exploit simulation techniques, it does not prevent the use of simulation mechanisms to analyse and select the generated design alternatives with respect to non functional properties of interest.

Given the ubiquity of software, there exists a corpus of literature devoted to the design of embedded systems and posing a special focus to non functional requirements. In this respect, several works are based on the use of general-purpose languages such as UML and the UML profile for MARTE [38] as alternatives to domain-specific languages like, e.g., AUTOSAR and RCM. GASPARD is a MARTE-based framework for the design of parallel embedded systems [39]. It prescribes a workflow made-up of subsequent analyses and refinement steps, from higher to lower abstraction levels. Similar to the pro-

posed approach, the analysis at lower levels are meant to produce feedback for the design activities and the process is automatised by means of model transformations. Compared to our approach, GASPARD seems to tackle different non functional aspects which can be complementary to timing. Moreover, GASPARD does not provide for the compact representation of the design space. MARTE is adopted also in [40] to design the high-level architecture of the software system and for the code generation. It uses UML for modelling the software components and MARTE for modelling the hardware and the software to hardware allocations. One major difference with respect the proposed approach is that timing requirements are verified by means of simulations on the generated code and not starting from functional models. In [41], the authors propose a technique to specify tasks and their allocation to cores. The technique is based on MARTE and allows to perform simulation and task allocation optimisation based on the execution whereas the approach proposed in this doctoral thesis statically generates all the possible allocation configurations.

Chapter 6

Conclusions and Future Works

The work presented in this thesis describes a timing-aware model-driven approach for the software development of vehicular embedded systems. In particular, it tackles the problem of guiding the engineer in taking timing-aware decisions at design level when modifications are generally less expensive than modifications at later stages of the development. The overall contribution of this thesis can be broken down into four main research contributions, which are: i) a metamodel definition for RCM, ii) a set of model transformations for the generation of RCM from EAST-ADL models, iii) a mechanism for the automatic selection and back-propagation of the generated RCM models to the design level and iv) a visualisation mechanism for representing the set of the generated RCM models as on RCM models with uncertainty, only.

The formalisation of the RCM metamodel is pivotal for leveraging the proposed approach. The set of model transformations provide automation means for integrating EAST-ADL and RCM, avoiding manual, time-consuming and error-prone activities. In general, the integration between EAST-ADL and RCM requires the generation of a number of RCM models which can rapidly grow exponentially with respect to the number of software components in the software architecture and their allocation. We proposed to solve this by leveraging the properties of a constraint-based transformation language, JTL, to automatically derive all the possible RCM models entailing meaningful and unique timing and allocation configurations. By doing so we could leverage schedulability analysis at design level avoiding the problem of manually iden-

tifying a suitable RCM model, in terms of timing characteristics. In addition to replace the need for late and expensive modifications, the proposed approach also discloses the opportunity of employing expensive resources, such as engineers, more efficiently allowing them to focus only on the design activities exploiting schedulability analysis results without having to investigate nor manually edit execution models. The selection and back-propagation mechanism together with the visualisation mechanism provide a powerful tool for the identification of the best RCM models in terms of timing characteristics thus for taking timing-aware design decisions.

Despite the generation of the RCM models is transparent to the engineer and it can be guided through logic constraints, issues about scalability and performance may remain open when dealing with complex functional models. In this respect, one of the main future investigation direction encompasses the study of a smarter generation process which could reduce the number of the generated RCM models. To this end, one possible solution could be the use of more (domain-)specific logic constraints. Another possible solution to a smarter generation could be to extend the proposed methodology for the optimisation of further system properties, e.g., memory demands. In fact, our experience shows that, besides timing, other relevant system properties need to be dealt with at design level and can be used for pruning the space of the generated RCM models. To summarise, schedulability analysis can represent one step in an exploration chain, where the solution spaces are sequentially investigated based on different system properties [42].

Bibliography

- [1] Manfred Broy, Ingolf H Kruger, Alexander Pretschner, and Christian Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356–373, 2007.
- [2] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: the state of the practice. *Software, IEEE*, 20(6):61–69, 2003.
- [3] Douglas C. Schmidt. Guest Editor’s Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.
- [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.
- [6] AUTOSAR Technical Overview, Version 4.3, The AUTOSAR Consortium, Dec., 2016. <http://autosar.org>.
- [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] Daniel Galin. *Software quality assurance: from theory to implementation*. Pearson Education India, 2004.
- [9] Romina Eramo, Alfonso Pierantonio, and Gianni Rosa. Managing uncertainty in bidirectional model transformations. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 49–58. ACM, 2015.

- [10] Tao Zan, Hugo Pacheco, and Zhenjiang Hu. Writing bidirectional model transformations as intentional updates. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 488–491. ACM, 2014.
- [11] Daniel D Gajski, Frank Vahid, Sanjiv Narayan, and Jie Gong. *Specification and design of embedded systems*. Prentice Hall Englewood Cliffs, 1994.
- [12] Jean Paul Calvez, Alan Wyche, and Charles Edmundson. *Embedded real-time systems*. J. Wiley, 1993.
- [13] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K Mok. Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2-3):101–155, 2004.
- [14] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödín. 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*.
- [15] Alessio Bucaioni, Saad Mubeen, Federico Ciccozzi, Antonio Cicchetti, and Mikael Sjödín. Technology-preserving transition from single-core to multi-core in modelling vehicular systems. In Springer, editor, *13th European Conference on Modelling Foundations and Applications*, July 2017.
- [16] Rick Salay, Michalis Famelis, and Marsha Chechik. Language independent refinement using partial modeling. In *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering, FASE'12*, pages 224–239. Springer-Verlag, 2012.
- [17] 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.
- [18] Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT). OMG Group.
- [19] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. volume 88, pages 1070–1080. MIT Press, 1988.

- [20] 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.
- [21] Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. A model for technology transfer in practice. *IEEE software*, 23(6):88–95, 2006.
- [22] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, and Mikael Sjödin. Anticipating implementation-level timing analysis for driving design-level decisions in east-adl. In *International Workshop on Modelling in Automotive Software Engineering*, September 2015.
- [23] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Saad Mubeen, Mikael Sjödin, and Alfonso Pierantonio. Handling uncertainty in automatically generated implementation models in the automotive domain. In *42nd Euromicro Conference series on Software Engineering and Advanced Applications*, September 2016.
- [24] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.
- [25] Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, 2012. Available at: <http://www.timmo-2-use.org/pdf/T2UBrochure.pdf>. 2012.
- [26] Saad Mubeen, Thomas Nolte, Mikael Sjödin, John Lundbäck, and Kurt-Lennart Lundbäck. Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints. *Software & Systems Modeling*, Jan 2017.
- [27] Friedhelm Stappert, Jan Jonsson, Mottok Jürgen, and Johansson Rolf. A Design Framework for End-To-End Timing Constrained Automotive Applications. In *Embedded Real-Time Software and Systems (ERTS)*, 2010.
- [28] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.
- [29] 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.

- [30] Saad Mubeen, Thomas Nolte, John Lundbäck, Mattias Gålnander, and Kurt-Lennart Lundbäck. *Refining Timing Requirements in Extended Models of Legacy Vehicular Embedded Systems Using Early End-to-end Timing Analysis*, pages 497–508. Springer International Publishing, Cham, 2016.
- [31] Saad Mubeen, Mikael Sjödin, Thomas Nolte, John Lundbäck, Mattias Gålnander, and Kurt-Lennart Lundbäck. End-to-end timing analysis of black-box models in legacy vehicular distributed embedded systems. In *21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2015.
- [32] Bran Selic and Leo Motus. Using models in real-time software design. *Control Systems, IEEE*, 23(3):31–42, June 2003.
- [33] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded system design for automotive applications. *Computer*, 40(10):42–51, October 2007.
- [34] Antonio Cicchetti, Federico Ciccozzi, Silvia Mazzini, Stefano Puri, Marco Panunzio, Tullio Vardanega, and Alessandro Zovi. CHES: a Model-Driven Engineering Tool Environment for Aiding the Development of Complex Industrial Systems. In *27th International Conference on Automated Software Engineering (ASE 2012)*, September 2012.
- [35] Patricia Derler, Edward A. Lee, and Alberto Sangiovanni-Vincentelli. Modeling cyber-physical systems. *Proceedings of the IEEE (special issue on CPS)*, 100(1):13 – 28, January 2012.
- [36] Pieter J. Mosterman and Hans Vangheluwe. Computer automated multi-paradigm modeling: An introduction. *SIMULATION*, 80(9):433–450, 2004.
- [37] Danica Chang Jeff C. Jensen and Edward A. Lee. A model-based design methodology for cyber-physical systems. In *2011 7th International Wireless Communications and Mobile Computing Conference*, pages 1666–1671, July 2011.
- [38] The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, 2010. <http://www.omg.org/omgmarte/>.

- [39] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Rabie Ben Atitallah, Anne Etien, Philippe Marquet, and Jean-Luc Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(4):39, 2011.
- [40] Federico Ciccozzi, Tiberiu Seceleanu, Diarmuid Corcoran, and Detlef Scholle. UML-based Development of Embedded Real-Time Software on Multi-core in Practice: Lessons Learned and Future Perspectives. *Journal of IEEE Access*, 2(1):1–12, September 2016.
- [41] Federico Ciccozzi, Juraj Feljan, Jan Carlson, and Ivica Crnkovic. Architecture optimization: Speed or accuracy? both! *Software Quality Journal*, 22(1):1–28, October 2016.
- [42] 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.

II

Included Papers

Chapter 7

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

IEEE Access (impact factor: 3.244). December, 2016

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.

7.1 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 7.2 presents the context of this work together with its related works. Section 7.3 introduces the RCM metamodel and extensions for timing elements. Section 7.4 shows the DL2RCM transformation while Section 7.5 discusses its applicability on a case study. Finally, Section 7.6 highlights the benefits of having a proper metamodel for RCM while Section 7.7 draws conclusions and discusses future works.

7.2 Background and related work

7.2.1 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 7.1 shows the abstraction levels together with methodologies and languages used at each one of them.

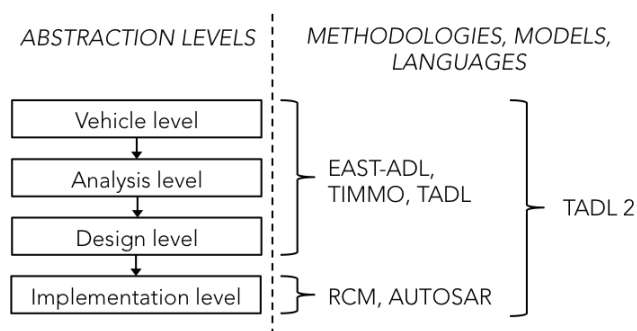


Figure 7.1: EAST-ADL abstraction levels

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.

7.2.2 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 analysis¹. 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 11.3. 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

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

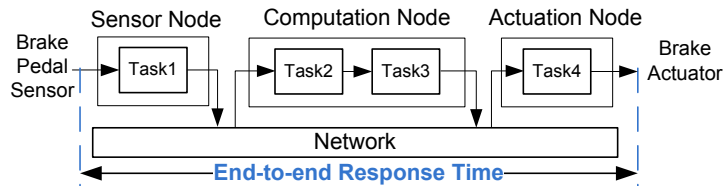


Figure 7.2: Example showing end-to-end response time

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 11.3.

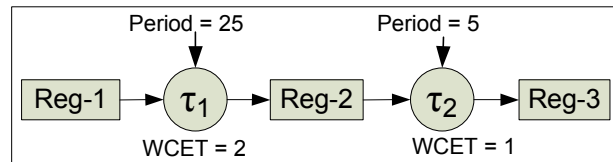


Figure 7.3: A task chain with independent activations of tasks

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 11.4. 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 11.5. The age and reaction delays are also identified in Figure 11.5. 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

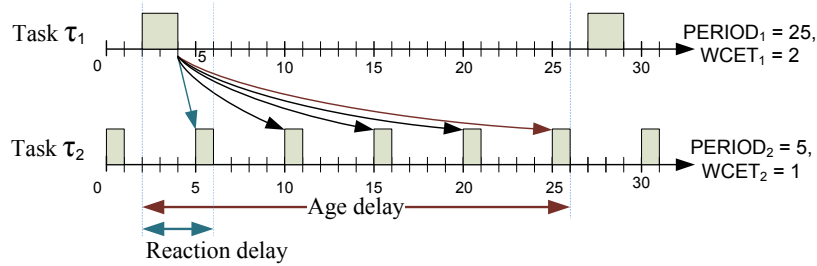


Figure 7.4: Example showing end-to-end delays

al. [20] described end-to-end timing constraints 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].

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

7.3 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 7.5 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

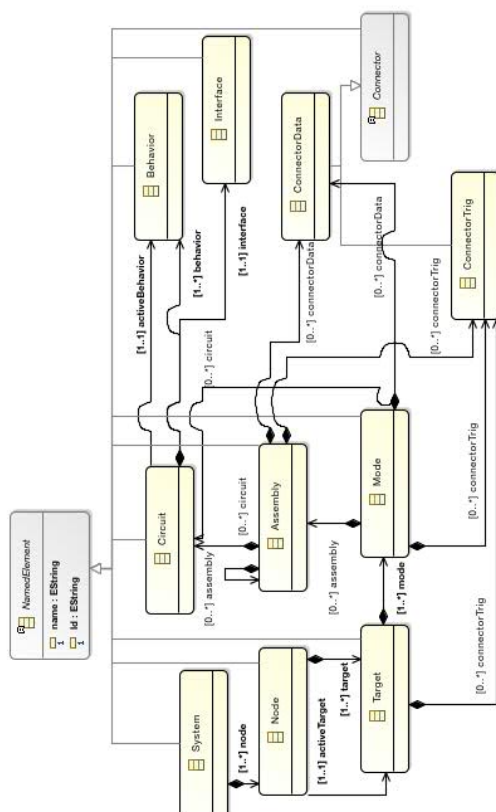


Figure 7.5: Fragment of the RCM metamodel representing the backbone elements

between data ports and control ports, respectively.

RCM explicitly separates data and control flow. Figure 7.6 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

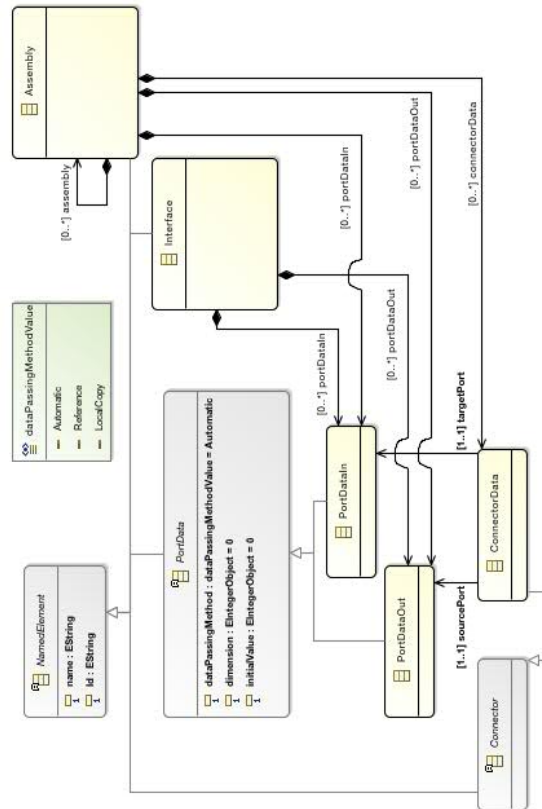


Figure 7.6: Fragment of the RCM metamodel representing the data flow elements

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 7.8 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.

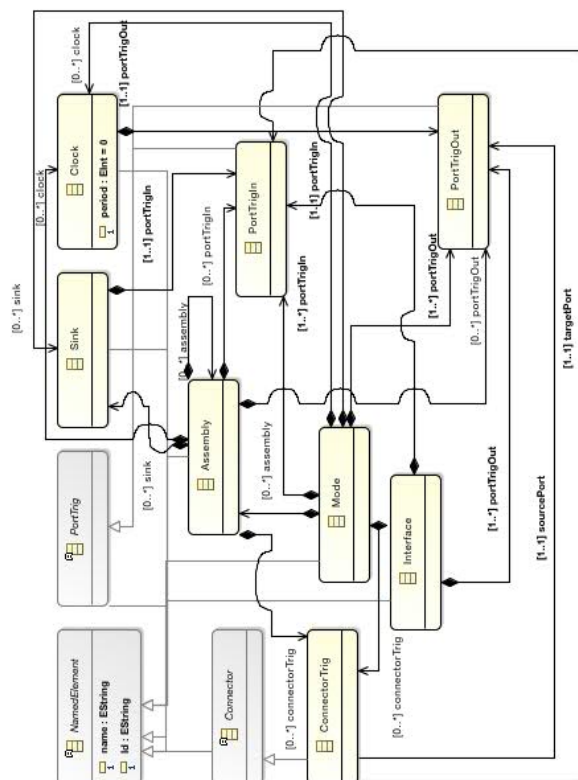


Figure 7.7: Fragment of the RCM metamodel representing the control flow elements

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.

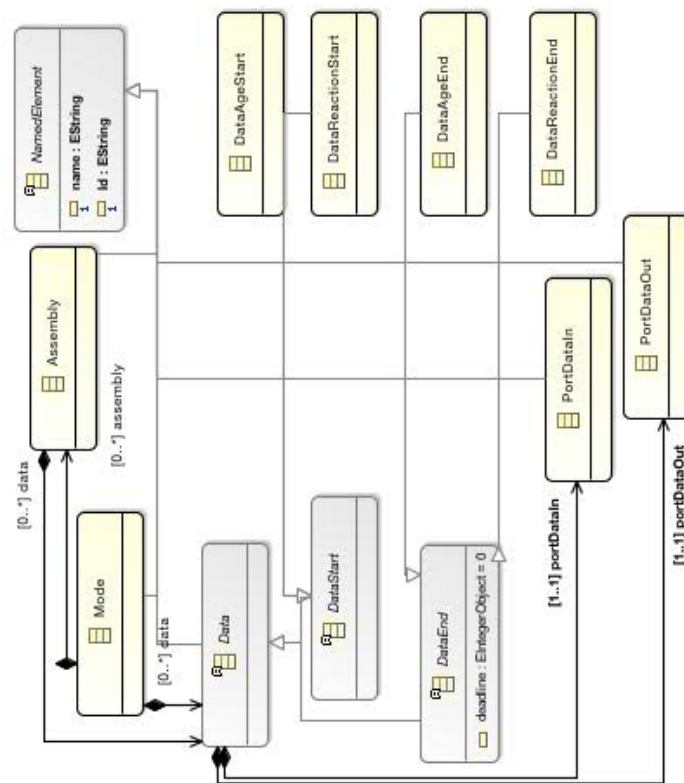


Figure 7.8: Fragment of the RCM metamodel representing the timing constraints and properties for different types of delay in event chains

Figure 7.8 depicts an excerpt of the RCM metamodel containing the metamodels 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 from 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.

7.4 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, 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 7.2, 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 abstraction 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 7.3. The former has been described in [7] and implemented as a UML profile within the Eclipse Papyrus project

2. Figure 7.9 shows the extract of the EAST-ADL metamodel containing the concepts entailed by the DL2RCM transformation³.

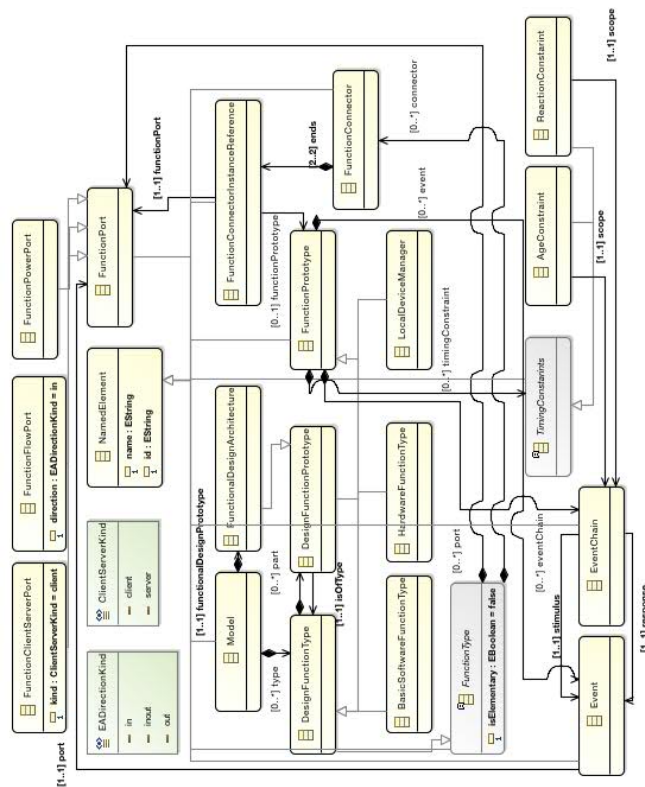


Figure 7.9: Fragment of the EAST-ADL metamodel for Function Modeling at the design level

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

²<http://eclipse.org/papyrus/>

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

| 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 7.1: Main relations holding in the DL2RCM transformation

when defining the relations composing the transformations. Table 7.1 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 7.9). This means that the inner architecture of a prototype is defined through its related type. Such a pattern, not leveraged 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

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

```

pseudocode we make use of helper functions (e.g., CREATEHIERARCHY, 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 *DesignFunctionPrototype* element. In the case it is not an elementary element (line 14), all the contained *DesignFunc-*

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
=CREATECONTROLFLOWIN(fc, a);
33:                 PortDataIn      pdi      =      CREATEPORT-
DATAIN(fc.functionPort, as);
34:             else
35:                 ConnectorTrig      conTS      =      CREATECON-
TROLFLOWOUT(fc, a);
36:                 PortDataOut      pdo      =      CREATEPORT-
DATAOUT(fc.functionPort, as);
37:             end if
38:         else
39:         end if
40:     else
41:         Circuit c = DP2C(e.functionPrototype, as);
42:         if end.functionPort is FunctionFlowPort then
43:             if end.functionPort is in then
44:                 ConnectorTrig      conTC      =      CREATECON-
TROLFLOWIN(fc, a);
45:                 PortDataIn      pdi      =      CREATEPORT-
DATAIN(fc.functionPort, c);
46:             else
47:                 ConnectorTrig      conTS      =      CREATECON-
TROLFLOWOUT(fc, a);
48:                 PortDataOut      pdo      =      CREATEPORT-
DATAOUT(fc.functionPort, c);
49:             end if
50:         else
51:         end if
52:     end if
53: end for

```

tionProtype 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. Af-

Algorithm 1 DL2RCM transformation

```

54: end function
55: function DP2A(DesignFunctionPrototype dfp, Assembly a)
56:   Assembly as = CREATEASSEMBLY(dfp, a);
57:   for connector in dfp do
58:     C2C(connector, a)
59:   end for
60:   for part in dfp do
61:     if part is not elementary then
62:       Assembly as1 = DP2A(part, as);
63:     else
64:       Circuit c1 = DP2C(part, as);
65:     end if
66:   end for
67:   return as;
68: end function
69:
70: function DP2C(DesignFunctionPrototype dfp, Assembly a)
71:   Circuit c = CREATECIRCUIT(dfp, a);
72:   ConnectorTrig conTC = CREATECONTROLFLOWIN(dfp, c, a);
73:   ConnectorTrig conTS = CREATECONTROLFLOWOUT(dfp, c, a);
74:   return c;
75: end function
76:
77: function TC2TC(FunctionalDesignPrototype fdp, Mode mo)
78:   for tc in fdp do
79:     Event startTC = tc.scope.stimulus;
80:     Event endTC = tc.scope.response;
81:     ConnectorData conS = FIND(mo.assembly, startTC);
82:     ConnectorData conE = FIND(mo.assembly, endTC);

```

terwards, the FDP function calls *DP2A* or *DP2C* on its spare *DesignFunction-Prototype* 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)

Algorithm 1 DL2RCM transformation

```
83:   if tc is AgeConstarint then
84:       DataAgeStart startA = CREATEDATAAGESTART(tc);
85:       DataAgeEnd endA = CREATEDATAAGEEND(tc);
86:       ASSIGNPORTS(startA, endA, conS, conE)
87:   else
88:   end if
89:   if tc is ReactionConstarint then
90:       DataReactionStart startR = CREATEDATAREACTION-
START(tc);
91:       DataReactionEnd endR = CREATEDATAREACTIO-
NEND(tc);
92:       ASSIGNPORTS(startR, endR, conS, conE)
93:   else
94:   end if
95:   end for
96:   for part in fdp do
97:       TC2TC(part, mo.assembly)
98:   end for
99: end function
```

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 *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 7.9) for searching, within 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).

7.5 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 7.10. 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 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.

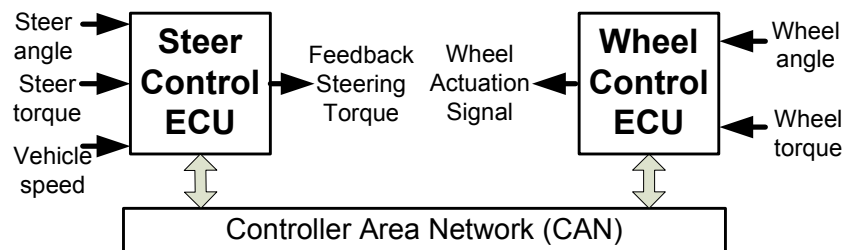


Figure 7.10: Architecture of the steer-by-wire system

For the sake of simplicity and intuitive presentation of the transformation, the simplified internal software architecture of WC ECU is modeled with EAST-ADL using EAST-ADL Rubus Designer⁴ as shown in Figure 7.11.

⁴<http://www.arcticus-systems.com>

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 7.11.

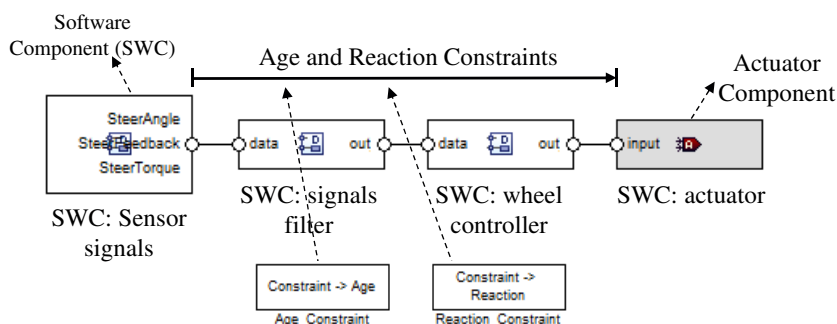


Figure 7.11: Software architecture of WC ECU modeled with EAST-ADL and TADL2

Figure 7.12 shows the Ecore model conforming to the EAST-ADL design level metamodel depicted in Figure 7.9. By applying the DL2RCM transformation presented in Section 7.4, the Ecore model in Figure 7.13 is obtained. Please note that, the model in Figure 7.13 is conforming to the RCM metamodel. 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.

A representation, given in Rubus Designer concrete syntax, of the model showed in Figure 7.13, is presented in Figure 7.14. The specified TADL2 timing constraints (i.e., Age and Reaction) in Figure 7.11 are also translated to RCM timing constraints shown by “Age Start”, “Age End”, “DR Start” and “DR End” objects in Figure 7.14.

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

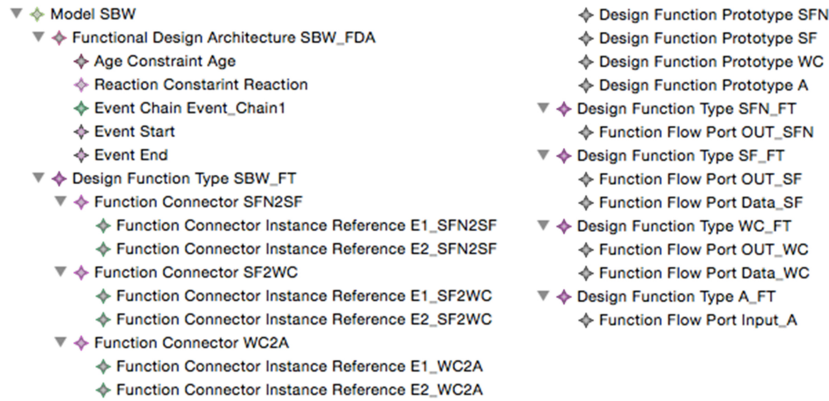


Figure 7.12: Serialized model of the EAST-ADL WC ECU architecture

We assume that all tasks corresponding to the four SWCs in Figure 7.14 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 $60 \mu s$ - $2000 \mu s$. 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 7.14 on which these delays are specified). The calculated age and reaction delays are $5360 \mu s$ and $15360 \mu s$ respectively. A comparison between the specified constraints and calculated delays shows the system satisfies the specified timing constraints.

7.6 Evaluation and discussion

The work discussed in this article finds its motivation and application context within the modernisation 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, as follows:

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

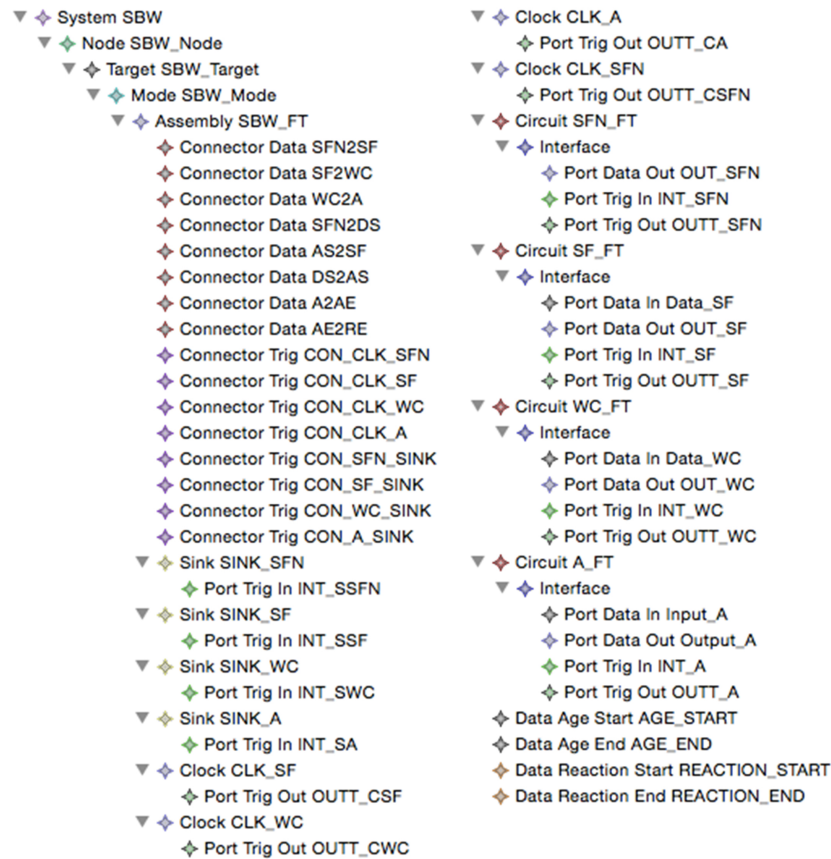


Figure 7.13: Serialized model of the RCM WC ECU architecture

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

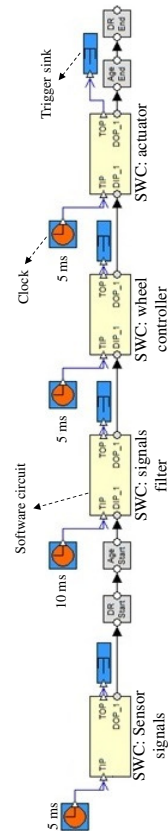


Figure 7.14: Translated software architecture of WC ECU in RCM

both polished some redundancies due to the lower level of abstraction representation of modeling entities and optimised 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 7.4 and 7.5, 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 7.5 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.

7.7 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 hav-

ing 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 Technical 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 8

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

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

1st International Workshop on Modelling in Automotive Software Engineering (MASE) (acceptance rate: 41%) co-located with the 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.

8.1 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 phases². 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., AUTOSAR³ 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.

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

²In 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 level⁴ 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 8.2 we present related work documented in the literature. In Section 8.3 we describe a running example taken from the automotive domain, and in Section 8.4 we apply the proposed methodology to it. In Section 8.5 we discuss benefits and limitations of the proposed methodology and conclude the paper in Section 10.7.

8.2 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

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

process prescribed by the Model-Driven Architecture standard⁵. 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., Simulink⁶ or MARTE⁷). However, some preconditions should hold: i) the transformation language should fully support non-bijectivity; 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].

8.3 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-ICE⁸ 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 8.1. 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 pre-processed 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 μ s, respectively. Since the implementation details are not available at the design level, the WCETs are estimated based on the expert's judgments. 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 8.1.

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

8.4 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>

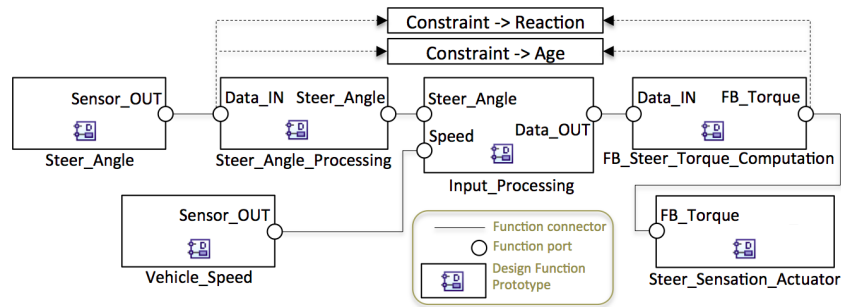


Figure 8.1: Internal software architecture of SC_ECU at design level.

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 8.2, 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 8.2) 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 8.2). 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 8.2); 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 8.2).

8.4.1 Transformation Phase

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

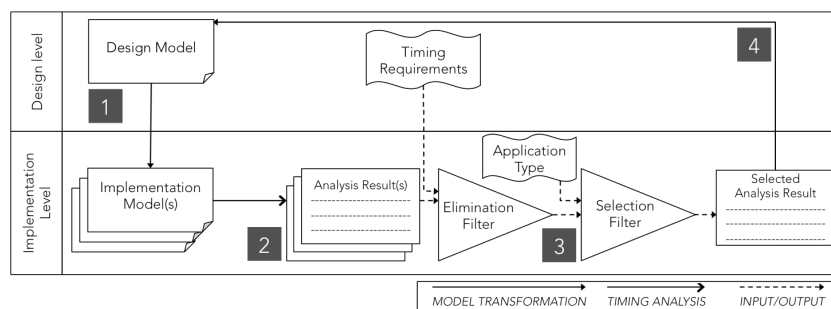


Figure 8.2: Methodology supporting delay analysis at design level.

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 8.1 depicts a fragment of the DL2RCM transformation¹⁰, which is expressed in the textual concrete syntax of JTL and applied on models given by

⁹<http://www.eclipse.org/modeling/emf/>

¹⁰Implementation 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        {
6          name=name,
7          id=id
8        };
9      enforce domain rcm a : RCM::Assembly {
10       connectorData = cd:RCM::ConnectorData {
11         name=name,
12         id=id+"_d",
13         sourcePort = RCM::PortDataOut { ... },
14         targetPort = RCM::PortDataIn { ... }
15       },
16       connectorTrig = ...
17     };
18     where { (con.ends->select(end |
19       end.functionPort.oclIsKindOf(designlevel::
20       FunctionFlowPort) and
21       end.designFunctionPrototype.isOfType.isElementary=true)
22       ->forall(end | E2C(end,a) and E2CCS(end,a) )); }
23   }
24   relation E2CCS {
25     name2, id2: String;
26     checkonly domain dl e :
27       designlevel::FunctionConnectorInstanceReference {
28       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     }
47   where { ... }
48   relation E2C {
49     ...
50   }

```

Listing 8.1: Fragment of the DL2RCM transformation in JTL.

The DL2RCM model transformation, applied to our design model in Figure 8.1, generates 64 implementation models¹¹ (one of them is depicted in Figure 8.3). 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 8.2) 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 8.2: Logic constraint applied to the DL2RCM transformation.

¹¹Each 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 models¹².

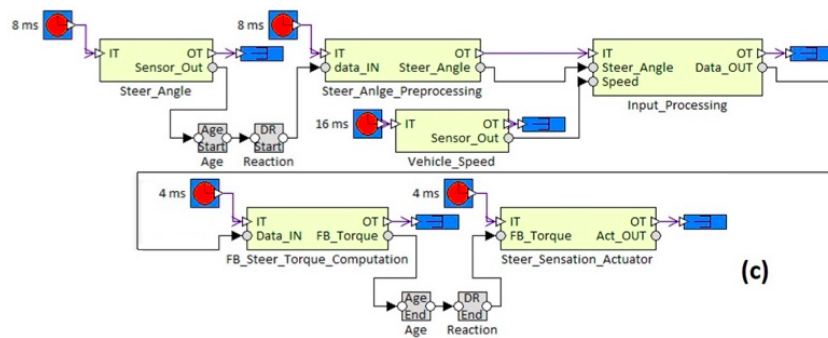


Figure 8.3: Generated implementation model example.

8.4.2 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 8.1, are forwarded to the filtering phase.

¹²All 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 8.1: 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.

8.4.3 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 8.1 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 8.1 as (c), depicted in Figure 8.3, and (g) are selected¹³. Finally, the models and their analysis results are propagated back to the design model (as annotations done by text-to-model transformations).

¹³The 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.

8.5 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.

8.6 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.

Chapter 9

Paper C: Handling Uncertainty in Automatically Generated Im- plementation Models in the Automotive Domain

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

42nd Euromicro Conference Series on Software En- gineering and Advanced Application (SEAA) (acceptance rate: 36%). Limassol, Cyprus. September, 2016

Abstract

Models and model transformations, the two core constituents of Model-Driven Engineering, aid in software development by automating, thus taming, error-proneness of tedious engineering activities. In most cases, the result of these automated activities is an overwhelming amount of information. This is the case of one-to-many model transformations that, e.g. in design-space exploration, can potentially generate a massive amount of candidate models (i.e., solution space) from one single model. In our scenario, from one design model we generate a set of possible implementation models on which timing analysis is run. The aim is to find the best model from a timing perspective. However, multiple implementation models can have equally good analysis results. Therefore, the engineer is expected to investigate the solution space for making a final decision, using criteria which fall outside the analysis' criteria themselves. Since candidate models can be many and very similar to each other, manually finding differences and commonalities is an impractical and error-prone task. In order to provide the engineer with an expressive representation of models' commonalities and differences, we propose the use of modelling with uncertainty. We achieve this by elevating the solution space to a first-class status, adopting a compact notation capable of representing the solution space by means of a single model with uncertainty. Commonalities and differences are thus represented by means of uncertainty points for the engineer to easily grasp them and consistently make her decision without manually inspecting each model individually.

9.1 Introduction

Model-Driven Engineering (MDE) [1] is rapidly evolving in academia and have gained considerable foothold in industrial software-development projects. While model transformations can relieve software developers from significant engineering effort and mitigate errors typical of manual translations, they can also potentially create an overwhelming amount of information. Especially, design-space exploration techniques, characterised by one-to-many model transformations, have the potential to generate hundreds, thousands, or more, candidate solutions (i.e., models) from one single model. Despite automated analyses can be employed for evaluating the appropriateness of each candidate solution – as done in our previous work [2] – their usefulness for the engineer can be limited as the solution space is never really unveiled in the process. In fact, while the analysis refines the solution space by sorting out solutions not complying to given requirements, the engineer still has multiple choices and remains *uncertain* about the one to take: a decision can only be made by manually inspecting and comparing all candidate models. However, since candidate models can be very similar to each other, a manual traversing of the solution space is impractical and error-prone. This is worsened by the fact that the number of alternatives, as well as their size, may grow exponentially for several reasons, e.g., more complex source models.

In this paper, we propose the use of modelling with uncertainty in order to explicitly represent the uncertainty that typically accompanies many stages of the development process [3]. More specifically, we revise our methodology [2] in order to accommodate a compact notation capable of representing the solution space by means of a single model (with uncertainty). The intent is to provide the engineer with an expressive representation of all candidate models with their commonalities and distinctions by means of *uncertainty points*. The engineer can therefore easily grasp the differences among candidate models and consistently make her decision without manually inspecting each model individually. Such a support is provided by employing the metamodel-independent technique presented in [4]. Moreover, an industrial application from the automotive domain is used to illustrate the advantages of the proposal.

Outline. The remainder of the paper is organised as follows. Section 11.2 illustrates the context of this work, while the subsequent section describes a motivating examples taken from the automotive domain. Section 9.4 introduces the uRubus metamodel, i.e., execution models with uncertainty. Section 10.6 discusses the pros and cons of modelling with uncertainty. Section 10.3 presents

related work documented in literature while Sect. 10.7 draws conclusions and future work.

9.2 Background

In the automotive domain, the adoption of models and MDE led to the standardisation of an architectural description language, called EAST-ADL [5]. EAST-ADL proposes a top-down development approach relying on four abstraction levels – vehicle, analysis, design and implementation – which implicitly ensure separation of concerns through the engineering phases. Each abstraction level is described by means of metamodelling constructs and hides unnecessary information from lower abstraction levels. EAST-ADL has been developed with particular focus on the functional and structural modelling. However, it does not focus on execution and timing modelling [6]¹. To this end, EAST-ADL is usually complemented, at implementation level, with additional notations that explicitly support execution and timing modelling. Among other alternatives, Rubus Component Model (RCM) [7] is a modelling language which gained industrial recognition as an EAST-ADL complementary technology. RCM was developed by Arcticus Systems² in collaboration with Mälardalen University and it is currently used by several international companies, e.g., Volvo CE³, BAE Systems⁴, for execution and timing modelling of distributed resource-constrained real-time software systems. EAST-ADL provides means for abstraction and separation of concerns, but it does not provide explicit support for automation among the different abstraction levels.

Therefore, in our previous work [2], we have described a methodology for seamlessly linking the modelling language used at EAST-ADL design level and RCM with the aim of enabling high-precision timing analysis⁵ at EAST-ADL design level. The methodology before this contribution, depicted in Fig. 9.1, leveraged model-driven techniques as follows. Starting from an EAST-ADL design model, the methodology generated the set of all the corresponding meaningful Rubus models, which represented our candidate solutions. The gener-

¹Lately, EAST-ADL has been extended for supporting the modelling of timing requirements [5].

²<https://www.arcticus-systems.com>

³<http://www.volvoce.com/dealers/sv-se/swecon/Pages/homepage.aspx>

⁴<http://www.baesystems.com>

⁵In the remainder of the paper, *high-precision timing analysis* is referred simply as *timing analysis*

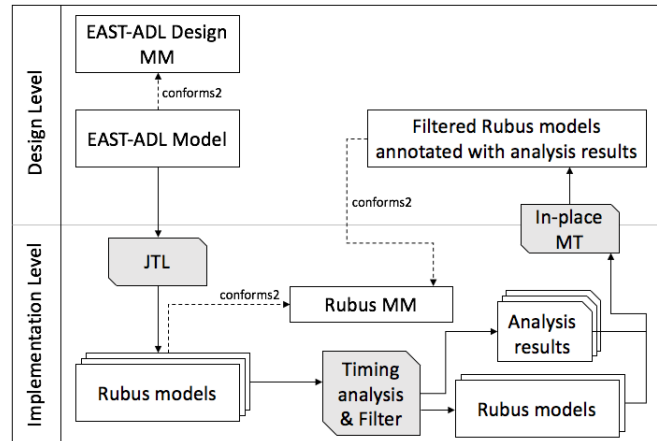


Figure 9.1: Original methodology supporting timing analysis

ation was performed by means of a model transformation defined in JTL [8] that non-deterministically generated all the models satisfying the constraints encoded in the transformation itself. At this point, timing analysis was run on the generated Rubus models resulting in a set of analysis results. These results were checked against a non-empty set of timing requirements expressed on the vehicle functionality. The result which better met the given timing requirements was selected and the corresponding Rubus model was conveyed back⁶ to the engineer. It is worth noting that, when selecting among analysis results, multiple results and thereby Rubus models could be selected if timing requirements were met. However, when this happened, the engineer was required to manually inspect the set of *filtered* Rubus models annotated with analysis results individually. From a broader perspective, this operation is frequent in human-in-the-loop processes where domain knowledge is needed to meet decisions that cannot be made by the tools. Therefore, providing (semi-) automated support that prevents the engineer from manually traversing the solution space is key to success.

To this end, we realised that there was need for our methodology to entail a compact notation to represent the solution space (e.g., Rubus models) by means of a model with uncertainty. In such a representation, model differences

⁶Back-propagation was achieved through in-place model transformations that annotated filtered Rubus models with related analysis results.

are enucleated in uncertainty points that provide the engineer with a straightforward locality for understanding how models differ one with another.

9.3 Motivating Scenario

Let us apply the methodology introduced in Sect. 11.2 on the automotive application called Intelligent Parking Assist (IPA) system. The IPA system assists drivers in parking their vehicles. To this end, it uses a warning system, composed of a set of proximity sensors and backup cameras, for detecting obstacles and calculating optimum manoeuvres.

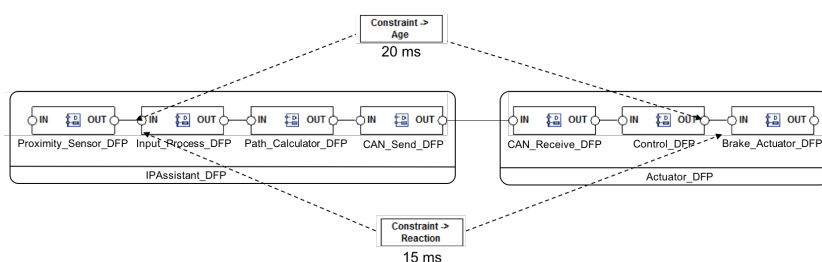


Figure 9.2: Partial software architecture of the two nodes in IPA system at the design-level of EAST-ADL.

For the sake of simplicity, we consider only a portion of the software architecture consisting of two nodes, namely *IPAssistant* and *Actuator* connected to a single network that implements the Controller Area Network (CAN) protocol [9] (Fig. 9.2). Figure 9.2 depicts the EAST-ADL design level model of the partial software architecture⁷. In the hierarchy of an EAST-ADL design model, the so-called design function prototype (DFP) represent a specific instance of a vehicle functionality⁸. The partial IPA architecture consists of seven DFPs in a chain. *Proximity_Sensor_DFP*, *Input_Process_DFP*, *Path_Calculator_DFP* and *CAN_Send_DFP* DFPs are part of the software architecture of the *IPAssistant* node. The remaining three DFPs in the chain, *CAN_Receive_DFP*, *Control_DFP* and *Brake_Actuator_DFP* are part of the software architecture of the

⁷We have modelled the IPA system with the help of Rubus ICE [10]

⁸EAST-ADL implements the type-prototype pattern. Therefore, a DFP represents a specific instance of a design function type, which defines its type. the complete explanation of the EAST-ADL metamodel is not in the scope of this work. The interested reader is referred to [5] for details

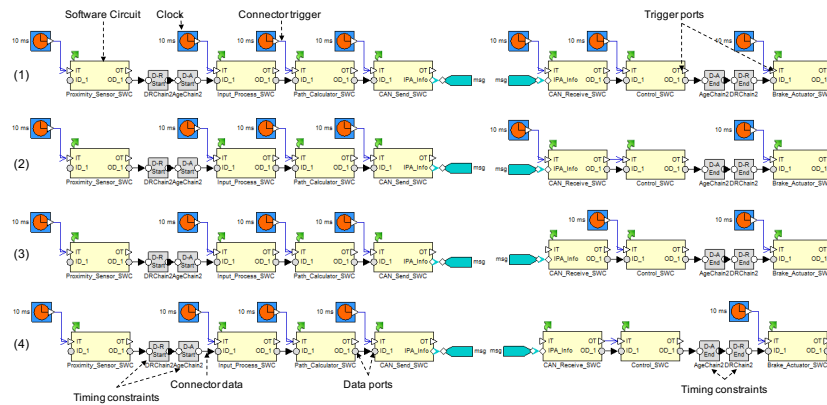


Figure 9.3: 4 of the 32 Rubus models generated from the EAST-ADL model

Actuator node. Please note that CAN_Send_SWC sends a network message that is received by CAN_Receive_SWC. There is a periodic constraint of 10 ms that is specified on each DFPs in the chain. However, the information about whether each DFPs is activated independently or by its predecessor is not available. The following timing requirement is specified too:

- “The calculated age and reaction delays [11] shall not exceed 20 ms and 15 ms, respectively.”

Within EAST-ADL, timing requirements are specified by timing constraints [12]. Therefore, there are two timing constraints, namely Data Age (AgeChain2) and Data Reaction (DRChain2), that are specified from the input flow port of Input_Process_SWC to the output flow port of Control_SWC as shown in Fig. 9.2.

So far according to our original methodology (Fig. 9.1), the EAST-ADL model in Fig. 9.2 is transformed in 32 Rubus models. The first 4 Rubus models⁹ are depicted in Fig. 9.3. In the hierarchy of a Rubus model, a software circuit (SWC) encapsulates basic software functions. RCM distinguishes between data and control flow therefore a SWC has data and trigger ports. 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. A simplified version of the Rubus metamodel

⁹The interested reader can find the whole set of artefacts at <http://www.mrtc.mdh.se/SEAA2016>.

is presented in Sect. 9.4. All these models differ from each other depending upon whether a SWC is activated independently by a clock element or by its preceding SWC. Considering the age constraint of 20 ms specified in Fig. 9.2, only 14 out of 32 Rubus models satisfy it whereas only 1 Rubus model satisfies the specified reaction constraint of 15 ms⁹. Despite the automated analysis has filtered the solution space, there are still 14 Rubus models which must be inspected by the engineer for deciding which one should be selected for proceeding in the development process. However, with the current support, such an inspection might be a daunting task as the selected Rubus models greatly overlap one with another. For instance, let us consider the Rubus models marked with (1) and (2) in Fig.9.3. The only difference between these two models is on how the Control_SWC SWC is activated: in the model marked with (1) it is activated from a clock element, while in the model marked with (2) is activated from its preceding SWC. If these small differences are hard to catch when dealing with a reasonably small number of models and model elements, they are nearly impossible to spot when dealing with hundreds or thousands models and model elements.

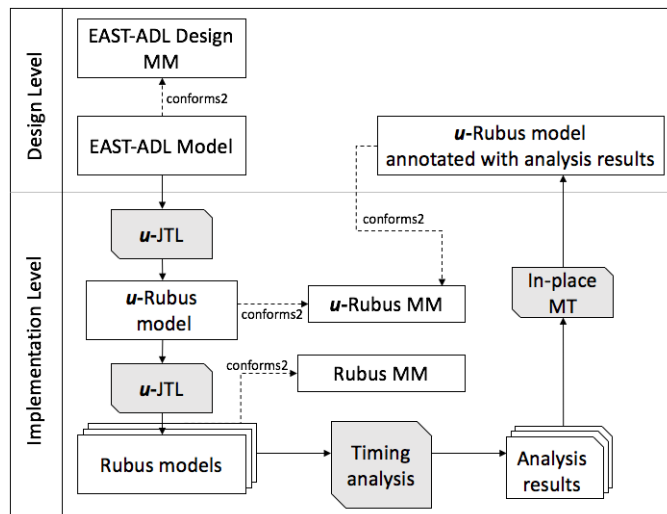


Figure 9.4: New methodology supporting timing analysis and uncertainty

Contribution. In order to ease the inspection of the solution space represented by Rubus models, we enhanced our methodology (Fig. 9.4) by introducing

the *u*-Rubus metamodel, a compact notation to represent the solution space by means of a single *u*-Rubus model (that can represent uncertainty). Uncertainty points are employed for representing commonalities and distinctions of the Rubus models. The *u*-Rubus metamodel was generated by means of an automated transformation defined in the revised version of JTL, which we hereafter call *u*-JTL [4]. Since our timing analysis is currently not able to run on *u*-Rubus models, we exploit a concretiser operator (in the sense of [3]) provided by *u*-JTL that, starting from a *u*-Rubus model, returns all Rubus models encoded in it and on which timing analysis can be run. Analysis results are then back-propagated as annotations to the *u*-Rubus model through an in-place model transformation.

9.4 *u*-Rubus

In this section, we present the *u*-Rubus metamodel. Such a modeling notation is obtained by endowing Rubus with uncertainty elements in order to deal with the multitude of Rubus models presented above. The intent is to provide the engineer with a representation that permits to deal with a set of Rubus models as if they were a single model and do reasoning with all the possible models at the same time.

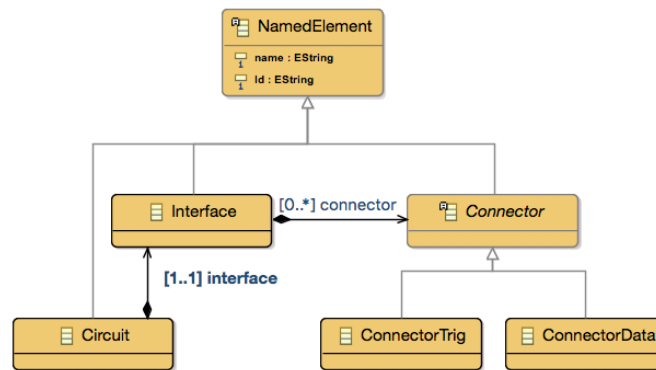
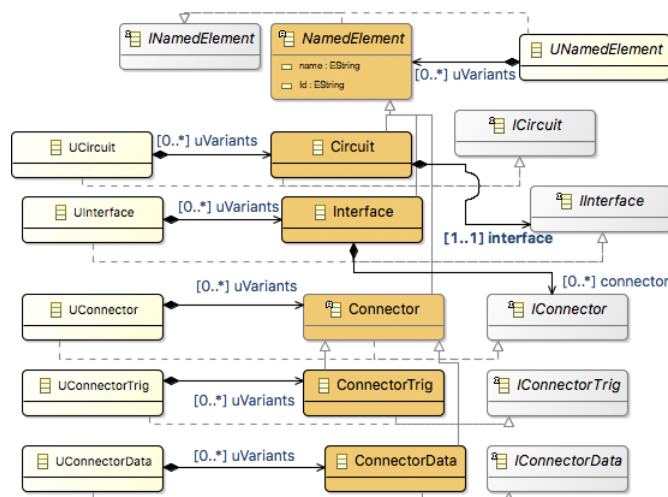


Figure 9.5: A Rubus metamodel fragment

With reference to the small Rubus fragment⁹ in Fig. 9.5, an execution model consists of *Circuit*(s), that have exactly one *Interface* with *Connector*(s). In turn, connectors can be either *ConnectorData* or *ConnectorTrig*.

Figure 9.6: A *u*-Rubus Metamodel fragment

Trig to denote data- and control-flow linking a circuit to another. In Fig. 9.3 part of the Rubus models generated with a JTL program with the original methodology are shown. In many cases it has been observed that generated models share most of their model elements, making engineer's life harder as comprehending the differences among the models is not always straightforward.

Recently, the JTL language has been given an *intensional* semantics in order to generate models with uncertainty [4]. Transformations, instead of delivering myriads of models, can generate models with uncertainty, i.e., models denoting *multiple possibilities*. As a result, engineers do not need to manually compare models to discern between them anymore, but rather they can combine the variants associated to uncertainty points to explore the solution space. In order to consistently represent *uncertain* elements, i.e., elements that are optional or mutually exclusive, the Rubus metamodel has to be extended with additional constructs. This is performed by an automated transformation (see [13]) that, starting from uRubus, generates the *u*-Rubus metamodel shown in Fig. 9.6 as follows:

- i) any *class* in Rubus is added to *u*-Rubus; in addition
- ii) auxiliary classes *Uclass* and *Iclass*, with *class* and *Uclass* subclasses of

Iclass, are added to *u*-Rubus;

- iii) association *uVariants* : *Uclass* $\diamond \rightarrow$ *class* is added to *u*-Rubus;
- iv) for each association *a* : *class*₁ $\diamond \rightarrow$ *class*₂ in Rubus, an association *a* : *class*₁ $\diamond \rightarrow$ *Iclass*₂ is added to *u*-Rubus.

In particular, the procedure can also be illustrated in term of pattern rewriting rules as illustrated in Fig. 9.7. The first three steps (*i-iii*) realize the mapping from the source to the target pattern in the first row, while the last step (*iv*) is represented in the second row. For instance, when the first mapping is applied to *Circuit* in Rubus, then it is propagated to *u*-Rubus together with the newly created *UCircuit* and *ICircuit* metaclasses as shown in Fig. 9.6. The *UCircuit* metaclass represents uncertainty points where to anchor multiple alternative *Circuit* instances. Whereas, the role of the *IInterface* is to let the propagated *interface* composition in *u*-Rubus to refer to either a single *Interface* instance or to multiple instances through the *UInterface* (as subclass of *IInterface*).

As aforementioned, the new methodology makes use of *u*-Rubus models for representing the solution space as for instance illustrated in Fig. 9.8. In particular, the green elements u_1, \dots, u_5 are *UConnectorTrig* uncertainty points representing two (mutually excluded) connectors each: a timed one triggered by a clock element, say u'_i , and another directly triggered by the preceding circuit, say u''_i . Such a model represents an overall number of 2^5 Rubus models¹⁰. Currently the timing analysis can only be performed on sets of individual Rubus models, therefore the multivalued *concretisation* operator (see [14]), part of the *u*-JTL environment, and defined as

$$\text{concr} : \text{uRubus} \rightarrow \mathcal{P}(\text{Rubus}),$$

returns the set of all Rubus models encoded in the corresponding *u*-Rubus model. It is worth noting that the original JTL transformation, in charge of generating a set of Rubus models from an EAST-ADL model, did not have to be modified to generate *u*-Rubus models. This is due to the fact that the new *u*-JTL transformation engine is semantically equivalent to the one of JTL, although the way models are represented is different. Once the Rubus models are obtained by concretising the *u*-Rubus model, we can perform timing analysis. Without being too specific, the outcome of such an analysis is a subset of Rubus models satisfying given timing requirements.

¹⁰The overall number comes from the number of possible variants (2) to the power of the number of the connector uncertainty points u_i (5).

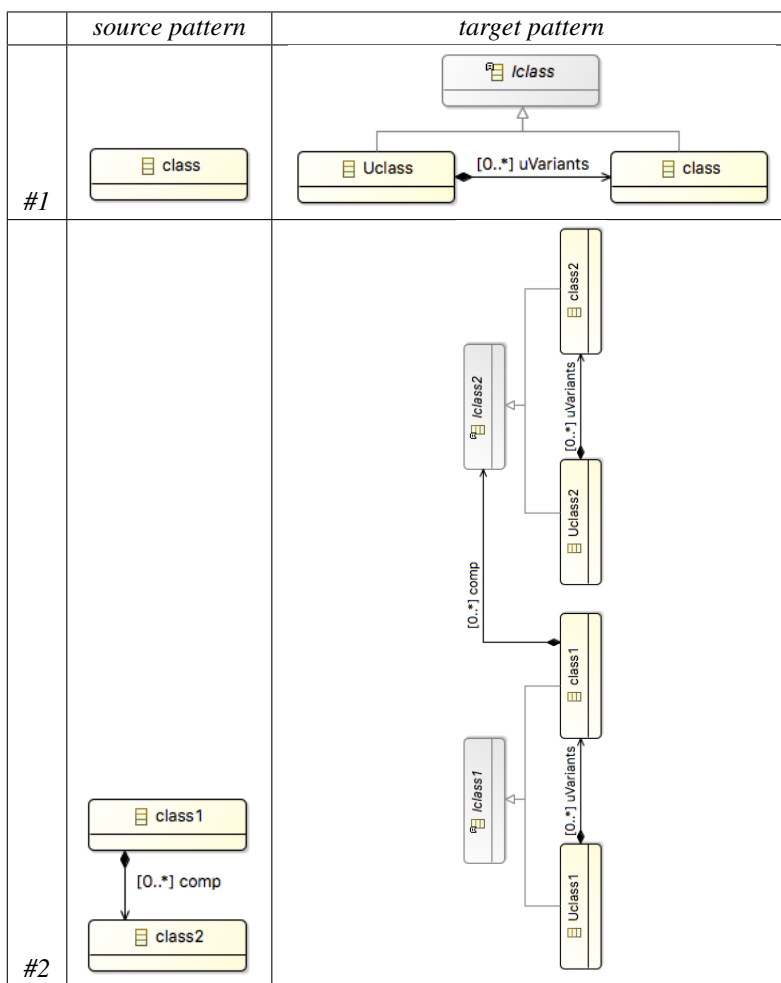


Figure 9.7: Rubus to *u*-Rubus mappings

Each Rubus model obtained by means of the *concr* operator is univocally identified by the variants chosen for each uncertainty point u_i . For instance, in the case shown in Fig. 9.8 the Rubus model with only clock elements is given by the 5-tuple $\langle u'_1, \dots, u'_5 \rangle$. Therefore, the tuples identifying all the models, which satisfied the timing analysis, are translated back into annotations in the

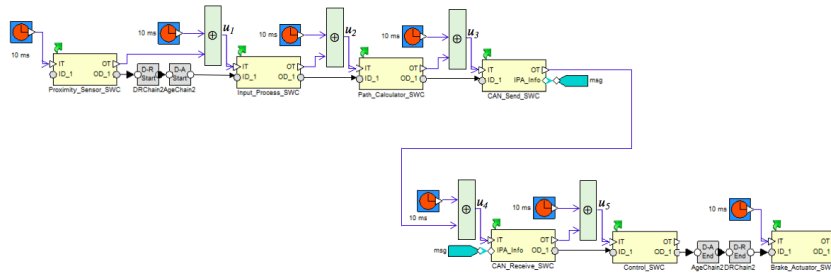


Figure 9.8: The u -Rubus model generated by u -JTL representing the solution space

u -Rubus model together with analysis results.

Besides being able to better locate the differences among the models, the main advantage of this proposal consists in harnessing the possibility to reason with all the models as a whole. In fact, the engineer can search through the models, which passed the timing analysis, by discarding those with characteristics non conforming to criteria that fall outside the analysis itself. For instance, in the context of the automotive application presented in Sect. 9.3, timing variations are of crucial relevance that the engineer cannot neglect. At times, it might be very important to prefer models presenting more independent clocks because they can better accommodate the branching and merging of data along the chain. Also, independent clocks suit better to SWCs having more than one data input port. On the other hand, it might be desirable in some other models to have dependent activation of SWCs receiving messages from the network, e.g., CAN_Receive_SWC in Fig. 9.8 as they ensure that fresh data from the network traverses through the rest of the model.

9.5 Discussion

In this paper, we have proposed a compact notation for representing a solution space by means of a model with uncertainty. We have described how the proposed notation can ease the exploration of the solution space, especially when candidate solutions display minimal variations among themselves. This contribution enhances our methodology for seamlessly linking EAST-ADL and RCM. More specifically, the notation, addressed as Rubus with uncertainty (u -Rubus), allows the developer to inspect the solution space represented by

the set of Rubus models automatically generated from a single EAST-ADL design level model as valid implementation alternatives. In the scenario presented here, timing analysis is run on the initial set of generated Rubus models. Thus, the set of valid alternatives, from a timing perspective, is selected. These alternatives can be very similar to each other, and it may be hard for the engineer to effectively compare them and comprehend how they differ one with another. This difficulty is exacerbated by the number of alternatives (as well as their modelling elements) that may grow exponentially due to, e.g., loose timing requirements. u -Rubus spawns the means for the engineer to grasp at a glance the solution space of interest through a compact visualisation of uncertainty points represented in a single model. The exploration of the solution space remains manual. In fact, this contribution represents a first step towards an analysis-based and semi-automatic design-space exploration mechanism for guiding the engineer towards selecting the most suitable Rubus model among a possibly huge set of alternatives. We have already started investigating the possibility to extend our methodology (Fig. 9.9) for running timing analysis on a u -Rubus model instead of the set of Rubus models. Doing so, we would be able to entirely act on u -Rubus model, from start to end, with no need for concretising/de-concretising mechanisms from/to a u -Rubus model. Moreover, with a u -Rubus model as sole artefact, we could be able to provide an analysis mechanism that, while analysing the candidate solutions (in terms of the u -Rubus model) also gives the possibility to the engineer to interactively decide upon uncertainty points (when and whether she wishes so) based on partial analysis results.

While we showed how u -Rubus can be exploited for investigating the solution space of Rubus models representing timing, this does not necessarily mean that a decision on a single alternative must always be taken. In fact, the idea of a compact notation can be exploited for successively exploring the solution space of models in relation to various properties. Let us imagine that timing and power consumption are two properties of interest. The engineer could start with running timing analysis for reducing the solution space. u -Rubus for timing (shown in this paper) would then be exploited for selecting among equally good alternatives (from a timing perspective). It can happen that, even with the help of u -Rubus, the engineer is not able or does not want to solve all the uncertainty points. At this point, unsolved uncertainty points, representing a set of Rubus models, would be analysed to measure expected power consumption. The results of this analysis will provide the engineer with additional information for her to decide in two ways: *i*) by decorating the model that instantiates u -Rubus for timing with power-related details (if timing-related

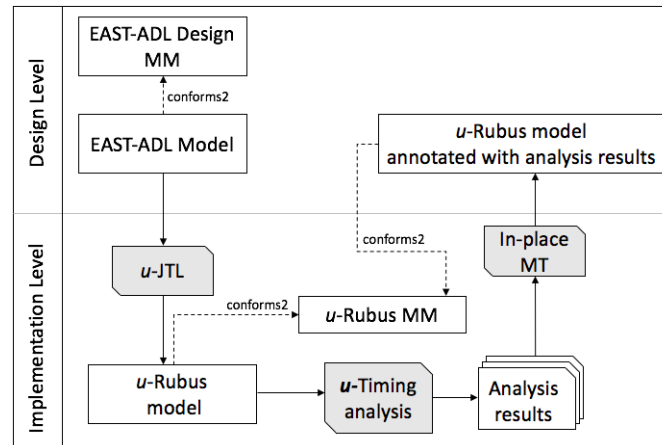


Figure 9.9: Future methodology supporting timing analysis for uncertainty

details are still needed) or *ii*) by creating a specifically generated u -Rubus for power consumption to switch the focus of investigation and selection to power only (in case decisions on timing are considered established). To summarise, timing analysis can represent one step in a potential exploration chain [15], where solution spaces are successively investigated based on different properties prioritisation.

9.6 Related Work

The problem of generating, analysing, and optimising multiple design alternatives has been largely investigated and is usually referred to as design-space exploration (DSE) [16]. Rule-based DSE [17, 18] can be considered as a possible implementation of the exploration in a MDE context: the space of available solutions is expressed in terms of a model and transformations are used to derive the corresponding alternatives. Depending on the characteristics of those models and transformations, in [15] the authors introduced a catalog of exploration patterns: our approach complies to the model generation pattern, that is it performs an exhaustive derivation of implementation models (lower level of abstraction), enriched with timing details, as derivable from the system architecture designed through EAST-ADL, and constrained by domain-specific rules. The generation is not meant to provide optimisation hints/solutions at

architectural level [19]; rather, it implements a quality-driven model transformation [17, 20] to select all the suitable (timing configuration) results given a certain system architecture as input.

In general, the goal of DSE mechanisms is reaching an optimal solution in terms of certain properties of interest, therefore the available works usually focus on generating appropriate candidates in an effective way. Moreover, based on user's choice and/or heuristics, potential solutions are pruned and the exploration is driven towards optimal alternatives. Abdeen et al. [18] propose to combine genetic algorithms with rule-based DSE in order to achieve a domain-independent multi-objective optimisation process. The approach is fully automated, hence aiming at reaching the optimal solution without user intervention. Instead, in [21] the authors embed search-based mechanisms in a model transformation language to generate optimal models as solutions to design problems. Generated models are evaluated through specific metrics, typically encoded in the transformation as rules and constraints. User's input is mentioned as a solution evaluation possibility, however no further clarification is given with respect to dealing with the presentation of the alternatives.

In Schätz et al. [17] rule-based DSE for the development of embedded systems is supported by means of a declarative generation approach. A model transformation mechanism hosted in Prolog is exploited to both define exploration rules and constrain the set of suitable alternatives. Differently to our approach, the user has to mentally render the set of available choices and write corresponding predicates to narrow down the solution space, which in our opinion can be a more complex task than visually comparing the generated alternatives.

The DESERT tool [22] provides support for DSE based on constraints, where the exploration and pruning rules have to be manually defined by the user. Similarly to our proposal, DESERT offers a more compact representation of available design alternatives in terms of ordered binary decision diagrams. However, this approach translates to an element-by-element choice which is devoted to the selection of a single preferred solution among the generated ones. Instead, our models with uncertainty allow to examine a solution as a whole and to keep multiple design alternatives until the necessary maturity was achieved to take a more constrained decision. In this respect, Kang et al. [23] advocate the need of cost-effective DSE by avoiding the exploration of design aspects irrelevant for a certain phase of the design process. In fact, depending on the maturity of the design, some alternatives might look equivalent to the user whom is not yet concerned with some of the details about the system that are changed. The authors introduce also a tool, FORMULA,

supporting a user-defined equivalence specification among solutions that guarantees the sole generation of non-isomorphic alternatives with respect to the existing equivalence relationships. The uncertainty representation introduced in this work supports FORMULA's vision in the sense that we permit to keep some choices open until a definitive decision can be taken. Moreover, the definition of uncertainty itself can be exploited as a definition of aspects to be explored, and uncertainty resolution techniques [24] can be used as alternative generation mechanism.

There exists a number of additional approaches, such as [25, 26, 27] to mention a few, which propose generic representations of the solution space tailored to DSE from different perspectives, i.e. targeting multiple optimisation aspects. Notably, Saxena and Karsai [25] introduce a generic DSE framework based on the extension of a domain-specific language for exploration purposes. Such extension is then translated to an appropriate intermediate format that can be exploited by multiple constraint solvers to compute disparate optimisations. The resulting solutions are listed and can be visualised in the tool, however a one-by-one browsing of the alternatives might be difficult to handle for the user, especially when their number grows and the difference between them was minimal. A similar approach is adopted in Octopus [27], a tool that supports DSE for software intensive embedded systems. A domain-specific model is translated towards a DSE tailored intermediate representation, which is exploited to perform several exploration tasks as analyses, searches, and diagnostics. The underlying goal is to implement an iterative development and refinement of the application model until the desired set of properties is completely satisfied. The intermediate representation can be also exploited to perform optimisations through parametrisation of selected properties, however the management of the potential uncertainty raised by the optimisation is not discussed in the work.

GASPARD [26] is a framework for the development of massively parallel embedded systems, and shares several solution mechanisms with what is described in our contribution. In particular, GASPARD provides a higher abstraction level modelling support based on UML and the MARTE profile; starting from such design level, the framework prescribes a workflow made-up of subsequent analyses and refinement steps, from higher to lower abstraction levels. Similarly to the automotive development process described in this paper, some analyses and refinements can be performed at the (EAST-ADL) design level, while others require lower abstraction details (notably timing). Moreover, the transition from higher to lower abstraction levels naturally raises the issue of managing multiple lower level alternatives for the same higher level model. Indeed, also in [26] the authors advocate for a refinement process able to support

the growing number of alternatives that should be reduced step-by-step by an analysis method and the corresponding pruning of inadequate solutions. However, the authors do not provide any detailed discussion about the management of multiple alternatives at each step, and the refinement process seems to rely on the selection of a single candidate for each level of abstraction. In such a context, exploiting models with uncertainty would disclose the opportunity of keeping equally good alternatives for the next (lower) abstraction level, until an analysis would definitively discard a certain solution.

9.7 Conclusion and Future Work

As software systems increase in size, complexity and heterogeneity there is a growing consensus on the need to leverage existing techniques, methods, and tools across abstraction. In the context of automotive software, model-based techniques underpin analyses that typically refine solution spaces, which consists of hundred, thousand, or more candidate solutions. Nevertheless, often the engineer might want to comparatively inspect the models in order to consider additional requirements that fall outside those taken into account by the analyses. In this paper, we enhance our previous methodology by introducing the μ -Rubus metamodel: a compact notation for formalizing the whole solution space in terms of models with uncertainty. The advantages of the proposal consists in letting the engineer *i*) to reason about multitudes of models as a whole; and *ii*) to better locate the differences among the models for identify models fulfilling criteria dictated by the engineer's domain expertise. This work represents a first attempt in leveraging abstraction and automation in design space exploration. Future work will investigate how timing analysis for individual Rubus models can be lifted to μ -Rubus models in order to have better analysis performance and provide the engineer with more immediate feedback.

Acknowledgments

This work is supported by the Swedish Knowledge Foundation (KKS) through the SMARTCore project, by the Swedish Research Council (VR) through the SynthSoft project, and by the Swedish Foundation for Strategic Research (SSF) through the PRESS project. We thank our industrial partners Arcticus Systems AB and Volvo CE, Sweden. Moreover, the authors are grateful to Gianni Rosa for his comments and insights during technical discussions.

Bibliography

- [1] D C Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.
- [2] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, and Mikael Sjödin. Anticipating implementation-level timing analysis for driving design-level decisions in east-adl. In *International Workshop on Modelling in Automotive Software Engineering*, September 2015.
- [3] Michalis Famelis, Rick Salay, and Marsha Chechik. Partial models: Towards modeling and reasoning with uncertainty. *ICSE*, pages 573–583, 2012.
- [4] Romina Eramo, Alfonso Pierantonio, and Gianni Rosa. Managing uncertainty in bidirectional model transformations. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 49–58. ACM, 2015.
- [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] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture*, <http://dx.doi.org/10.1016/j.sysarc.2013.10.008>, Oct. 2013.
- [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] 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.
- [9] ISO 11898-1. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.
- [10] Rubus-ICE: Integrated component Development Environment, 2013. <http://www.arcticus-systems.com>.
- [11] 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.
- [12] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.
- [13] Romina Eramo, Alfonso Pierantonio, and Gianni Rosa. Uncertainty in bidirectional transformations. In *6th International Workshop on Modeling in Software Engineering, MiSE 2014 - Proceedings*, pages 37–42, New York, New York, USA, January 2014. University of L’Aquila, L’Aquila, Italy, ACM Press.
- [14] Rick Salay, Michalis Famelis, and Marsha Chechik. Language independent refinement using partial modeling. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 224–239. University of Toronto, Toronto, Canada, April 2012.
- [15] 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.
- [16] Matthias Gries. Methods for evaluating and covering the design space during early design development. *Integr. VLSI J.*, 38(2):131–183, December 2004.

- [17] B. Schätz, F. Hölzl, and T. Lundkvist. Design-space exploration through constraint-based model-transformation. In *Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on*, pages 173–182, March 2010.
- [18] Hani Abdeen, Dániel Varró, Houari Sahraoui, András Szabolcs Nagy, Csaba Debreceni, Ábel Hegedüs, and Ákos Horváth. Multi-objective optimization in rule-based design space exploration. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14*, pages 289–300, New York, NY, USA, 2014. ACM.
- [19] Martin Walker, Mark-Oliver Reiser, Sara Tucci-Piergiovanni, Yiannis Papadopoulos, Henrik Lnn, Chokri Mraidha, David Parker, DeJiu Chen, and David Servat. Automatic optimisation of system architectures using east-adl. *Journal of Systems and Software*, 86(10):2467–2487, 2013.
- [20] Mauro Luigi Drago, Carlo Ghezzi, and Raffaella Mirandola. Towards quality driven exploration of model transformation spaces. In *Procs. of the 14th Int. Conf. on Model Driven Engineering Languages and Systems, MODELS'11*, pages 2–16, Berlin, Heidelberg, 2011. Springer-Verlag.
- [21] Joachim Denil, Maris Jukss, Clark Verbrugge, and Hans Vangheluwe. *System Analysis and Modeling: Models and Reusability: 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings*, chapter Search-Based Model Optimization Using Model Transformations, pages 80–95. Springer International Publishing, 2014.
- [22] Sandeep Neema, Janos Sztipanovits, Gabor Karsai, and Ken Butts. *Embedded Software: Third International Conference, EMSOFT 2003, Philadelphia, PA, USA, October 13-15, 2003. Proceedings*, chapter Constraint-Based Design-Space Exploration and Model Synthesis, pages 290–305. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
- [23] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems: 16th Monterey Workshop 2010, Redmond, WA, USA, March 31-April 2, 2010, Revised Selected Papers*, chapter An Approach for Effective Design Space Exploration, pages 33–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

- [24] Rick Salay, Michalis Famelis, and Marsha Chechik. Language independent refinement using partial modeling. In *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*, FASE'12, pages 224–239, Berlin, Heidelberg, 2012. Springer-Verlag.
- [25] Tripti Saxena and Gabor Karsai. *Model Driven Engineering Languages and Systems: 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*, chapter MDE-Based Approach for Generalizing Design Space Exploration, pages 46–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [26] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Rabie Ben Atitallah, Anne Etien, Philippe Marquet, and Jean-Luc Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Trans. Embed. Comput. Syst.*, 10(4):39:1–39:36, November 2011.
- [27] Twan Basten, Martijn Hendriks, Nikola Trčka, Lou Somers, Marc Geilen, Yang Yang, Georgeta Igna, Sebastian Smet, Marc Voorhoeve, Wil Aalst, Henk Corporaal, and Frits Vaandrager. *Model-Based Design of Adaptive Embedded Systems*, chapter Model-Driven Design-Space Exploration for Software-Intensive Embedded Systems, pages 189–244. Springer New York, New York, NY, 2013.

Chapter 10

Paper D: Technology-preserving transition from single-core to multi-core in modelling vehicular systems

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

13th European Conference on Modelling Foundations and Applications (ECMFA)
(acceptance rate: 38%). Marburg, Germany. July, 2017

Abstract

The vehicular industry has exploited model-based engineering for design, analysis, and development of single-core vehicular systems. Next generation of autonomous vehicles will require higher computational power, which can only be provided by parallel computing platforms such as multi-core electronic control units. Current model-based software development solutions and related modelling languages, originally conceived for single-core, cannot effectively deal with multi-core specific challenges, such as core-interdependency and allocation of software to hardware. In this paper, we propose an extension to the Rubus Component Model, central to the Rubus model-based approach, for the modelling, analysis, and development of vehicular systems on multi-core. Our goal is to provide a lightweight transition of a model-based software development approach from single-core to multi-core, without disrupting the current technological assets in the vehicular domain.

Keywords. Model-based engineering, metamodeling, multi-core, vehicular domain, embedded systems, real-time systems.

10.1 Introduction

Software is ubiquitous in our society. In automotive, vehicles have transitioned from being mechanics-intensive to software-intensive systems [1]. For instance, the throttle control system of a modern vehicle is realised by means of Electronic Control Units (ECUs), sensors, and actuators, connected by several networks, and run by software, which replace the mechanical linkage between the accelerator pedal and the throttle. The current trend in the vehicular domain is towards vehicles capable of autonomously driving. While most of the current vehicular systems still employ single-core ECUs, the tendency is to switch to ECUs equipped with multi-core microprocessors. In fact, next generation vehicles, particularly autonomous ones, are expected to require higher computational power, which can only be provided by multi-core solutions.

On the one hand, the shift to multi-core impacts the way vehicular software is designed, analysed and developed. Current model-based solutions, specifically tailored to single-core, are not as effective when dealing with multi-core specific challenges, such as core-interdependency and allocation of software to hardware. On the other hand, the vehicular industry cannot prescind from the current technological assets for many reasons, among which:

Legacy. It has been estimated that up to 90% of the software in a new vehicle release can be reused from previous releases when using model-based engineering [2].

Organisation. Original Equipment Manufacturers (OEMs) suppliers define their technological assets based on decennial contracts with Tier-1 and Tier-2 suppliers. Changes to these assets shall not affect these contracts.

Certified run-time support. Functional safety [3] is paramount for the safety criticality in vehicles [4]. Current model-based solutions rely on certified development environments and real-time operating systems [5]. Typically, the certification process adds a development cost overhead between 25 and 100%, and it lasts for several years [6].

We have investigated the extension of Rubus [7], a commercial model-based approach for vehicular single-core systems, to multi-core with the intent of not disrupting the current vehicular technological assets related to it. Our hypothesis is two-fold. (H1) Abstraction provided by models and automation provided by model transformations can be a game changer in the development of multi-core applications. Abstraction permits to detach software functional modelling

from multi-core hardware modelling and software/hardware allocation modelling. Automation can support the developer in taking important decisions, such as how to allocate tasks to available cores in order to maximise a specific quality aspect [8]. (H2) A lightweight transition of a model-based approach from single-core to multi-core which does not affect critical aspects such as certified run-time support and lastingness of legacy applications is possible.

In model-based engineering, metamodels play a pivotal role as they define the set of available modelling entities and relationships for representing the software architecture and its quality attributes. Moreover, they enable automation via model transformations. However, it is essential that metamodels effectively prescribe the type system, the structure, and the behaviour of domain-specific applications [9]. In [10] we have discussed some modelling languages (among which Rubus Component Model) used for single-core vehicular applications and highlighted the issues arising when using them for modelling multi-core applications. In particular, existing structural hierarchies lack concepts for representing multi-core aspects (e.g., cores and partitions) and do not provide explicit support for core-interdependency and allocation of software to hardware.

In this paper, we propose an extension to the Rubus Component Model (RCM) [11], core of the Rubus approach, to support multi-core. This represents the first crucial step in the transition from single-core to multi-core. The contribution of the proposed extension is two-fold. We provide a modelling language able to prescribe type system, structure, and behaviour of multi-core applications (C1). In particular, the proposed extension comprises modelling elements for representing the software architecture, the hardware platform, and the software to hardware allocation. We ensure backward compatibility with legacy single-core applications modelled with RCM and do not entail any modification to the Rubus run-time layer, the Rubus Kernel (C2).

The remainder of the paper is structured as follows. Section 10.2 introduces RCM and motivates its selection as well as its extension. Section 10.3 presents a comparison between existing related approaches documented in the literature and our solution. Section 10.4 describes the proposed solution in all its constituents. Section 10.5 describes the application of the proposed solution to an industrial vehicular application. Section 10.6 and Section 10.7 discuss the benefits and limitations of our solution and conclude the paper, respectively.

10.2 The Rubus Component Model

There are several modelling languages used in the vehicular domain, such as RCM, AUTOSAR [12], ProCom [13], COMDES [14], AADL [15], to name a few. These languages were not conceived to deal with the complexity of predictable vehicle software specifically developed to run on multi-core platforms.

We focus on RCM and its extension for multi-core due to the following reasons. RCM is a good candidate to overcome the issues related to predictability thanks to its statically synthesised communication as well as its predictable and fine-grained execution model [16]. RCM uses pipe-and-filter communication and distinguishes between the control and data flows among its software components. In [17], we showed that these two features are central for providing early timing verification of the modelled system, e.g., by supporting end-to-end timing analysis [18]. Another reason for focusing on RCM is the small runtime footprint of the developed software (automatically generated from RCM models) as compared to other languages [17].

RCM is developed by Arcticus Systems AB¹ in collaboration with Mälardalen University. Through the years, RCM has been adopted by several OEM, Tier-1 and Tier-2 companies (e.g., Volvo Construction Equipment, BAE Systems Hägglunds, Hoerbiger and Knorr Bremse) for the development of embedded real-time software. RCM provides the Rubus Kernel, a dedicated real-time operating system, which is available for different processor architectures and certified according to the ISO 26262 [5] standard ASIL D (Road vehicle – Functional Safety) from Safety Integrity AB².

RCM was originally thought for providing modelling purposes, but it did not feature model-based mechanisms, i.e. automation in terms of model transformation. In order to achieve a full-fledge model-based approach, in [19] we reverse-engineered the RCM specification in order to express it in a more canonical form, a metamodel, which we called RubusMM. RubusMM included concepts for expressing software architectures and concepts for describing timing information of vehicular single-core applications. In this paper, we extend RubusMM to enable modelling of software applications for multi-core.

¹<https://www.arcticus-systems.com>

²<http://www.safetyintegrity.se>

10.3 Related Work

AUTOSAR [12] is an industrial initiative to provide a standardised software architecture for the development of vehicular software systems. Since the emergence of AUTOSAR 4.0, multi-core support is part of the standard. Similar to RCM, AUTOSAR describes the application software by means of self-contained units called software components which are mapped to the ECUs. At the application software level, AUTOSAR does not distinguish between the control and the data flows. In [17], we discussed how this feature is central for providing early timing verification of the modelled system. AUTOSAR does not provide means for modelling the execution platform [20]. Recently, several works on the use of AUTOSAR for multi-core have been proposed both from industry and academia. However, their main focus is on the adaptation of the AUTOSAR run-time support rather than on specific modelling challenges such as, e.g., allocation of the software components. In [21], the authors investigated the use of AUTOSAR for virtualised architecture and they identified some challenges on the use of AUTOSAR for multi-core. They concluded that additional features for the dynamic allocation of the software were needed. In [22] and [23], the authors evaluated AUTOSAR systems realised with a centralised architecture where the layered architecture was entirely allocated to one of the available cores only. Both the approaches were able to demonstrate that the behaviour of the multi-core software system and its footprint did not significantly vary from the corresponding single-core configuration. However, in both approaches, the uneven distribution of the workload among the cores led to performance and timing verification issues. In [24] and [25] the authors described AUTOSAR systems based on virtualised architectures where hypervisors coordinate multiple software systems with same or different real-time operating system(s). The use of hypervisors complicates early timing verification as it introduces additional complexity. From a footprint point of view, the virtualised architecture may lose its efficiency as each software system can carry a different real-time operating system. Both approaches rely on certified versions of AUTOSAR systems. In the AMALTHEA project [26], AUTOSAR standardised software architecture and methodology are used as a base for a development methodology aiming at reducing the effort in exchanging data.

Besides technologies specific to the vehicular domain, several works have discussed the use of the UML language and its profile MARTE [27]. Being general-purpose, these technologies are often used as complementary to domain-specific languages as, e.g., AUTOSAR and RCM. In [28], the authors present the VERTAF/Multi-core UML-based framework for the development

of multi-core software. Within VERTAF/Multi-core, the software system is described by means of UML class diagrams, timed state machines and sequence diagrams. Model transformations are used for generating extensions to these models for checking the viability of the design with respect to schedulability and conformance to the specifications. In [29] and [30] MARTE is used for representing the high-level architecture of the software system and as enabler for code-generation. In the first approach, UML is used for modelling the software components while MARTE is used for modelling hardware and software to hardware allocations. Starting from these models, code is automatically generated and timing verification through simulation is run. The second approach focuses on the system deployment of component-based systems. MARTE is used for modelling high level description models from which different models representing allocations of components are generated by means of code generation. In [8], MARTE is used for describing a task model and the allocations of tasks to cores for combined simulation- and execution-based task allocation optimisation. In [31] the authors introduce a MARTE-based framework, named GASPARD, for the design of parallel embedded systems. Herrera et al. [32] discuss a framework for the design space exploration of embedded systems based on MARTE. The framework, called COMPLEX, uses MARTE for describing the different architecture solutions composing the design space.

AADL [15] is an architecture description language developed for the avionic domain, but currently used for modelling embedded systems in general. Similarly to RCM, AADL provides multi-core support and a clear separation of concerns between software and hardware elements. However, unlikely to RCM, the software architecture is described at a lower level of abstraction in terms of, e.g., *Processes* and *Threads*.

10.4 Extending Rubus Component Model for Multi-core

In this section, we describe the extension to RCM for modelling vehicle software on multi-core. The extension is formalised by means of metamodelling. We compare the extended RCM with its previous definition, given in [19], thus highlighting differences and commonalities. The extension comprises the addition of modelling packages, classifiers, features, and relations as well as the modification of some hierarchical structures.

With respect to the previous definition, we have introduced packages for ensuring a better separation of concerns, improving the understandability of

the metamodel, and simplifying future extensions. The RubusMM packages involved in the extension are *RCM_COMMON*, *RCM_HW* and *RCM_SW*³. *RCM_HW* contains the elements for modelling the hardware platform: *Target*, *Allocator*, *Core*, and *Partition*. *RCM_SW* contains the elements for modelling the software architecture: *Allocatable*, *Mode*, *Assembly*, and *Software Circuit*. *RCM_COMMON* contains elements which are common to different packages as, for instance, *System* and port elements. Fig. 10.1 shows a fragment of RubusMM containing elements from *RCM_HW* for modelling the hardware platform. *System* represents the system under development. As all the el-

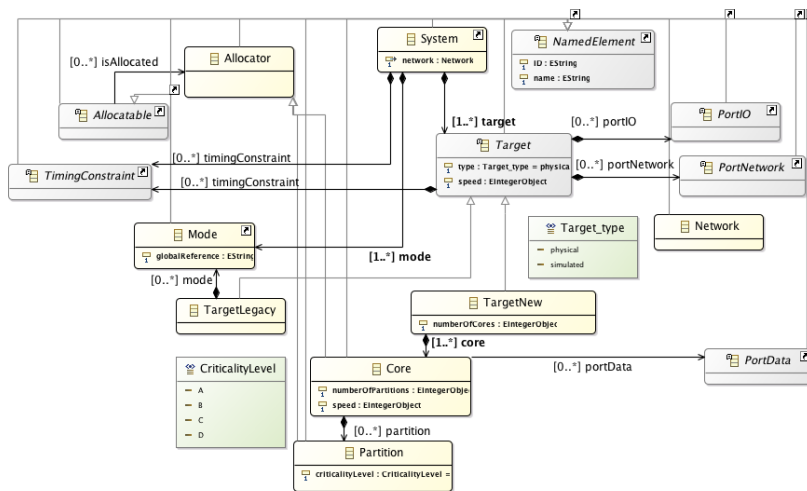


Figure 10.1: Fragment of the *RCM_HW* package for modelling the hardware platform.

elements in RubusMM, it inherits from the abstract metaclass *NamedElement* which provides two attributes: *name* and *ID*. We extended *System* with the reference *timingConstraint* for enabling the specification of timing constraints, occurrences and events which are used for timing verification.⁴ These constraints are used for running timing analysis, but we employed them for auto-

³The complete explanation of RubusMM is not in the scope of this work. The interested reader may refer to [19].

⁴*TimingConstraint* and other elements from different RCM packages are not part of this extension. However, they are put in relation to the extension as they contribute to a holistic view of the language and its peculiarities.

matically generating the set of RCM models satisfying a given set of timing requirements too [33].

A *System* contains one *Network*, one or more *Target* elements, and one or more *Mode* elements. A *Network* element models all the messages exchanged among the *Node* elements. It has two attributes, *protocol* and *speed*, which specify the protocol (e.g., Controlled Area Network (CAN) [34]) and the speed of the network in Kbit/s, respectively. A *Target* is a hardware-specific element which represents a processor architecture. The definition of *Target* has been extended with the references *timingConstraint*, *portIO*, and *portNetwork*. *portIO* and *portNetwork* model the peripherals and the inter-node communication, respectively.

In the previous definition of RubusMM, *Target* contained *Mode*, representing the software application. However, the containment relation between *Target* and *Mode* was too restrictive for modelling multi-core applications. Such a containment prescribed in fact that *Mode* elements, representing software, were structurally contained by hardware, represented by *Target* elements. Although not providing a clear separation between software and hardware, this structural containment suited the single-core case, since allocation of software to hardware was not variably split across different cores. Modelling for multi-core demanded more flexibility, since allocation of software to hardware is a variability point, which can hardly be represented by a structural containment.

In order to provide such a flexibility, while ensuring backward compatibility with legacy RubusMM models, we have modified the existing hierarchy as follows. We have added the metaclasses *TargetLegacy* and *TargetNew*, both inheriting from the abstract metaclass *Target*. *TargetLegacy* represents a legacy (single-core) ECU and it contains one or more *Mode* elements. This containment is specified through the reference *mode*. *TargetNew* represents a single- or multi-core ECU and contains one or more *Core* elements, which in turn can contain *Partition* elements. Both *Core* and *Partition* elements inherit from the abstract metaclass *Allocator*, representing hardware elements to which software elements, represented by the metaclass *Allocatable*, can be allocated. The metaclasses *Allocator* and *Allocatable*, together with the reference *isAllocated*, provide the flexible mechanism for the allocation of software to hardware that we needed, without any structural containment.

The metaclass *Target* provides the following attributes: *speed*, which specifies its speed in MHz, and *type*, which specifies whether it is a *physical* or a *simulated* target. A simulated target represents the simulation of the actual target processor in a host environment such as Windows or Linux.

Both *TargetLegacy* and *TargetNew* inherit *speed* and *type*. Moreover, *Tar-*

getNew provides additional multi-core specific attributes. *numberOfCores* specifies the number of cores composing the *TargetNew* and it is used by the model-based timing analysis and to automatically allocate software to hardware. The reference *core* links *Core* elements to their respective *TargetNew*. *Core* may contain *Partition* elements. The attribute *numberOfPartitions* specifies the number of partitions within a *Core* and the reference *partition* links them to the *Core*. The attribute *criticalityLevel* specifies the safety criticality level according to the ISO 26262 standard. There are four criticality levels (A to D) in this standard. A is the lowest criticality level, whereas D is the highest criticality level (the Rubus Kernel supports and is certified for all of them). Hence, the *Partition* element allows to develop multi-criticality software systems, where some parts of the software architecture are more critical than the others. *Target*, *TargetLegacy*, *TargetNew*, *Core*, *Partition*, *Allocator*, *Allocatable*, as well as their attributes and related references were not part of the previous RubusMM definition.

Fig. 10.2 shows a fragment of the RubusMM containing elements from the *RCM_SW* and the *RCM_COMMON* packages for modelling the software architecture. In RCM a software circuit, represented in RubusMM by *SWC*, is the

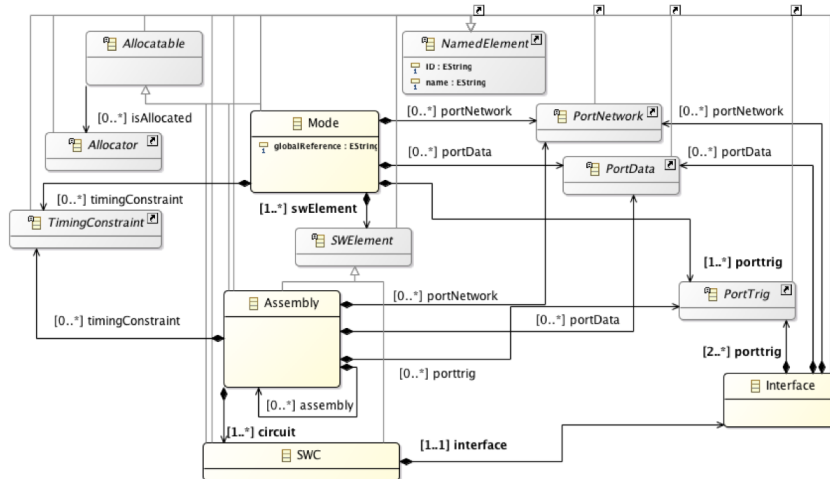


Figure 10.2: Fragment of the *RCM_SW* package for modelling the software architecture.

lowest-level hierarchical element that encapsulates basic software functions. A

SWC contains one *Interface* which groups all its ports. As RubusMM distinguishes between the data and control flows, an *Interface* contains *PortData* and *PortTrig* elements. The *PortData* elements manage the data communication among *SWC* deployed on the same *Target*. The *PortTrig* elements manage the activation of the *SWC* elements.

A *PortNetwork* is a port for the data communication of *SWC* elements deployed on different *Target* elements. The *PortData* elements of a *Core* are referenced to the *PortData* elements of the *SWC*s allocated on that *Core*. Similarly, the *PortNetwork* elements of a *Node* are referenced to the *PortNetwork* elements at *SWC* level. An *Assembly* groups *SWC* and *Assembly* elements in a hierarchical fashion.

Its reference *timingConstraint* enables the specification of timing constraints, occurrences and events which are used for timing verification. With respect to the previous definition, *SWC* and *Assembly* have been extended with the inheritance relation from the abstract metaclass *Allocatable*. A *Mode* groups *Assembly* and *SWC* elements and it is used for modelling a specific application of the software architecture (e.g., start-up or error mode). The attribute *globalReference* serves for creating a reference among all the *Mode* elements contributing to the same application. With respect to its previous definition, *Mode* has been extended with the inheritance relation from the abstract metaclass *Allocatable*. The metaclasses *Allocatable* and *Allocator* together with the reference *isAllocated* enable the specification of the allocation of software to hardware. More precisely, an *Allocatable* element can be deployed to an *Allocator* element by setting the *isAllocated* reference. *Allocatable*, *Allocator*, and related references were not part of the previous RubusMM definition.

10.5 Modelling the Brake-by-wire System

In this section, we leverage the extended RubusMM for modelling the Brake-by-wire (BBW) vehicular application. The BBW system is a stand-alone braking system equipped with an anti-lock braking (ABS) function, which allows to control the brakes through electronic means. To this end, it does not employ any mechanical connection between the brake pedal and the brake actuators. Fig. 10.3 depicts the block diagram of the BBW system.

A sensor, attached to the brake pedal, acquires the signal expressing the position of the pedal. The signal is sent to a computational unit which translates it into a brake torque. A sensor on each wheel acquires the signal expressing the speed of the wheel. The speed of each wheel, together with the computed

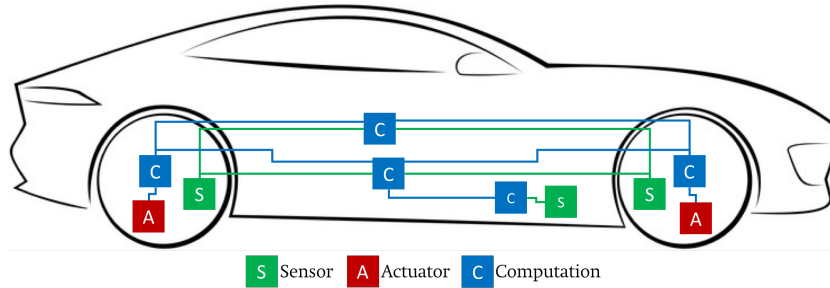


Figure 10.3: Block diagram of the BBW system.

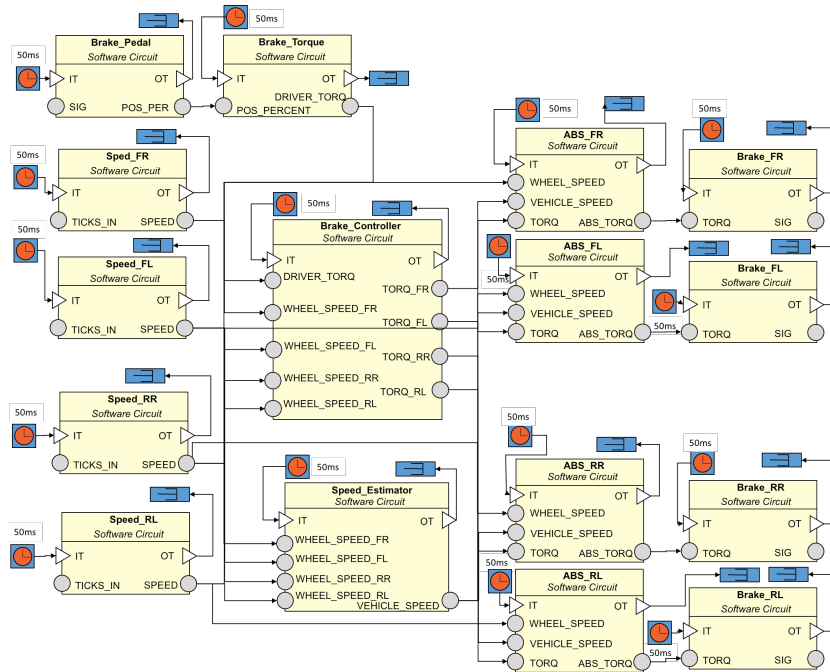


Figure 10.4: RubusMM model representing the software architecture of the BBW system.

brake torque, is sent to a computational unit which calculates the brake torque

for each wheel. Also, the speed of each wheel is sent to a computational unit which calculates the speed of the vehicle. The speed of the vehicle and the brake torque of each wheel are used from the ABS units for calculating the optimal brake torque for each wheel for avoiding locking the brakes. Finally, the actuators on the wheels produce the actual brake. Fig. 10.4 shows a RubusMM model depicting the software architecture of the BBW system.

The model consist of 16 software circuits where i) *Brake_Pedal* models the software operating the sensor on the brake pedal, ii) *Speed_FR*, *Speed_FL*, *Speed_RR*, and *Speed_RL* model the software operating the speed sensors on the wheels, iii) *Brake_Torque*, *Brake_Controller*, *Speed_Estimator*, *ABS_FR*, *ABS_FL*, *ABS_RR*, and *ABS_RL* model the software on the computational units and iv) *Brake_FR*, *Brake_FL*, *Brake_RR*, and *Brake_RL* model the software operating the actuators on the wheels.

In order to show how the extended RubusMM supports the modelling of multi-core applications (H1), while ensuring backward compatibility with legacy single-core applications (H2), we propose two different deployment configurations. In the first configuration, the BBW system is deployed to a MPC5744P microcontroller, which is a 32-bit uncore microcontroller designed for vehicular applications.

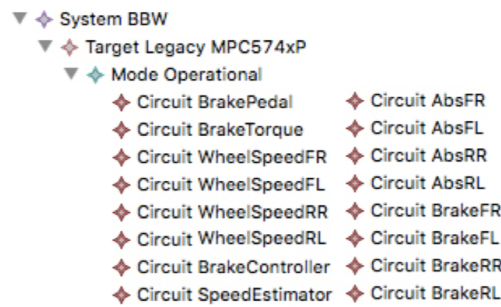


Figure 10.5: Serialisation of the BBW system deployed to a uncore microcontroller.

Fig. 10.5 shows an Ecore serialisation of such a configuration. Note that, according to what described in Section 10.4 regarding the modelling of legacy applications, the deployment on single-core is expressed leveraging the containment relation between the 'TargetLegacy' *MPC574xP* and the 'Mode' element *Operational*.

In the second configuration, the BBW system is deployed to an Infineon

SAK-TC299TP-128F300S BBmicrocontroller, which is a tri-core microcontroller developed for applications with high demands of performance and safety.

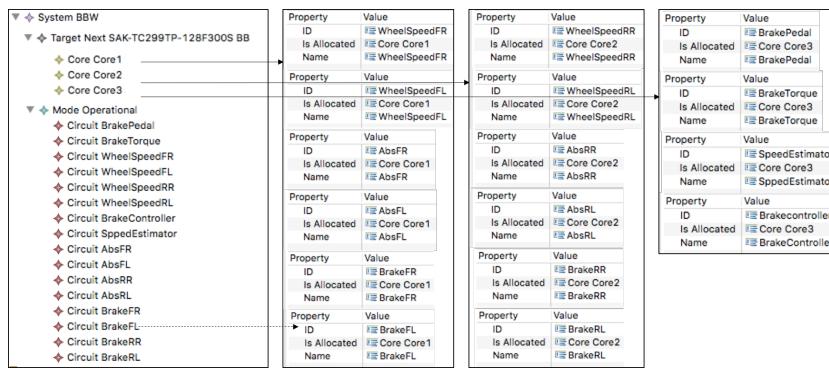


Figure 10.6: Serialisation of the BBW system deployed to a tri-core microcontroller.

Fig. 10.6 shows an Ecore serialisation of this configuration. In this case, the deployment information is modelled by means of the 'isAllocated' reference expressed between 'Allocatable' and 'Allocator' elements. More precisely, the software circuits modelling the sensors, the computation units and the actuators of the two front wheels (*WheelSpeed_FR*, *WheelSpeed_FL*, *Abs_FR*, *Abs_FL*, *Brake_FR*, *Brake_FL*) are allocated to *Core 1* of the *SAK-TC299TP-128F300S BB* target, as shown by the arrow in the top-right corner of Fig. 10.6. Similarly, the SWCs modelling the sensors, the computation units and the actuators of the two rear wheels (*WheelSpeed_RR*, *WheelSpeed_RL*, *Abs_RR*, *Abs_RL*, *Brake_RR*, *Brake_RL*) are allocated to *Core 2* of the *SAK-TC299TP-128F300S BB* target. The remaining SWCs modelling the computational units are allocated to *Core 3* of the *SAK-TC299TP-128F300S BB* target. As discussed in Section 10.4, the extended RubusMM leverages a clearer separation of concerns between software and hardware elements as well as an explicit and more flexible allocation mechanism. Let us suppose that the allocation specified in Fig. 10.6 does not satisfy a given set of fault-tolerance requirements. One way of addressing this would be to model a lockstep [35] configuration of the BBW system where each core runs a copy of the complete software, in parallel. In order to model such an allocation with the extended RubusMM, it is sufficient to allocate all software circuits composing a 'Mode' to each single 'Core'.

10.6 Lesson Learned

In this paper, we have proposed an extension to RCM for modelling next generation of vehicular multi-core systems (H1). The main challenge faced during the extension of RCM was how to introduce the new modelling elements without affecting the lastingness of legacy RCM applications (H2). In the first definition of RCM, pragmatic choices for more efficient modelling and analysis of single-core applications were made when defining the language. In addition to not providing clear separation of concerns between hardware and software, these choices complicated the extension of RCM, as in the case of the containment relation between *Target* and *Mode* discussed in Section 10.4. In fact, that structural containment, although dramatically simplifying model navigation for analysis and code generation purposes in case of single-core applications, did not suit variability of software to hardware allocation in the multi-core case. In this respect, the proposed extension prescribes an allocation mechanism which is more flexible and apt to be automated by means of model transformations. Please note that, we have previously provided RubusMM with support for variability modelling [36]. This feature can be very valuable for representing sets of allocations of software components to multiple cores, all in a single model with variability points representing allocations.

To maximise backward compatibility, we introduced the new modelling elements as leaves in the metamodel hierarchy, as in the case of, e.g., *Core* and *Partition*. This choice could demand additional modelling effort as the engineer can be required to model the entire hierarchy in order to design valid models from scratch. This can be mitigated by tooling features, allowing the modeller to directly model a leaf, while automatically generating the path to the model root populated with a set of default values.

In Section 10.2, we have pointed out early timing verification as one of the main reasons which made RCM very appreciated in the vehicular domain and its extension for multi-core compelling. In this respect, when extending RCM, we have explicitly addressed timing verification by allowing the specification of timing constraints, occurrences and events at several levels of the structural hierarchy by means of the references *timingConstraint*. This ensures full compatibility with the existing model-based timing analysis provided by Rubus. Moreover, it enables the use of the most recent timing analysis for vehicular embedded systems on multi-core [37]. Without the extension provided in this paper, the timing analysis for multi-core would not have been possible in Rubus due to the missing structural and timing information.

Functional safety is paramount for the safety criticality of vehicular sys-

tems. For being adopted in the vehicular domain, model-based solutions must provide certified run-time support, e.g., real-time operating system, along with modelling languages able to capture all the characteristics of a vehicular application. The Rubus Kernel is certified according to the ISO 26262 standard ASIL D while Rubus ICE (i.e., the development environment supporting Rubus) is undergoing the same certification. In this respect, we have extended RCM according to the virtualisation design option, as described in [38], which enables the reuse of the certified Rubus Kernel. On the one hand, the reuse of the Rubus Kernel makes also the explicit modelling of the memory not necessary since the mapping of data ports to physical memory is handled by the Rubus Kernel itself. On the other hand, this makes the current definition of RCM not suited for approaches where explicit modelling of the memory is pivotal. Moreover, despite the Rubus Kernel footprint is significantly small, the virtualised design option increases the overall footprint of the developed vehicular application since each core or partition can host a separate instance of the Rubus Kernel.

10.7 Conclusion and Future Work

In this paper, we have discussed the extension of the Rubus Component Model for modelling vehicular multi-core applications while ensuring backward compatibility with legacy single-core applications. The proposed extensions also support the modelling of multi-criticality applications on single- as well as multi-core platforms. We have leveraged an industrial vehicular application to validate the proposed extension, also in terms of backward compatibility.

One line of future work will investigate how to support the analysis and verification of vehicular embedded systems with multi-criticality levels on multi-core with respect to predictable timing behaviour. Moreover, we will investigate how to adapt the certified Rubus Kernel for providing run-time support to these systems on multi-core. Another line of future work will investigate how to provide automatic support for the allocation of software to hardware. In particular, we are developing model transformations that, starting from a model with no modelled allocations and a set of timing constraints, produce a set of models featuring the set of different allocations of software to hardware optimised for satisfying the set of timing constraints. We are planning to represent the set of generated models by means of the compact notation presented in [36]. Such a notation uses modelling with variability for representing a multitude of models with one single model with variability points.

Acknowledgments

The work in this paper is supported by the Swedish Knowledge Foundation (KKS) through the PreView and MOMENTUM projects, and by the Swedish Research Council (VR) through the SynthSoft project. We thank our industrial partners Arcticus Systems, Volvo Construction Equipment and BAE Systems Hägglunds, Sweden.

Bibliography

- [1] Robert N Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.
- [2] Peter Thorngren. keynote talk: Experiences from east-adl use. In *EAST-ADL Open Workshop, Gothenberg*, 2013.
- [3] David Smith and Kenneth Simpson. *Functional safety*. Routledge, 2004.
- [4] Neil R Storey. *Safety critical computer systems*. Addison-Wesley Longman Publishing Co., Inc., 1996.
- [5] ISO 26262-1:2011: Road Vehicles in Functional Safety. <http://www.iso.org/>.
- [6] Paul Pop, Detlef Scholle, Hans Hansson, Gunnar Widforss, and Malin Rosqvist. The SafeCOP ECSEL Project: Safe Cooperating Cyber-Physical Systems Using Wireless Communication. In *Digital System Design (DSD), 2016 Euromicro Conference on*, pages 532–538. IEEE, 2016.
- [7] Rubus ICE-Integrated Development Environment. <http://www.arcticus-systems.com>.
- [8] Federico Ciccozzi, Juraj Feljan, Jan Carlson, and Ivica Crnković. Architecture optimization: speed or accuracy? both! *Software Quality Journal*, pages 1–24, 2016.
- [9] D C Schmidt. Guest Editor’s Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.
- [10] Saad Mubeen and Alessio Bucaioni. Modeling of Vehicular Distributed Embedded Systems: Transition from Single-core to Multi-core. In *14th International Conference on Information Technology : New Generations*. Springer, April 2017.

- [11] 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.
- [12] AUTOSAR Technical Overview, Version 4.3, The AUTOSAR Consortium, Dec., 2016. <http://autosar.org>.
- [13] Sverine Sentilles, Aneta Vulgarakis, Tomas Bures, Jan Carlson, and Ivica Crnkovic. A Component Model for Control-Intensive Distributed Embedded Systems. In *11th International Symposium on Component Based Software Engineering (CBSE), 2008*, pages 310–317. Springer, October 2008.
- [14] Xu Ke, K. Sierszecki, and C. Angelov. COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2007*, pages 199–208, August 2007.
- [15] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (AADL): An introduction. Technical report, DTIC Document, 2006.
- [16] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture*, 60(2):207–220, 2014.
- [17] Saad Mubeen, Thomas Nolte, Mikael Sjödin, John Lundbäck, and Kurt-Lennart Lundbäck. Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints. *Software & Systems Modeling*, pages 1–31, 2017.
- [18] 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.

-
- [19] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Saad Mubeen, and Mikael Sjödin. A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation from EAST-ADL. *Journal of IEEE Access*, 5(1), December 2016.
- [20] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded system design for automotive applications. *Computer*, 40(10), 2007.
- [21] Dominik Reinhardt, Dirk Kaule, and Markus Kucera. Achieving a scalable e/e-architecture using autosar and virtualization. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, 6(2013-01-1399):489–497, 2013.
- [22] Gary Morgan and Andrew Borg. Multi-core automotive ECUs: Software and hardware implications. Technical report, ETAS Group, Tech. Rep, 2009.
- [23] Niko Böhm, Daniel Lohmann, and Wolfgang Schröder-Preikschat. A Comparison of Pragmatic Multi-core adaptations of the AUTOSAR system. In *7th annual Workshop on Operating System Platforms for Embedded Real-Time Applications (OSPRT)*, pages 16–22, 2011.
- [24] Dominik Reinhardt and Markus Kucera. Domain Controlled Architecture-A New Approach for Large Scale Software Integrated Automotive Systems. *PECCS*, 13:221–226, 2013.
- [25] Dominik Reinhardt and Gary Morgan. An embedded hypervisor for safety-relevant automotive E/E-systems. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, pages 189–198. IEEE, 2014.
- [26] AMALTHEA Project Profile, Apr., 2017. <http://www.amalthea-project.org>.
- [27] The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, 2010. OMG Group, January 2010.
- [28] Chao-Sheng Lin, Pao-Ann Hsiung, Chih-Hung Chang, Nien-Lin Hsueh, Chorng-Shiuh Koong, Chih-Hsiung Shih, Chao-Tung Yang, and William C-C Chu. Model-Driven Multi-core Embedded Software Design. 2011.

- [29] Federico Ciccozzi, Tiberiu Seceleanu, Diarmuid Corcoran, and Detlef Scholle. UML-Based Development of Embedded Real-Time Software on Multi-Core in Practice: Lessons Learned and Future Perspectives. *IEEE Access*, 4:6528–6540, 2016.
- [30] Alejandro Nicolas, Hector Posadas, Pablo Peñil, and Eugenio Villar. Automatic deployment of component-based embedded systems from UML/MARTE models using MCAPI. In *Design of Circuits and Integrated Circuits (DCIS), 2014 Conference on*, pages 1–6. IEEE, 2014.
- [31] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Rabie Ben Atitallah, Anne Etien, Philippe Marquet, and Jean-Luc Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(4):39, 2011.
- [32] Fernando Herrera, Héctor Posadas, Pablo Peñil, Eugenio Villar, Francisco Ferrero, Raúl Valencia, and Gianluca Palermo. The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems. *Journal of Systems Architecture*, 60(1):55–78, 2014.
- [33] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, and Mikael Sjödin. Anticipating Implementation-Level Timing Analysis for Driving Design-Level Decisions in EAST-ADL. In *International Workshop on Modelling in Automotive Software Engineering*, September 2015.
- [34] ISO 11898-1. Road Vehicles Interchange of Digital Information Controller Area Network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.
- [35] Stefan Poledna. *Fault-tolerant real-time systems: The problem of replica determinism*, volume 345. Springer Science & Business Media, 2007.
- [36] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Saad Mubeen, Mikael Sjödin, and Alfonso Pierantonio. Handling Uncertainty in Automatically Generated Implementation Models in the Automotive Domain. In *42nd Euromicro Conference series on Software Engineering and Advanced Applications*, September 2016.
- [37] A. Burns and R. Davis. Mixed Criticality Systems - A Review, eighth edition. Technical report, Dept. of Computer Science, University of York, 2016. <https://www-users.cs.york.ac.uk/burns/review.pdf>.

- [38] Matthias Becker, Dakshina Dasari, Vincent Nélis, Moris Behnam, Pinho Luís Miguel, and Thomas Nolte. Investigation on AUTOSAR-Compliant Solutions for Many-Core Architectures. In *18th Euromicro Conference on Digital System Design*, volume 18, August 2015.

Chapter 11

Paper E: A Model-based Approach for Vehicular Systems

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

MRTC Report MDH-MRTC-321/2017-1-SE. Västerås, Sweden. December, 2017. Submitted for journal publication.

Abstract

This paper introduces a novel model-based approach for the software development of vehicular embedded systems. The proposed approach discloses the opportunity of improving efficiency of the development process by providing support to identify viable design solutions with respect to selected non functional requirements. To this end, it leverages the interplay of two modelling languages for the vehicular domain whose integration is achieved by a suite of model transformations. An instantiation of the methodology is discussed for timing requirements, which are among the most critical ones for the development of vehicular systems. The applicability of the methodology is demonstrated as proof of concepts on industrial use cases performed in cooperation with our industrial partners.

Keywords – Model-driven development; vehicular embedded systems; EAST-ADL; component model; model transformations.

11.1 Introduction

As vehicles transitioned from being mechanical-intensive to software-intensive systems [1], a cost-effective software development became paramount in the vehicular domain. Researchers and practitioners agreed that *abstraction* and *automation*, the founding pillars of Model-Driven Engineering (MDE), could be game changers towards the achievement of a cost-effective software development process as they contribute to shorten the development time and employ expensive resources more efficiently [2]. To this end, several domain-specific modelling languages were introduced both for designing the vehicular software and for representing its non functional properties such as timing. Among other languages, the Electronics Architecture and Software Technology Architecture Description Language (EAST-ADL) has been developed by the automotive industry to support modelling of vehicular functions and both their software realisation and desired non functional properties [3]. Within EAST-ADL, the system modelling is performed at four different levels of abstraction which are vehicle-, analysis-, design- and implementation-level (from highest to lowest abstraction level). The requirements on the vehicle functionality of the system are captured at the vehicle level. At the analysis level, the system is defined in terms of abstract functional architecture with a provision for high-level analyses. Typically, the vehicular software is modelled at the design level by means of function, hardware and allocation models. At the implementation level, the design models are enriched with detailed execution information on e.g., timing (worst-case execution time, etc.). The implementation models are defined by means of other languages, such as the AUTomotive Open System ARchitecture (AUTOSAR) [4] or the Rubus Component Model (RCM) [5]. Often, implementation models are used as the base for code synthesis. However, support to models integration (e.g., among EAST-ADL and RCM models) in the development of vehicular embedded systems is still scarce and the translation from design- to implementation-level is mainly performed manually. Too often, this lack of automation defers the verification of non functional properties to the last phases of the development process. Empirical research shows that modifications during these phases can be 40 times more expensive than the same modifications done during the design of the software and can introduce inconsistencies if they are not properly back propagated [6].

In this context, our hypothesis is that providing automation for model integration would enable early verification of non functional requirements (e.g. timing requirements) during the design of vehicular embedded systems thus improving the cost-efficiency of their development. In fact, early verification

of non functional requirements would limit the need for expensive modifications on the almost ready-to-deliver software, and automation would reduce the overall development time as well as enhance the use of expensive (engineering) resources.

In this paper, we propose a solution for early verification of non functional requirements by introducing MoVES, a model-driven methodology for the development of distributed vehicular real-time embedded systems on single- and multi-core platforms. Considering the importance of timing in the development of vehicular real-time systems, as acknowledged by several international projects and industrial initiatives (TIMMO-2-USE¹ and AUTOSAR²), the proposed methodology is instantiated to support the development and architectural exploration of system-designs with temporal awareness. MoVES leverages the interplay of EAST-ADL and RCM for expressing functional and implementation models, respectively. Moreover, it features a fully automated mechanism defined in terms of six different model transformations that describe precise relationships between EAST-ADL and RCM. In particular, starting from the EAST-ADL function and hardware models, model transformations generate a set of RCM models. Model transformations automatically generate allocation information on the RCM models from the EAST-ADL allocation model, too. As there might be multiple implementation models for the same design, a source EAST-ADL model cannot be univocally translated into a single target RCM model [7]. Currently, most approaches only consider one particular model out of the many possible alternatives [8]. In this work, we leverage the properties of a constraint-based transformation language, the Janus Transformation Language [9] (JTL), to automatically derive all the possible RCM models entailing meaningful and unique timing and allocation configurations. Timing analysis is run on the generated RCM models. Eventually, model transformations propagate the generated RCM models and the related timing verification results to the design level for enabling timing-aware design decisions. It is important to note that, the process of generating and analysing RCM models is transparent to the engineer and can be guided by means of logic constraints. Moreover, the engineer does not have to manually define or investigate RCM models, but rather select the preferred RCM models from the set of the automatically generated ones. We validated MoVES through a set proof of concepts conducted in tight cooperation with our industrial partners in the automotive domain. These showed promising results in terms of i) applicability of the methodology and ii) reduction of late modifications at implementation level.

¹<https://itea3.org/project/timmo-2-use.html>

²<https://www.autosar.org>

The main scientific contributions brought by MoVES are:

- a mechanism for the automatic translation of design models into implementation models,
- a mechanism for the automatic allocation of software to hardware, and
- a mechanism for the back-propagation of the verification results and related implementation models to design models.

The rest of the paper is organised as follows. Section 11.2 sets the background for this research work along with its contributions and relations with authors' previous work. Section 11.3 describes the methodology and its constituents. Section 11.5 describes the application of the methodology on a running example mimicking industrial settings. Section 11.6 discusses strengths and weakness of the proposed methodology. Section 11.7 describes related approaches documented in the literature and compares them to our solution. Finally, Section 11.8 concludes the paper.

11.2 Background

MDE is a discipline which aims at improving software development by employing abstraction and automation by using models, metamodels and model transformations [2]. Metamodels formalise the requirements, the structure and the behaviour of software systems within a particular domain. Models allow to design software systems declaratively using the elements and the concepts formalised by the metamodels, thus using constructs pertaining to the problem domain rather than constructs pertaining to the underlying technology. Model transformations are automatic means for analysing models and for synthesising new artefacts (models, source code, etc.) from a set of source models [10]. In the automotive domain, as vehicle transitioned from being mechanical- to software-intensive systems [11], MDE has gained industrial recognition as demonstrated by several international initiatives and projects, such as EAST-ADL [3], RCM [5] and AUTOSAR [4].

In the followings, we describe the background of this research work and its contribution in terms of the modelling languages and the model-based timing analysis leveraged for the definition of MoVES. In particular, in Section 11.2.1 and in Section 11.2.2 we introduce and describe the main elements of the EAST-ADL and RCM languages, respectively. In Section 11.2.3 we discuss

the leveraged timing analysis, while in Section 11.2.4 we detail the contributions presented in this paper and put them in relation to the authors' previous work.

11.2.1 EAST-ADL

EAST-ADL is a modelling language which captures the essentials of vehicular Electrical and Electronic (E/E) systems concerning their documentation, design, analysis and synthesis. EAST-ADL is specified through ten different packages, each of which addresses different aspects of vehicular E/E system and their development. In the proposed instantiation of the methodology, we leverage specific concepts from the *structure*, *requirements* and *timing* packages³.

The structure package serves for the specification of the software architecture in terms of basic elements and interactions among them. In order to ensure separation of concerns through the development process, the structure package makes use of four abstraction levels, which are *vehicle*, *analysis*, *design* and *implementation*. However, such a separation is only conceptual and some modelling elements can span over several abstraction levels. MoVES connects to the design level and more specifically to the *FunctionalDesignArchitecture*, *HardwareDesignArchitecture* and *Allocation* concepts.

Within EAST-ADL, a *FunctionalDesignArchitecture* describes how software functions interact. At this level, software functions are represented by means of *DesignFunctionPrototype* elements linked by *FunctionConnector* elements. A *DesignFunctionPrototype* is typed to a *DesignFunctionType* element which specifies its interface, in terms of *FunctionPort* elements, and its inner architecture, in terms of additional *DesignFunctionPrototype* elements. Within EAST-ADL, a *HardwareDesignArchitecture* describes the physical architecture of the vehicular embedded system. The basic modelling entity is the *HardwareComponentPrototype* which is typed to a *HardwareComponentType*. The *HardwarePortConnector* elements model the communication between two or more *HardwareComponentPrototype* elements by connecting the related *HardwarePort* elements. An EAST-ADL Allocation model consists of a set of *FunctionAllocation* elements binding *AllocatableElement* to *AllocationTarget* elements. *AllocatableElement* is an abstract superclass which specifies the elements that can be allocated. *DesignFunctionPrototype* and *FunctionConnector* are defined as extensions of the *AllocatableElement* superclass.

³Please note that the complete explanation of EAST-ADL is outside the scope of this work. The interested reader can refer to [3].

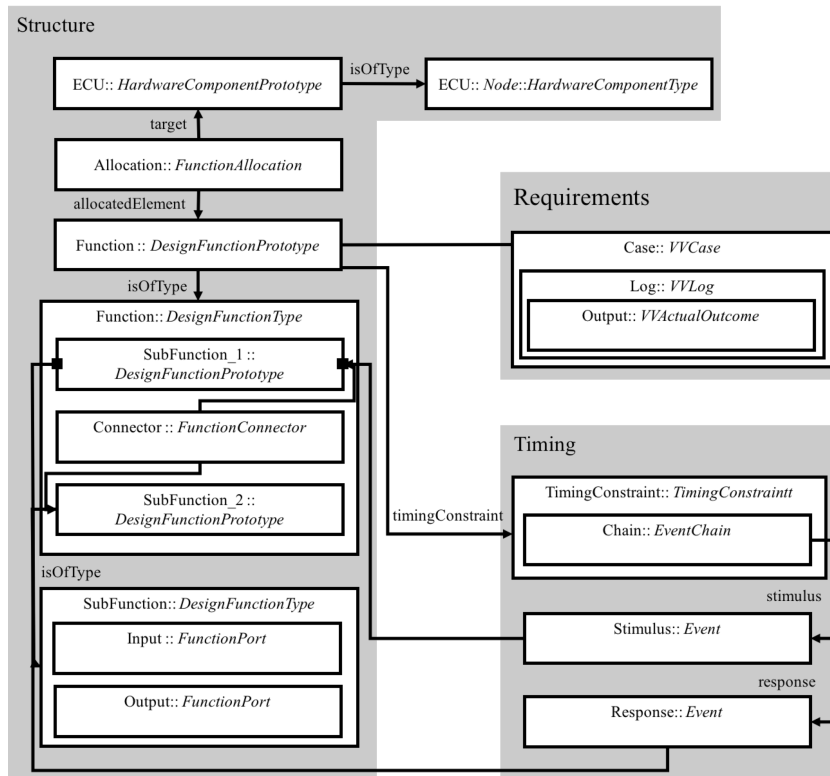


Figure 11.1: Simplified EAST-ADL model containing concepts from the EAST-ADL structure, timing and requirements packages

Similarly, AllocationTarget is an abstract superclass which specifies to which elements an AllocatableElement can be allocated. HardwareComponentPrototype and HardwarePortConnector elements are defined as extensions of the AllocationTarget class. Within the proposed methodology, the concepts from the function, hardware and allocation models are used by the model transformations for the automatic generation of RCM models, as described later.

In this work, we show an instantiation of MoVES focusing on the verification of timing requirements. Therefore, let us see how the timing requirements and properties are modelled within EAST-ADL. The EAST-ADL timing package contains concepts for modelling the timing constraints stemming from

the extra-functional requirements. Within EAST-ADL, a timing constraint is modelled by means of *TimingConstraint* elements, which are associated to *DesignFunctionPrototype* elements. The specification of the timing constraints is realised using two *Event* elements, which mark the scope of the related timing constraint and are contained within an *EventChain* element. The *EventChain* and *Event* elements are used by the proposed methodology for the specification of automatically generated timing constraints in the RCM models.

The requirement package offers means for describing the properties that the vehicular embedded system has to possess and their verification. To this end, the requirement package is further divided into two sub-packages which are *UseCases* and *VerificationValidation*. MoVES leverages specific concepts from the latter only. Within the *VerificationValidation* package, the *VVCase* elements represent concrete test activities which are associated to the *DesignFunctionPrototype* elements. A *VVCase* is modelled in terms of the *VVProcedure* and an *VVLog* elements which represent the adopted verification and validation technique and its description, respectively. A *VVLog* element is modelled in terms of a *VVActualOutcome* element, which specifies the actual output of the verification and validation activity. The proposed methodology uses the *VVLog* elements for back propagating the RCM models together with their timing verification results to the related EAST-ADL model.

Figure 11.1 shows a simplified EAST-ADL model represented as a block diagram using the EAST-ADL concepts discussed above. The model depicts a *DesignFunctionPrototype* called *Function*, which is allocated to a *HardwareComponentPrototype* called ECU. The *Function* and ECU elements are typed to the related *DesignFunctionType* and *HardwareComponentType*, respectively. Accordingly, ECU is an atomic node while *Function* is composed by two sub-functions called *SubFunction_1* and *SubFunction_2* connected via a *FunctionConnector* called *Connector*. Additionally, *Function* is associated with a *VVCase* called *Case* and a timing constraint called *TimingConstraint*.

11.2.2 RCM

RCM is a modelling language for the predictable development of resource-constrained embedded real-time systems developed by Arcticus Systems⁴ in collaboration with Mälardalen University. With respect to the EAST-ADL structural abstraction levels, RCM acts at the implementation level and it is currently used by several OEM, Tier-1 and Tier-2 companies (e.g., Volvo Construction Equipment, BAE Systems Hägglunds, Hoerbiger and Knorr Bremse)

⁴<https://www.arcticus-systems.com>

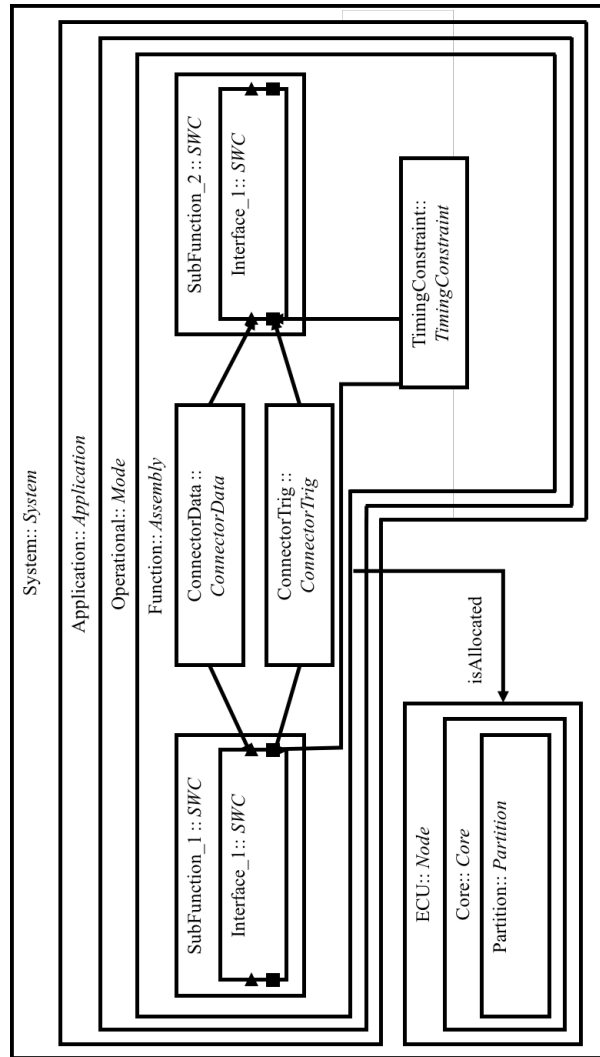


Figure 11.2: Simplified RCM model

in the vehicular domain. In its current definition, RCM provides support for the modelling of the software architecture, the execution platform, the allocation

information and the timing properties of vehicular embedded systems⁵ [12]. Within RCM, the embedded software architecture is modelled by means of the *Software Function* (SWC) elements and interactions among them. An SWC is the lowest-hierarchical element, which encapsulates basic software functions, and is defined by a *Behaviour* and an *Interface* elements. The Interface element is responsible for grouping the ports of a SWC. As RCM distinguishes between data and control flows among SWCs, an interface element contains two kinds of port: data and control. The interactions among SWCs are modelled by means of *Connector* elements. SWCs can be grouped in *Assembly* elements for constructing the software architecture in a hierarchical manner. SWCs and Assembly elements are contained in *Mode* elements, which are means for distinguishing different states or conditions in a system. For example, a system can execute the start-up mode when bootstrapping and afterwards shifts to the operational mode. Mode elements are contained within *Application* elements, which represent independent software functionalities of the system. Application elements provide means for isolating different software functionalities as well as for specifying the safety-criticality level in accordance to the ISO 26262 standard for the functional safety of road vehicles [13]. We refer to the RCM software models as the RCM models which contain the modelling elements for representing the software architecture, only. As the main goal of RCM is to provide support for the development of predictable vehicular embedded systems, timing properties and constraints are pivotal in the language, and they can be specified at different hierarchical levels of the software architecture (Application, Mode, Assembly). Timing constraints are modelled by *Timing Constraint* elements which are specified on the data ports of the related software element. Within RCM, the execution platform of the vehicular embedded system under development is modelled in terms of *Node*, *Core* and *Partition* elements. A Node element models the specific processor architecture and defines a unique run-time environment for the software architecture. A Node element contains one or more Core elements, which model the processing or computing unit of a Node element. Similarly, Core elements can contain one or more Partition elements, which represent the logical division of a Core elements into multiple computing resources. We refer to the RCM execution platform models as the RCM models which contain the modelling elements for representing the execution platform, only. Allocation information is modelled by means of the *isAllocated* relation specified between any two *Allocatable* and *Allocator* elements. Allocatable is an abstract superclass which specifies

⁵Please note that the complete explanation of the RCM language is outside the scope of this work. The interested reader can refer to [12].

the RCM software elements that can be allocated. Application, Mode, Assembly and SWC elements are defined as extensions of the Allocatable superclass. Similarly, Allocator is an abstract superclass which specifies to which execution platform element an Allocatable element can be allocated. Node, Core and Partition elements are defined as extensions of the AllocationTarget class.

Figure 11.2 shows a simplified RCM model represented as a block diagram using the RCM concepts described above. In particular, the model depicts a System called *System* which consists of a Node called *ECU* and an Application called *Application*. ECU is modelled as a single-core and single-partition node. Application is modelled by means of a Mode called *Operational* which contains an Assembly called *Function*. The internal architecture of the Assembly consists of two SWCs, called *SubFunction_1* and *SubFunction_2*, connected by two Connector elements, called *ConnectorData* and *ConnectorTrig*, for the data and the control flows, respectively. A TimingConstraint element is specified between the data output port of *SubFunction_1* and the data input port of *SubFunction_2*.

11.2.3 Timing Analysis

Many vehicular embedded systems are constrained by stringent timing requirements that must be satisfied during their development. End-to-end timing analysis is a well-established technique to verify the timing requirements that are specified on these systems. Such an analysis must be integrated to the tool chain that is used for the model- and component-based development of these systems. In order to support the timing analysis an appropriate system view, called end-to-end timing model, should be extracted from the software architecture of the system under analysis. The end-to-end timing model consists of two models, namely timing model and linking model. The timing model includes the timing properties (e.g., priorities, periods, worst- and best-case execution times, offsets and jitter) and timing requirements (e.g., deadlines and delay constraints) regarding all tasks, messages and task chains in the distributed embedded system. On the other hand, the information about links, dependencies, control flows (activation information) and data flows (information regarding data exchanges) among tasks and messages in all task chains are captured in the linking model. For example, consider a task chain shown in Figure 11.3. The chain is distributed over three nodes that are connected by a network. The system timing model includes all the timing information (discussed above) in the three nodes and the network. Whereas, the linking model includes all the linking information (discussed above) in the chain that initiates

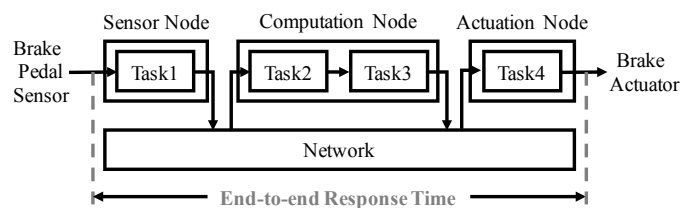


Figure 11.3: Example showing end-to-end response time

at Task1 in the Sensor Node and terminates at Task4 in the Actuation Node. It should be noted that the end-to-end timing model discussed above is in line with the classical timing model for distributed embedded system [14, 15, 16]. The analysis engines use the end-to-end timing model to analyse the timing behavior of the system. In this paper we consider the end-to-end timing analysis given in [15, 17]. The analysis has been implemented in several industrial tools, e.g., [17]. The analysis results consist of response times of tasks and messages as well as system utilization. The analysis also 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 sensor input at the sensor node and the corresponding response, e.g., the brake actuation signal at the actuation node as shown in Figure 11.3. If the tasks in a chain are activated independently (e.g., by periodic clocks) then various types of end-to-end delays must also be computed to verify the timing behavior of the system. *Age* and *Reaction* are two such delays that are commonly found in vehicular embedded systems. The age delay in a task chain corresponds to the freshness of the data that is available at the output of the chain. This delay finds its importance in the control systems domain in vehicles. Whereas, the reaction delay in a task chain corresponds to the first reaction at the output of the chain for a given stimulus at the input of the chain. This delay finds its application in the body electronics domain in vehicles. In order to explain the age and reaction delays, consider a task chain in a single-node system as shown in Figure 11.4.

The chain consists of two tasks, namely τ_1 and τ_2 . The tasks are activated by independent clocks of periods 25 milliseconds and 5 milliseconds respectively. Assume that the Worst-Case Execution Times (WCETs) of these tasks are 2 milliseconds and 1 millisecond respectively. Task τ_1 reads data from register Reg-1 and writes data to Reg-2. Similarly, task τ_2 reads data from Reg-2

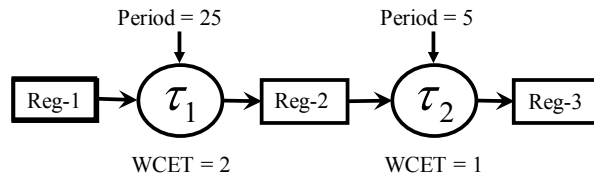


Figure 11.4: A task chain with independent activations of tasks

and writes data to Reg-3. Since, the tasks are activated independently with different clocks, there can be multiple outputs (Reg-3) corresponding to any single input (Reg-1) to the chain as shown by several uni-directional arrows in Figure 11.5.

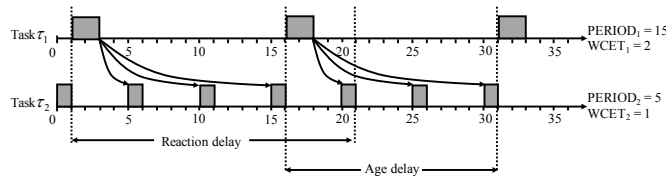


Figure 11.5: Example showing end-to-end delays

The age and reaction delays are also identified in Figure 11.5. These delays are equally important in distributed embedded systems.

11.2.4 Paper Contributions in Relation with Authors' Previous Work

In this work, we present MoVES, a model-driven methodology for real-time distributed vehicular systems on single- and multi-core, supporting the development and architectural exploration of system designs with temporal awareness. The methodology leverages the interplay of EAST-ADL and RCM and consists of a fully automated mechanism defined in terms of a set of model transformations. RCM was originally thought for providing modelling purposes, but it did not feature a canonical definition of the language in terms of a metamodel. In [18], we reverse-engineered the RCM language and presented a preliminary metamodel definition of the RCM core elements. In [19], we provided a complete metamodel definition for RCM for modelling and timing

analysis of vehicular embedded systems on single-core. In [12] we extended the RCM metamodel definition for the development of vehicular systems on multi-core. This extension introduced modelling elements for the representation of the execution platform and the allocation information. In this paper, we leverage an even more enhanced version of the RCM metamodel definition given in [12], which includes the concept of application. Early timing verification is paramount in the vehicular domain. To this end, in [20] we proposed a mechanism for the exploitation of RCM timing capabilities at EAST-ADL design level. Such a mechanism leveraged the RCM metamodel definition given in [19] and consisted of a single model transformation for the generation of the RCM software architecture and timing properties from an EAST-ADL model, only. Moreover, the mechanism in [19] can not be applied for the development of vehicular embedded systems on multi-core. In this paper, by exploiting the new modelling capabilities of RCM, we introduce a methodology for the development of real-time distributed vehicular embedded systems on single- and multi-core, which guides the engineer to viable solutions with respect to timing requirements. To this end, the presented methodology leverages a refined version of the model transformation described in [20] and introduces i) four new model transformations for the automatic translation of the EAST-ADL FunctionalDesignArchitecture, HardwareDesignArchitecture and Allocation to RCM models and ii) a new model transformation that captures the timing analysis results and the corresponding RCM models and propagates them to the design level. Finally, we discuss the applicability of the proposed methodology by leveraging an industrial running example.

11.3 The MoVES Methodology: why?

Among other factors, early verification of non functional requirements can positively affect the cost-efficiency of the software development for vehicular real-time embedded systems. Currently, early verification of non functional requirements is hard to achieve due to the lack of automation supporting models integration and analysis. For instance, let us consider a typical development process as described by the flowchart in Figure 11.6 (a).

As meaningful non functional analysis (such as timing) must be run on implementation models, the engineer is required to create a RCM model manually. The non functional analysis of interest is run on the manually created RCM model and the result is verified against the given set of non functional requirements. If the specified requirements are not met, the engineer is required

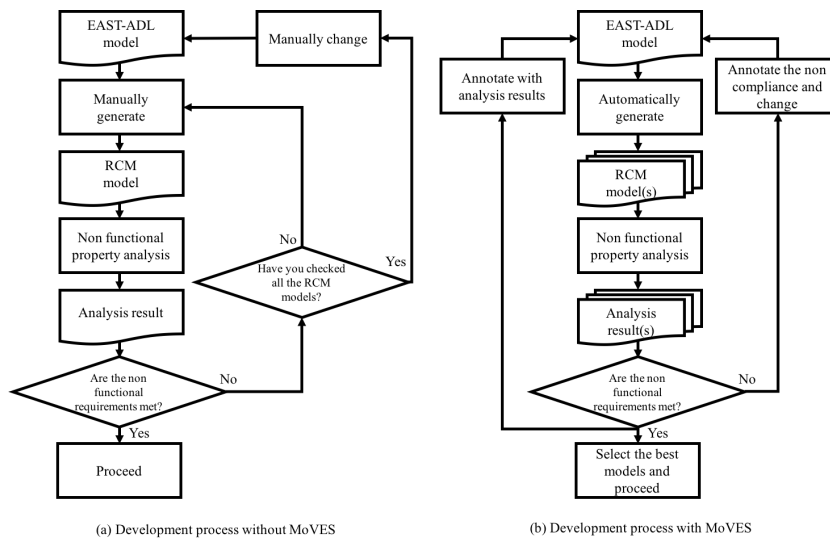


Figure 11.6: Comparison between a development process without (a) and with MoVES (b)

to iterate the process and modify or create a new RCM model until a compliant one is found. Since the process of creating and verifying implementation models is expensive, it is not leveraged early in the development process for having quick and early feedback on the design level models. To boost early verification, in this paper we propose MoVES, a novel model-driven methodology for the development of vehicular real-time embedded systems supporting early verification of non functional properties.

Let us consider a development process equipped with the proposed methodology as described by the flowchart in Figure 11.6 (b). In this setting, all meaningful RCM models are automatically generated from the design models and analysed by means of model transformations. Given a set of non functional requirements, model transformations are responsible for the selection and back propagation of the best RCM model (or set of models), too. Besides relieving the engineer from the manual and iterative definition of a RCM model, the proposed methodology enables early verification at design level. Moreover, while several iterations may be needed in the manual process to reach a RCM model that fulfils non functional requirements, MoVES is able to generate all meaningful RCM models and identify the best one(s) (as shown in Section 11.6) automatically in one single iteration.

11.4 MoVES for Timing

Timing requirements are crucial for our domain of interest, vehicular real-time embedded systems, and that timing-related issues are typical problems arising very late in the development. For this reasons, in this work we discuss an instantiation of MoVES for supporting the development and architectural exploration of system-designs with temporal awareness.

MoVES leverages the interplay of EAST-ADL and RCM and provides automation for their integration by means of six model transformations. Figure 11.7 gives a graphical representation of the methodology and its composing tasks.

The first step of the methodology is the automatic generation of the RCM models representing the software architecture and its timing properties at the implementation level. Such a generation process is characterised by a one-to-many mapping meaning that multiple RCM software models can be a valid translation of an EAST-ADL FunctionalDesignArchitecture, where each of the RCM model would entail different timing and control flow information. Within MoVES, this generation is entrusted to the FDA2RCM model transformation.

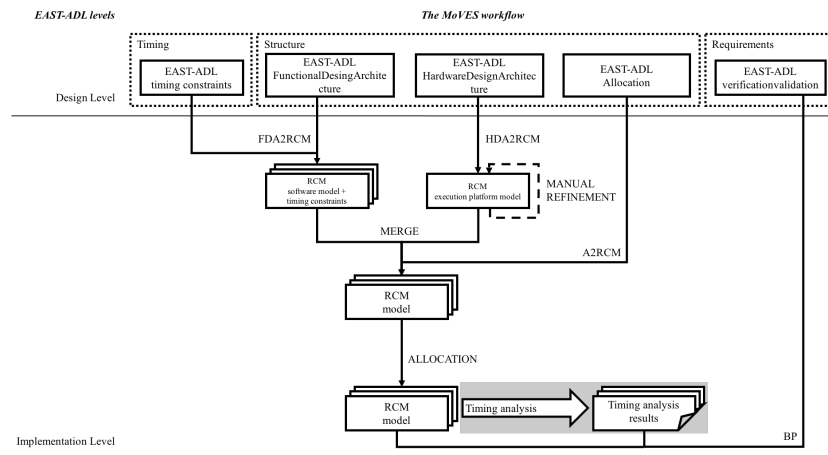


Figure 11.7: The MoVES and its composing tasks

In particular, starting from an EAST-ADL FunctionalDesignArchitecture complemented with EAST-ADL timing requirements, FDA2RCM generates, in a single execution, the set of the corresponding RCM software models equipped with RCM timing constraints as opposite to a manual generation considering only one specific solution. The second step of the methodology is the automatic generation of the RCM model representing the execution platform at the implementation level. This step is performed by the HDA2RCM model transformation which starts from an EAST-ADL HardwareDesignArchitecture and generates a corresponding RCM execution platform model. As RCM models the execution platform in a more detailed way than EAST-ADL (RCM employs the concepts of core and partition), the engineer can manually refine the generated RCM execution platform models. Detailed execution platform models are pivotal for the specification of the allocation information which, in turn, affects timing analysis. The third step of the methodology merges the RCM software and execution platform models into complete RCM models, where the allocation information can be translated from the EAST-ADL Allocation. To this end, the MERGE model transformation is responsible for merging each generated RCM software model with the generated RCM execution platform model. The result is a set of complete RCM models. The fourth step of the methodology is the generation of the allocation information on the complete RCM models and it is entrusted to the A2RCM model transformation. In particular,

A2RCM is responsible for the translation of the allocation information from an EAST-ADL Allocation model to the RCM complete models generated as a result of the MERGE transformation. Since RCM leverages a more fine-grained allocation mechanism than EAST-ADL, the methodology, by means of the ALLOCATION model transformation, is able to generate additional RCM allocation configurations that can not be derived from EAST-ADL. At this point, the RCM models contain all the information needed for the model-based timing analysis. Once the timing analysis is run⁶, the analysis results are produced and collected. The last step of the methodology is the back-propagation of the analysis results at the design level for enabling timing-aware design decisions and it is performed by the BP transformation. To this end, BP enriches the initial EAST-ADL model with the analysis results and the related RCM models such that the engineer can take timing-aware design decision on the EAST-ADL models without creating or nor editing RCM models.

In the following sections, we present a detailed discussion of each of the above mentioned model transformations. The complete implementation of the proposed methodology is available at <http://www.mrtc.mdh.se/MoVES/>.

11.4.1 FDA2RCM

FDA2RCM is a one-to-many model-to-model transformation between EAST-ADL and RCM, which is realised in the Eclipse Modeling Framework (EMF)⁷ using the Janus Transformation Language (JTL) [9]. JTL is a constraint-based bidirectional model transformation language specifically tailored to support non-determinism by generating all the possible target models in a single execution. Its implementation relies on the Answer Set Programming (ASP) [21], 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 is responsible to find and generate, in a single execution, all the possible target models that are consistent with the transformation rules following a deductive process. JTL adopts a QVTr-like syntax and allows a declarative specification of relationships between MOF models. It supports object pattern matching and automatically creates traces information to record what occurred during a transformation execution.

⁶The proposed methodology leverages model-based timing analysis. However, the analysis itself is not part of the contributions of this work. The interested reader can refer to [17] for further details.

⁷<https://eclipse.org/modeling/emf/>

The FDA2RCM transformation is the starting point of the methodology and provides for the translation of the software architecture of the vehicular embedded system under development and its timing properties. An initial version of the FDA2RCM transformation is given in [20]. The proposed version i) extends the one in [20] with new rules for the translation of the EAST-ADL FunctionalDesignArchitecture elements into the RCM software elements composing the current RCM structural hierarchy (e.g., System, Application, etc.) and ii) replaces the logic constraints in the pre- and post-conditions of the transformation rules in favour of more compact and understandable transformation rules. In a nutshell, the FDA2RCM transformation is responsible for translating the elements of the EAST-ADL FunctionalDesignArchitecture to RCM software elements. In particular, it maps the EAST-ADL DesignFunctionPrototype, FunctionPort, FunctionConnector, AgeConstraint and ReactionConstraint elements to the RCM Assembly, SWC, Port, ConnectorData, DataAge and DataReaction elements, respectively. Additionally, the FDA2RCM transformation provides for the automatic generation of the RCM ConnectorTrig, Clock and Sink elements representing detailed control flow and timing information. As detailed control flow and timing information is not described in the EAST-ADL FunctionalDesignArchitecture, a single source EAST-ADL FunctionalDesignArchitecture can not be univocally translated in a single RCM model. For instance, the DFP2C and the DFP2CCS rules in Listing 11.1 define a non deterministic portion of the FDA2RCM transformation where a DesignFunctionPrototype element can be translated either to a SWC element or to a SWC element equipped with a Clock and a Sink element.

```

1 transformation FDA2RCM(dl:designLevel, rcm:RubusMM) {
2   relation DFP2C {
3     name, id:String;
4     checkonly domain dl
5       ps:designLevel::DesignFunctionPrototype {
6         name = name,
7         id = id,
8         type = t:designLevel::DesignFunctionType {
9           isElementary = true
10        }
11      };
12   enforce domain rcm at:RubusMM::Assembly {
13     circuit = c : RubusMM::Circuit {
14       name = name,
15       id = id
16     }
17   };

```

```
17     where {...}
18   }
19   relation DFP2CCS {
20     name, id:String;
21     checkonly domain dl
22       ps:designLevel::DesignFunctionPrototype {
23         ...
24       };
25     enforce domain rcm at:RubusMM::Assembly {
26       circuit = c:RubusMM::Circuit {
27         name = name,
28         id = id,
29         interface = i:RubusMM::Interface {...}
30       },
31       clock = clk:RubusMM::Clock {...},
32       sink = snk:RubusMM::Sink {...},
33       connectorTrig = con1:RubusMM::ConnectorTrig {...},
34       connectorTrig = con2:RubusMM::ConnectorTrig {...}
35     };
36     where {...}
37   }
```

Listing 11.1: Fragment of the FDA2RCM model transformation in JTL.

In this context, the JTL engine is able to generate, in a single execution, all the RCM software models entailing different and unique configurations of, e.g., SWC, Clock and Sink elements as opposite to a manual translation considering only a specific model. It is important to notice that, logic constraints can be applied for narrowing the space of the generated models. For instance, the execution of the FDA2RCM transformation to the source EAST-ADL model depicted in Figure 11.8 (a)⁸ could be narrowed by means of logic constraints which could guide the generation of RCM models to those entailing valid configurations of SWC, Clock and Sink elements, only.

Accordingly, only the two RCM models in Figure 11.8 (b) and Figure 11.8 (c) would be generated. In the former, the SWC Actuator is activated from the SWC Sensor through the Connector Connector_Trig, while in the latter it is activated by the independent Clock Clock_Actuator.

⁸For better understandability, we represent EAST-ADL and RCM models by means of a simplified graphical concrete syntax.

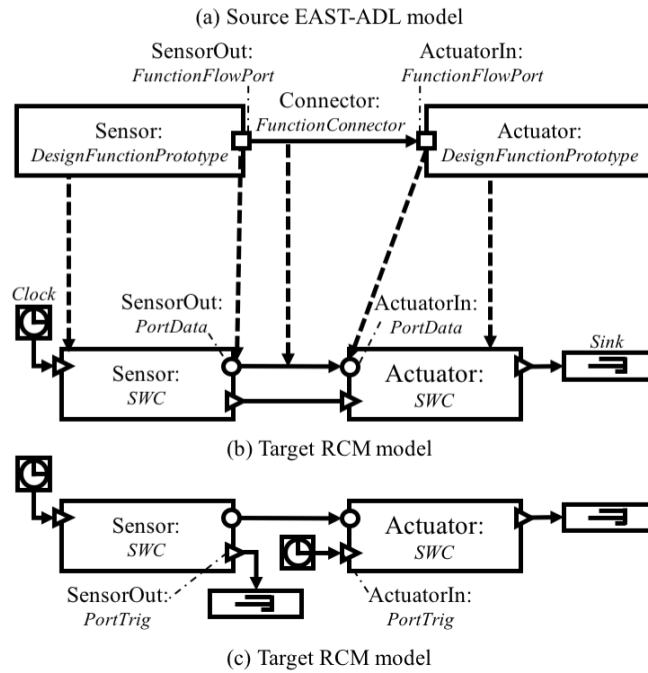


Figure 11.8: Example of simplified source and target models for the FDA2RCM transformation

11.4.2 HDA2RCM

HDA2RCM is a model-to-model transformation between EAST-ADL and RCM, which is realised by means of JTL. Together with the FDA2RCM transformation, HDA2RCM is the starting point of the methodology and provides automation means for the translation of the execution platform of the vehicular embedded system under development. In fact, execution platform models are pivotal for the specification of the allocation information which, in turns, affects timing analysis. Trivially, the over utilisation of a processor or a core can lead to timing deadline misses.

```

1 transformation HDA2RCM(dl:designLevel, rcm:RubusMM) {
2   relation AtomicHardwareComponentPrototype2Node {
3     id, name: String;
4

```

```
5      checkonly domain dl hardwareComponentPrototype :
        designLevel::HardwareComponentPrototype {
6          id = id,
7          name = name,
8          type = hardwareComponentType : designLevel::Node {...}
9      };
10
11     enforce domain rcm system : RubusMM::System {
12         connectorNetwork = connectorNetwork :
            RubusMM::ConnectorNetwork {},
13         node = node : RubusMM::Node {
14             id = id,
15             name = name
16             core = core : RubusMM::Core {
17                 ...
18                 partition = partition : RubusMM::Partition {...}
19             }
20         };
21     where {...}
22     }
23 }
```

Listing 11.2: Fragment of the HDA2RCM model transformation in JTL.

The HDA2RCM transformation is responsible for translating the EAST-ADL `HardwareComponentPrototype` elements to RCM execution platform elements. In particular, it maps the EAST-ADL `Node`, `HardwarePortConnector` and `HardwarePort` elements to the RCM `Node`, `ConnectorNetwork` and `Port` elements, respectively. As discussed in Section 11.2, RCM models the execution platform with a structural hierarchy of elements consisting of `Node`, `Core` and `Partition`. However, EAST-ADL provides modelling element for the representation of `Node` elements, only. Therefore, in order to generate valid RCM models, the HDA2RCM transformation automatically generates, for each RCM `Node` element, a `Core` and a `Partition` element, too. Please note that, the engineer can still manually refine the generated RCM execution platform model, if needed. Listing 11.2 depicts an extract of the HDA2RCM transformation consisting of the transformation rule responsible for the generation of RCM `Node`, `Core` and `Partition` elements. Figure 11.9 depicts an example of an execution of the HDA2RCM model transformation. In particular, the EAST-ADL model depicted in Figure 11.9 (a) and consisting of two connected `Node` elements, `Node1` and `Node2`, is translated into the RCM model depicted in Figure 11.9 (b) consisting of two connected `Node` elements, `Node1` and `Node2`, containing a `Core` and a `Partition` element each.

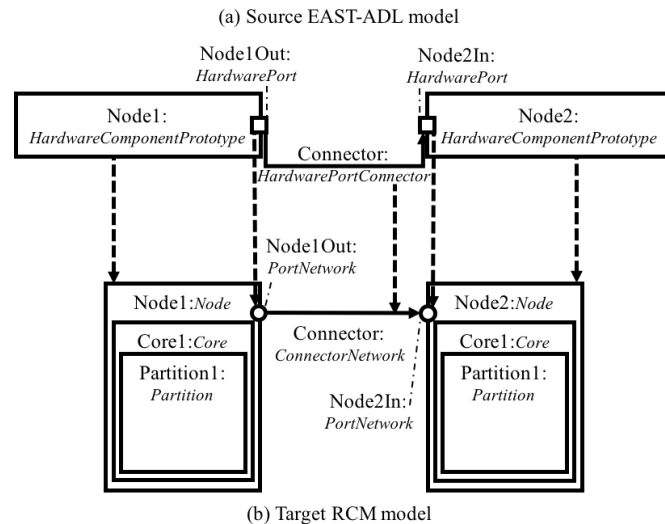


Figure 11.9: Example of simplified source and target models for the HDA2RCM transformation

11.4.3 MERGE and A2RCM

MERGE and A2RCM are two model-to-model transformations realised within EMF using the QVT Operational (QVT-O) language [22]. Query/View/Transformation (QVT) is a standard set of model transformation languages defined by the Object Management Group and it is composed by three model transformation languages, which are QVT-O, QVT Relations and QVT Core. QVT-O is an imperative language especially designed for writing unidirectional model transformations when declarative model transformations are hard to specify due to the absence of direct correspondence between elements of the source and target models. Thereby, a QVT-O model transformation explicitly specifies the steps to execute in order to generate a target model starting from a source one.

Once the FDA2RCM and the HDA2RCM transformations are run, a set of RCM software models and one RCM execution platform model are produced. In this context, the MERGE transformation is responsible for merging a RCM software model to the RCM execution platform model with the purpose of creating a complete RCM model where the allocation information can

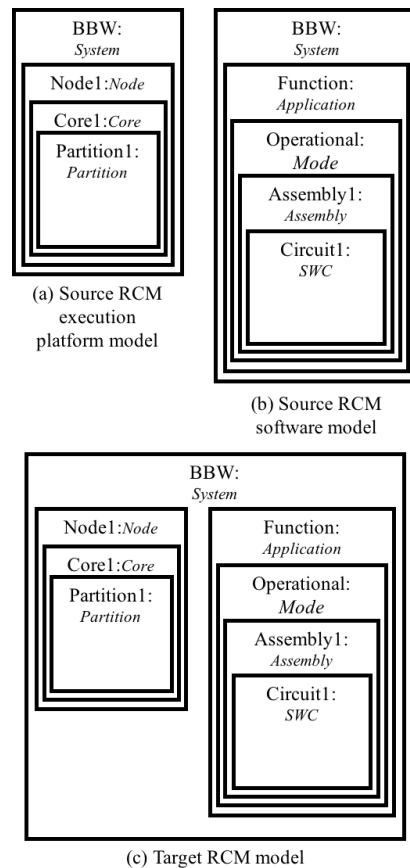


Figure 11.10: Example of simplified source and target models for the MERGE transformation

be translated and refined. In a nutshell, the MERGE transformation performs a weaving of the RCM models, where the modelling elements of the RCM execution platform model are linked to the System element of the RCM software model through its Node reference. Let us consider the RCM execution platform and software models depicted in Figure 11.10 (a) and Figure 11.10 (b), respectively. The former consists of a System element called BBW containing a Node element called Node1, which contains a Core element Core1. Eventu-

ally, Core1 contains a Partition element called Partition1. The latter consists of a System element called BBW, which in turns contains an Application element called Function. Function contains a Mode element called Operational, which contains an Assembly element called Assembly1. Assembly1 contains a SWC called Circuit1. Accordingly, the application of the MERGE transformation would produce the RCM model depicted in Figure 11.10 (c) where the RCM execution platform elements in Figure 11.10 (a) are integrated into the RCM software model in Figure 11.10 (b) by means of the Node reference of the System BBW element. A2RCM is an in-place transformation [23] which follows

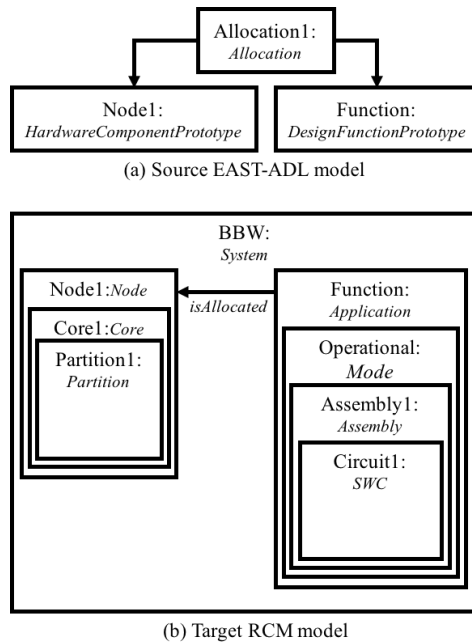


Figure 11.11: Example of a simplified source and target models for the A2C transformation

the MERGE transformation and it is responsible for translating the allocation information from the EAST-ADL Allocation to each of the RCM models generated from the MERGE transformation. Allocation information is crucial for the model-based timing analysis as, e.g., the over utilisation of a node or a core element can result in violating the timing requirements. As discussed in Sec-

tion 11.2, in EAST-ADL the allocation information is modelled by means of a set of Function Allocation elements which links the DesignFunctionPrototype element to the HardwareComponentPrototype element. For instance, the Function Allocation Allocation1 in Figure 11.11 (a) allocates the DesignFunctionPrototype Function to the HardwareComponentPrototype Node1. In RCM, the allocation information is specified by means of the isAllocated reference of the Allocatable elements. Therefore, the A2RCM transformation is responsible for setting the RCM isAllocated references according to the EAST-ADL Function Allocation elements. For instance, if we apply the A2RCM model transformation to the EAST-ADL Allocation depicted in Figure 11.11 (a), we obtain the RCM model in Figure 11.11 (b). Accordingly, the isAllocated reference of the Application element called Function is set to the Node element called Node1.

11.4.4 ALLOCATION

ALLOCATION is a one-to-many, in-place model transformation on the RCM language realised by means of JTL. As discussed in Section 11.2, compared to EAST-ADL, RCM leverages a more fine-grained structural hierarchy for the modelling of the execution platform and the allocation. In particular, within RCM, any Allocable element (Function, Mode, Assembly and SWC) can be allocated to any of the Allocator element (Node, Core and Partition). Due to the different granularity between RCM and EAST-ADL, complete allocation information can not be directly translated from an EAST-ADL Allocation. In this context, the ALLOCATION transformation provides automation means for the generation of the allocation information in the RCM models when a direct translation from EAST-ADL is not possible. In other words, the ALLOCATION transformation automatically generates the isAllocated reference of the RCM Allocatable elements and sets it to any of the RCM Allocator elements. As there can be several allocation strategies, the engineer is required to express a choice over the preferred one. This can be done by toggling the comments on the transformation rules which realise the desired allocation strategy (e.g., Assembly to Core, Assembly to Partition, SWC to Core, etc.). Based on the user choice, the ALLOCATION transformation is able to generate, in a single execution, all the RCM models which entail different and unique allocation configurations. Similarly to the FDA2RCM, logic constraints can be applied for narrowing the number of the generated RCM models. This is particularly useful when partial allocation information is already available (as in case of, e.g., legacy vehicular systems) or for discarding specific allocation configura-

tions.

```

1  transformation ALLOCATION(source:RubusMM, target:RubusMM) {
2
3      relation Assembly2AllocatedAssemblyNode{
4          name, id: String;
5
6          checkonly domain source as:RubusMM::Assembly {
7              name = name,
8              id = id
9          };
10
11         enforce domain target at:RubusMM::Assembly {
12             name = name,
13             id = id,
14             isAllocated = n:RubusMM::Node {...}
15         };
16     }
17 }

```

Listing 11.3: Fragment of the ALLOCATION model transformation in JTL.

Listing 11.3 shows a fragment of the ALLOCATION transformation which depicts the transformation rule responsible for allocating the Assembly element to Node element. Accordingly, the JTL engine generates, in a single execution, all the RCM models entailing different combinations of Assembly to Node elements. That is, the application of the ALLOCATION transformation to the RCM model depicted in Figure 11.12 (a) generates the two RCM models depicted in Figure 11.12 (b) and Figure 11.12 (c). In the former, Assembly1 is allocated to Node1, while in the latter Assembly1 is allocated to Node2.

11.4.5 BP

BP is an in-place, text-to-model transformation on the EAST-ADL metamodel realised by means of QVT-O. Within MoVES, BP is the last step for unveiling the RCM models and their related analysis results at the design level thus for enabling timing-aware design decisions. In fact, once the RCM model and the analysis results are unveiled at design level, the engineer can easily grasp the compliance of the starting EAST-ADL models to the specified timing requirements. Even further, she can select the most appropriate RCM model, among the compliant ones, for proceeding with the development process. The BP transformation uses the EAST-ADL VVCase modelling element from the

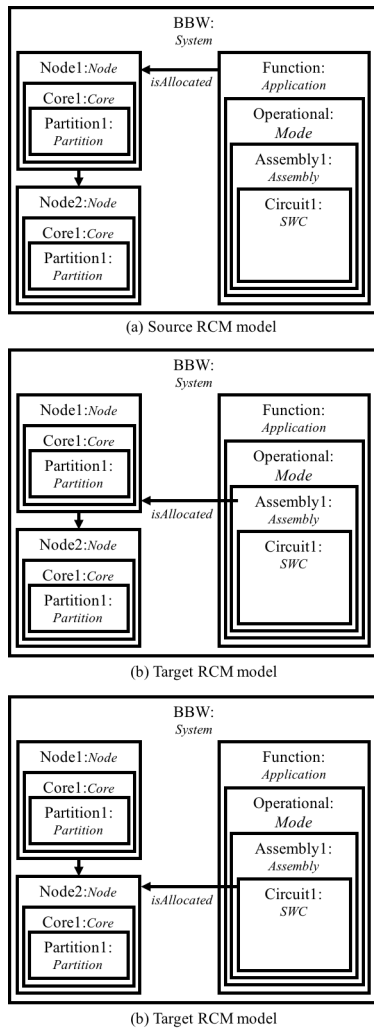


Figure 11.12: Example of simplified source and target models for the ALLOCATION transformation

Requirement package. In particular, it automatically creates the VVLog and VVActualOutcome modelling elements for the given VVCase element and sets

their attribute Date, ID and Name. Moreover, the BP transformation is responsible for setting the Text attribute of the EAST-ADL VVActualOutcome element with the URLs of the folders containing the timing analysis results and the related RCM models.

11.5 Case Study

In this section, we demonstrate the usability of MoVES by developing the adaptive cruise control system as an extension of the cruise control system. The cruise control system is a vehicular feature which allows the vehicle to keep a steady speed to the value provided by the driver. To this end, it employs (at least) 4 modules: one for communicating/controlling the engine, one for communicating/controlling the brakes, one for communicating with the driver's instrument cluster and one for the computation. However, the traditional cruise control system does not take into account traffic information such as presence of other vehicles or obstacles. The Adaptive Cruise Control (ACC) system is a vehicular feature which allows a vehicle's cruise control to adapt the vehicle's speed to the surrounding environment. More precisely, once the user sets a target speed and a time gap for the vehicle, a radar detects slow-moving vehicles or other obstacles that are in the path of the vehicle. In case an obstacle or a slower vehicle is detected, the ACC system slows down the vehicle or brakes to keep the desired distance between the ACC vehicle and the obstacle or the forward vehicle, where the distance is calculated as a function of the specified time gap and the speed of the vehicle. When the ACC system detects that the forward vehicle or the obstacle is no longer in the vehicle's path, it speeds up the vehicle to maintain the cruise speed set by the driver. With respect to the cruise control functionality, the ACC enhances the computation module with functionalities which provide the adaptive features. Figure 11.13 shows a block diagram of the ACC system⁹ that is adapted from [24].

The InstrumentClusterModule is responsible for collecting the user's inputs, such as speed and time gap, and for sending them to the AdaptiveCruiseControlModule. The EngineControlModule and the BrakeControlModule are responsible for sending the information regarding speed and braking of the vehicle to the AdaptiveCruiseControlModule, respectively. AdaptiveCruiseControlModule is the core of the ACC feature and it is responsible for calculating the acceleration and the braking of the vehicle in the presence of forward vehi-

⁹The interested reader can access the full ACC case study implementation at <http://www.mrtc.mdh.se/MoVES/>.

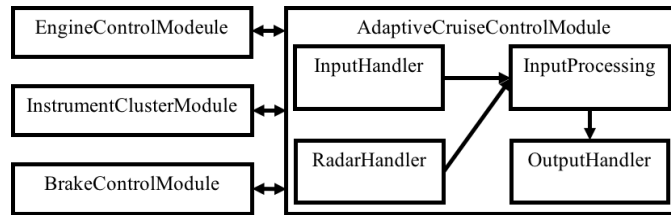


Figure 11.13: Block diagram of the ACC system.

cles or obstacles. In particular, the InputHandler software component is responsible for acquiring and processing the inputs coming from the other modules and to forward them to the InputProcessing software component. Similarly, the RadarInput software component is responsible for acquiring and processing the information coming from the radar and forwarding it to the InputProcessing software component. Based on the user inputs on the cruise mode and considering the presence of a forward obstacle, the InputProcessing software component is responsible for computing the decision whether the vehicle has to slow down, speed up or keep a steady speed. It communicates this information to the OutputHandler software component which, in turn, is responsible for sending the related brake torque and throttle signal to the BrakeControlModule and EngineControlModule, respectively.

According to MoVES, the development starts from an EAST-ADL FunctionalDesignArchitecture, HardwareDesignArchitecture and Allocation models. Figure 11.14 depicts an extract of the EAST-ADL FunctionalDesignArchitecture for the ACC. In the remainder of this section, for the sake of verbosity, we adopt a simplified concrete syntax both for the EAST-ADL and RCM models and omit some modelling elements if of no interest for the discussion.

The modules and their inner architecture are represented by means of the DesignFunctionPrototype and FunctionFlowPort elements. For instance, the RadarInput software component is represented by means of the RadarInput DesignFunctionPrototype element and the RadarIn and RadarOut FunctionFlowPort elements. The connections among software components and modules are represented by means of FunctionConnectorelements connecting the FunctionFlowPort elements. For instance, the RadarInput and InputProcessing DesignFunctionPrototype elements are connected by means of the RadarOut2RadarIn FunctionConnector connecting the RadarOut and RadarSignal FunctionFlowPorts. In addition to the architectural elements, two timing constraints, denoted by Reaction Constraint T1 and Age Constraint T2, and VVCase element, de-

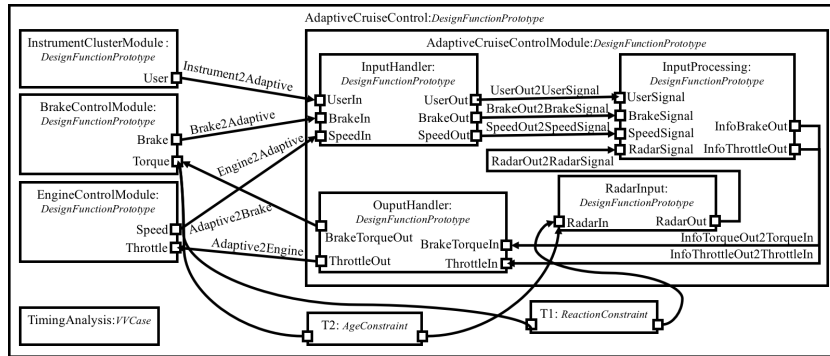


Figure 11.14: EAST-ADL FunctionalDesignArchitecture of the ACC system

noted by TimingAnalysis, are specified. According to these constraints:

The reaction and age delays between the arrival of the radar signal at the input of the InputProcessing software component and its delivery to the BrakeControlModule shall not exceed 25 and 15 milliseconds, respectively. Starting from the EAST-ADL FunctionalDesignArchitecture in Figure 11.14, MoVES automatically generates RCM software models. In particular, RCM System, Application and Mode elements, namely AdaptiveCruiseControl, AdaptiveCruiseControl and AdaptiveCruiseControl_Operational are generated. The AdaptiveCruiseControlModule, EngineControlModule, BrakeControlModule and InstrumentClusterModule DesignFunctionPrototype elements are translated into RCM Assembly elements while the InputHandler, RadarInput, InputModeControl, Processing and OuputHandler DesignFunctionPrototype elements are translated into RCM SWCs. PortData, ConnectorData and TimingConstraint elements are generated from the EAST-ADL FunctionFlowPort, FunctionConnector and TimingConstraint elements, respectively. Due to the lack of control flow information in EAST-ADL, RCM PortTrig, ConnectorTrig, Clock and Sink elements are automatically generated by the FDA2RCM.

However, as there might be multiple ways of specifying these elements, this generation produces four RCM software models each of which entails different and unique combinations of RCM PortTrig, ConnectorTrig, Clock and Sink elements. For instance, let us consider the activation of the OutputHandler SWC in the two RCM models in Figure 11.15. In the RCM model in Figure 11.15 (a), the OutputHandler SWC is triggered by an independent clock whether in the RCM model in Figure 11.15 (b) it is triggered by its predecessor, InputPro-

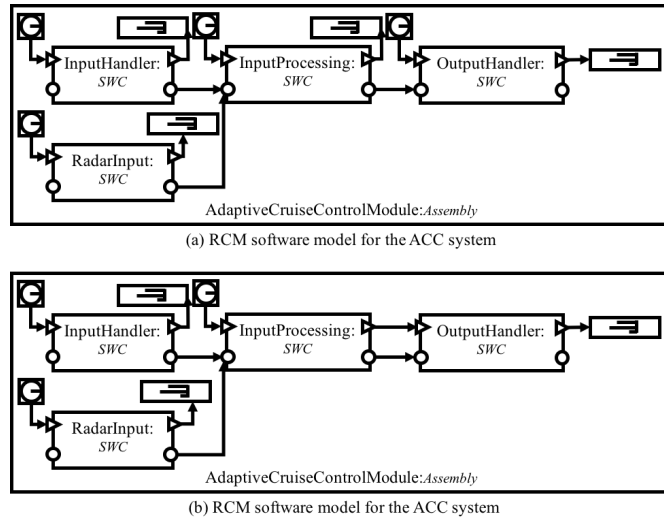


Figure 11.15: Two of the four RCM software models of the ACC system

cessing. Within MoVES, the second step is the translation of the EAST-ADL HardwareDesignArchitecture model representing the execution platform architecture. Figure 11.16 depicts the EAST-ADL HardwareDesignArchitecture for the ACC feature.

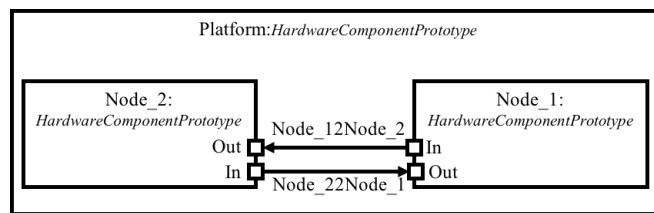


Figure 11.16: EAST-ADL HardwareDesignArchitecture of the ACC system

It is realised by means of two Node elements, Node_1 and Node_2, each of which represents a MPC560XP microcontroller that is a single-core microcontroller for vehicular and industrial safety applications¹⁰. Two HardwarePortConnector elements, Node_12Node_2 and Node_22Node_1, realise the commu-

¹⁰<http://www.nxp.com/products/automotive-products/>

nication between the two nodes. Starting from the EAST-ADL HardwareDesignArchitecture in Figure 11.16, MoVES generates the RCM model depicted in Figure 11.17.

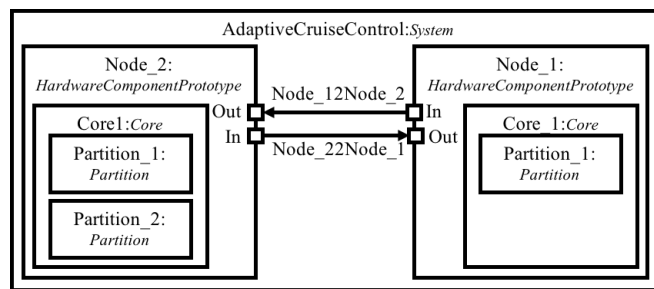


Figure 11.17: RCM execution platform model of the ACC system

In particular, the EAST-ADL Node and HardwarePortConnector elements are translated into RCM Node and NetworkConnector elements. According to the HDA2RCM transformation, the inner architecture of each generated RCM model is enriched with a Core and a Partition element. That is, Core_1 and Partition_1 elements are generated for the RCM Node_1 and Node_2 elements. Eventually, by means of manual refinements, an additional Partition element, Partition_2 is added to the RCM Node_2. At this point, MoVES merges the RCM software and execution platform models and the result is a set of four complete RCM models where the allocation information from the EAST-ADL Allocation can be translated. To this end, Figure 11.18 depicts the EAST-ADL Allocation model for the ACC feature.

The EAST-ADL Allocation model consists of four FunctionAllocation elements mapping AdaptiveCruiseControlModule to Node_1 and InstrumentClusterModule, EngineControlModule and BrakeControlModule to Node_2. Accordingly, MoVES translates the allocation information on the RCM models resulting from the MERGE transformation. Figure 11.19 depicts one example of RCM model along with the translated allocation information. Consequently, the isAllocated references of the InstrumentClusterModule, BrakeControlModule and EngineControlModule assemblies are set to the Node_2 element while the isAllocated reference of the AdaptiveCruiseControlModule

microcontrollers-and-processors/32-bit-power-architecture/ultra-reliable-mpc56xx-32-bit-automotive-industrial-microcontrollers-mcus/ultra-reliable-mpc560xp-mcu-for-automotive-industrial-safety-applications:MPC560xP

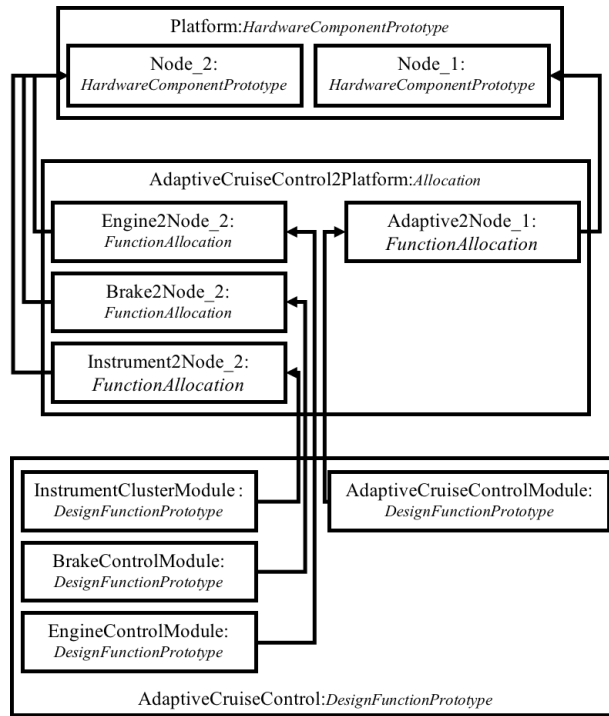


Figure 11.18: EAST-ADL Allocation of the ACC system

Assembly elements is set to the *Node_1* element. However, as EAST-ADL does not leverage the concepts of cores and partitions for the modelling of the execution platform, the allocation of the *InstrumentClusterModule*, *EngineControlModule* and *BrakeControlModule* elements can not be refined with respect to core and partition elements of *Node_2*. Nevertheless, such an information can be automatically generated from the ALLOCATION transformation as described in Section 11.3. In particular, for the ACC system, we decided to choose an allocation strategy which allocates Assembly to Partition elements. Consequently, as there are 8 different ways to allocates the three Assembly elements to the two Partition elements, MoVES generates a final set of 32 RCM models (8 refined RCM models for each of the 4 RCM models resulting from the A2C). Eight of the 32 final RCM models are depicted in Figure 11.20.

At this point, model-based timing analysis is run on each of the gener-

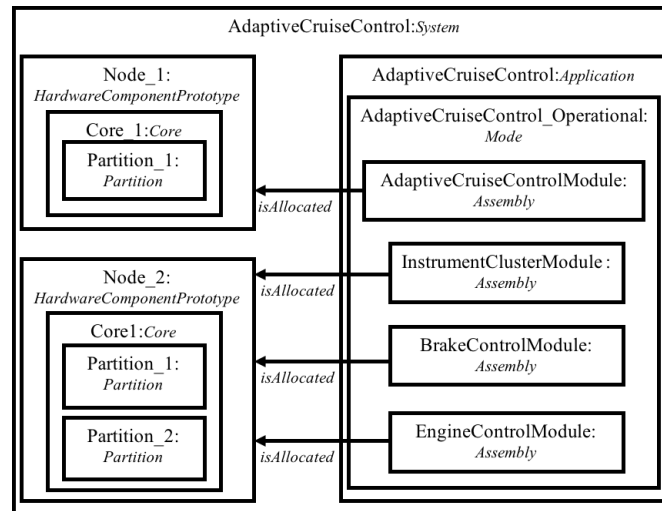


Figure 11.19: RCM model for the ACC system with the allocation information

ated 32 RCM models with the aim of verifying whether the specified reaction and age constraints are met. Table 11.1 summarises the results of the timing analysis. Table 11.1 summarises the analysis results for the 32 generated RCM models. It can be seen that, out of the 32 RCM models of the ACC system, only 16 RCM models satisfy the specified age and reaction constraints. In particular, MoVES is able to automatically identify the 5 RCM models with the best timing performances. In this respect, it is important to note that MoVES can be easily constrained for notifying the engineer only with the best RCM models rather than with all the compliant ones. Eventually, the VVLog and VVActualOutcome elements are created for the VVCase TimingAnalysis. Moreover, the attribute Text of the VVActualOutcome is set to the URLs of the folders containing the generated RCM models and their analysis results. At this point, the engineer can select any of the 16 RCM models satisfying the specified age and reaction constraints and continue with the synthesis of the code for the target platform.

| RCM Model ID | Calculated Age Delay (microseconds) | Calculated Reaction Delay (microseconds) |
|--------------|-------------------------------------|--|
| 1 | 21630 | 31170 |
| 2 | 21380 | 30920 |
| 3 | 21480 | 31020 |
| 4 | 21380 | 30920 |
| 5 | 21730 | 31270 |
| 6 | 21380 | 30920 |
| 7 | 21380 | 30920 |
| 8 | 21380 | 30920 |
| 9 | 11810 | 21810 |
| 10 | 11560 | 21560 |
| 11 | 11660 | 21660 |
| 12 | 11560 | 21560 |
| 13 | 11910 | 21910 |
| 14 | 11560 | 21560 |
| 15 | 11560 | 21560 |
| 16 | 11560 | 21560 |
| 17 | 11690 | 21690 |
| 18 | 11440 | 21440 |
| 19 | 11540 | 21540 |
| 20 | 11440 | 21440 |
| 21 | 11790 | 21790 |
| 22 | 11440 | 21440 |
| 23 | 11440 | 21440 |
| 24 | 11440 | 21440 |
| 25 | 21630 | 31170 |
| 26 | 21380 | 30920 |
| 27 | 21480 | 31020 |
| 28 | 21380 | 30920 |
| 29 | 21730 | 31270 |
| 30 | 21380 | 30920 |
| 31 | 21380 | 30920 |
| 32 | 21380 | 30920 |



Specified Age Constraint: 15000 microseconds
 Specified Reaction Constraint: 25000 microseconds
 = satisfied
 = best

Table 11.1: Age and Reaction delay analysis results

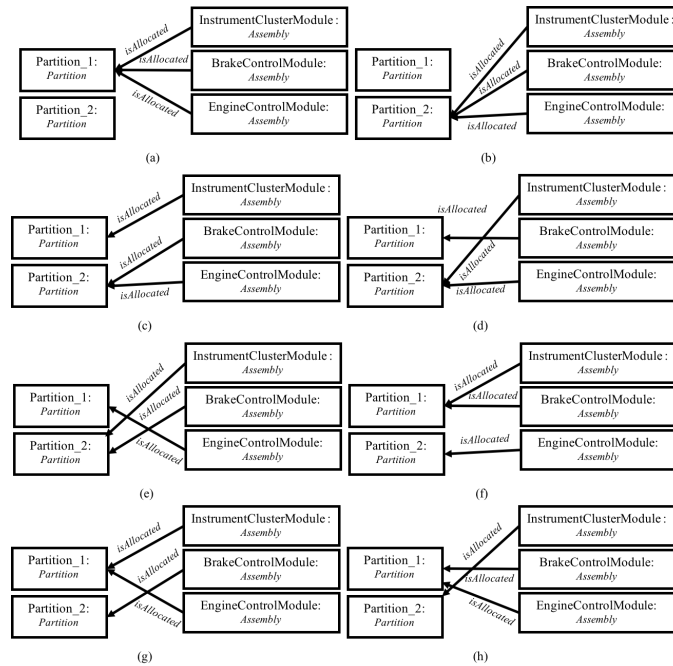


Figure 11.20: RCM models for the ACC system with the refined allocation information

11.6 Discussion and Validation

In this work, we introduced a novel model-driven methodology, MoVES, for the software development of real-time distributed vehicular embedded systems on single- and multi-core platforms. Considering the importance of timing, the proposed methodology supports the development and architectural exploration of system-designs with temporal awareness. To this end, MoVES leverages the interplay of two domain-specific modelling languages, EAST-ADL and RCM, and provides a fully automated mechanism for the generation of the RCM models containing detailed software, execution platform, timing and allocation information for schedulability analysis. EAST-ADL has been developed by the automotive industry and its consortium includes several international automotive companies such as McLaren, Volvo, FIAT, Hyundai, etc. The validation of the applicability and correctness of EAST-ADL is outside the scope of this

work, however, the interested reader, can refer to [3]. The RCM metamodel definition exploited in this work is an extension of the previous definition given in [12]. Through the years the applicability and correctness of RCM has been verified against several industrial system designs such as i) the Intelligent Parking Assist (IPA) System [25] (consisting of 2 Node elements and 42 SWCs), ii) the simplified IPA system [26] (consisting of 2 Node elements and 7 SWCs), iii) the Steer-by-wire System [20] (consisting of 1 Node and 6 SWCs) and the Brake-by-wire System [12] (consisting of 3 Node elements and 14 SWCs). Additionally, its industrial relevance has been acknowledged by our industrial partners (e.g., Volvo CE, BAE Systems, etc.) through several national and international projects [27].

The automation mechanism, core of the MoVES methodology, is realised by means of a suite of six model transformations. In this respect, the case study presented in Section 11.5 helps in discussing some interesting properties of the model transformations, such as syntactic and semantic correctness, complexity, termination and performance. With the term syntactic correctness, we refer to the ability of a transformation to produce valid target models when executed on valid source models [24]. Such a property holds for transformations leveraged in MoVES and the interested reader can easily check the validity of the generated RCM models by accessing the MoVES implementation¹¹ With the term semantic correctness, we refer to the ability of a transformation to produce semantically valid target models [24]. In this respect, it is important to note that none of the transformations within MoVES suffer of information loss. That is, they consist of a precise and finite set of rules for mapping EAST-ADL to RCM elements without altering, violating or colliding the structural hierarchies of the languages. Eventually, the semantic of the generated RCM models was validated by the leveraged schedulability analysis. We consider two dimensions for the transformations' complexity. The first dimension of the transformations' complexity refers to the complexity of the generated RCM models. In this respect, it is important to note that the transformations always generate RCM models of equal complexity of manually defined RCM models. The second dimension of the transformations' complexity refers to the size of the generated set of RCM models. All the leveraged model transformations, except FDA2RCM and ALLOCATION, are one-to-one. For the FDA2RCM transformation, if we set n as the number of the software functions enclosed by the timing constraints, then FDA2RCM would generate a maximum of $2^{(n-1)}$ RCM models. For instance, in the case of the ACC fea-

¹¹ <http://www.mrtc.mdh.se/MoVES/>

ture, despite the EAST-ADL FunctionalDesignArchitecture models 9 software functions, the FDA2RCM generates only 4 RCM models as the specified age and reaction constraints enclose 3 software functions. It is important to remark that, as discussed in Section 11.5, logic constraints can be applied for reducing the number of the generated RCM models by discarding those models which entails configurations of timing and control elements that are known to be not relevant. For the ALLOCATION transformation, if we set k as the number of RCM Allocator elements and n as the number of the RCM Allocated elements, then ALLOCATION would generate a maximum of $k^{(n)}$ RCM models. For instance, in the case of the ACC feature, we decided to proceed with an allocation strategy assigning the $n = 3$ RCM Assembly to the $k = 2$ Partition elements. Accordingly, a total number of 8 RCM models containing unique allocation configurations was generated. Similar to FDA2RCM, logic constraints can be specified for narrowing the generation process. For instance, let us consider the case in which we are not interested in the RCM models containing allocation configuration where the EngineControlModule Assembly is not allocated to the Partition.2. In other words, we fix the allocation of the EngineControlModule Assembly to the Partition.2. In this case, ALLOCATION would generate only 4 RCM models. Trivially, the transformations' complexity can affect the termination and performance properties. Although providing formal proof on these properties was outside the scope of the work, the case study showed that all the transformation terminate in few seconds¹².

We believe that MoVES discloses the opportunity to improve the cost-efficiency of software development process by means of i) automation and ii) reduced need for late modifications on the software. In particular, automation by means of model transformations allows to cut the development time while ensuring the compliance with the non functional requirements of the vehicular embedded software. Without MoVES, in fact, the software development would progress incrementally with team of engineers manually defining implementation models until a suitable one, from a non functional perspective, is found. On the contrary, by adopting MoVES, the implementation models would be automatically generated and non functional requirements verified at once allowing the engineers to focus and reason only on the compliant ones. Kurt-Lennart Lundbäck, CEO of Arcticus Systems, on the use of MoVES:

“I feel that autonomous vehicles on multi-core platforms are introducing a lot more of complexity and concerns. In this domain, automation can be a game-changer. For us, as a tool and technology providers, it would be partic-

¹²The case study was run on a 1,7 GHz Intel Core i7 processor, with 8 GB 1600 MHz DDR3 memory.

ularly beneficial to have automated support for things like ‘allocation’ for reducing the complexity of our tool suite, Rubus ICE, and improve its usability. For our customers, this can result in lower development effort and improved confidence in the quality of the software under development.”

It is important to note that MoVES does not introduce accidental complexity in the software development process. In fact, it is true that setting up the methodology might require an additional effort, but it is a one-time-effort as opposed to manual processes always requiring constant effort. Moreover, the engineer would have to deal only with the set of RCM models satisfying the non functional requirements which, as shown in Section 11.5 for timing, is a limited number (5 out of the 32 generated RCM models). By allowing early verification, MoVES discloses the opportunity of reducing late modifications on the vehicular embedded software, which empirical studies showed to be up to 40 times more expensive than same modifications during the design of the software [6]. In fact, by using MoVES, the engineer is either notified on the non compliance of the starting EAST-ADL models to the set of the considered non functional requirements or notified with the set of the compliant RCM models with which proceed for the development. In the former scenario, late modifications are prevented while in the latter they are not needed.

In this article, MoVES is presented by specifically targeting timing properties, given the paramount relevance of these properties in the design and development of vehicular embedded systems. Nonetheless, there are further non functional properties that play an important role during the development of these systems, namely memory usage, energy efficiency, and so forth. In this respect, it is worth to note that the methodology proposed by MoVES can be instantiated to consider these and other properties, as long as they are measurable and comparable at the EAST-ADL and RCM levels of detail. Additionally, other properties can be exploited for comparing multiple RCM models having equally good timing performance and selecting the best available RCM model solution. Moreover, further non functional properties could be considered from the initial stages of the proposed workflow to be integrated and used during the generation process of the possible solutions. In both cases, the MoVES would need to be extended only in terms of specific model transformations for the generation of the related non functional properties of interest.

11.7 Related Work

This article deals with several research problems, here grouped as development of vehicular software systems, development of multi-core systems, and support for design-space exploration. In the remainder of this section, for each of the mentioned problems relevant related works are discussed.

11.7.1 Development of vehicular embedded systems

The growing complexity of nowadays vehicular software demands adequate approaches for its effective development. AUTOSAR [4] is an industrial initiative to provide a standardised software architecture for the development of vehicular software systems. The timing model for AUTOSAR was developed in the TIMMO and TIMMO2USE projects [28, 29]. In these projects, a framework was developed to specify the end-to-end timing constraints and analyse the corresponding end-to-end delays [30, 31, 15]. In general, AUTOSAR is not meant to be used in isolation, but plays the role of the implementation level as e.g. prescribed by the EAST-ADL. Even if the layered structuring of EAST-ADL entails abstraction and separation-of-concerns in the development, there is no specific automation support for interconnecting the different layers. As a consequence, the results of analysis performed at lower levels of abstraction, e.g. by means of AUTOSAR, have to be manually tracked back to higher levels, e.g. design models. More in general, the discontinuities in the development process due to the abstraction gaps between the different layers have to be tackled manually by the developer. This task can be time-consuming and error-prone, especially when considering the complexity of modern vehicular systems [32]. On the contrary, in this article we propose to leverage automation through model transformations to keep the consistency between the different abstraction levels. The abstraction gaps naturally introduce non-determinism, which is managed by an appropriate transformation language (JTL). Multi-core architectures are part of the AUTOSAR standard since version 4.0. However, AUTOSAR does not distinguish between the control and the data flows at the application software level, a distinction that is fundamental for providing early timing verification [33]. Moreover, AUTOSAR does not provide means for modelling the execution platform [34]. These issues motivate our choice of using RCM as the implementation level for EAST-ADL and the MoVES methodology. CHESS is a cross-domain framework for the design of component-based embedded systems, including vehicular systems [35]. It is based on a combination of different languages, like MARTE and SYSML,

which gave birth to a specific UML profile. The framework provides modelling of embedded software for early analysis, such as dependability and schedulability, as well as for code generation, monitoring, and back-propagation. Currently, the CHES framework does not provide design-space exploration, nor it supports uncertainty in the development process. Vehicular systems are often referred to as cyber-physical systems (CPS) [36], especially when considering autonomous driving and networks of vehicles (fleets). Several approaches deal with CPS development by adopting multi-paradigm modelling techniques and leverage simulation mechanisms to perform early analysis of systems [37, 38]. Even if the analyses presented in this article do not exploit simulation techniques, the MoVES methodology does not prevent the use of simulation mechanisms to analyse and select the generated design alternatives with respect to quality attributes of interest.

11.7.2 Development of embedded systems

Given the ubiquity of software and its mission criticality, there exists a corpus of literature devoted to the design of embedded systems pertaining to disparate application domains and posing a special focus to QoS requirements. In this respect, several works are based on the use of UML and the UML profile for MARTE [39]. These general-purpose languages might be used as alternatives to domain-specific (i.e. vehicular) languages as, e.g., AUTOSAR and RCM. GASPARD is a framework based on MARTE for the design of parallel embedded systems [40]. It provides a modelling support based on UML and MARTE, and prescribes a workflow made-up of subsequent analyses and refinement steps, from higher to lower abstraction levels. Similarly to MoVES, some analyses and refinements can be performed at the (EAST-ADL) design level, while others can only be performed at lower abstraction levels (e.g., timing). Also for GASPARD moving from higher to lower abstraction levels raises the issue of managing multiple alternatives. Indeed, in [40] the authors advocate for a refinement process able to prune inadequate alternatives based on analysis results. However, the authors do not discuss the management of multiple alternatives, nor provide details about the refinement process that seems to rely on the selection of a single candidate for each level of abstraction. VERTAF/Multi-core [41] is a UML-based framework for the development of multi-core software. The software system is described by means of UML class diagrams, timed state machines and sequence diagrams, while model transformations are used for enabling analysis like schedulability. MARTE is adopted also in [42] and [43] to design the high-level architecture of the software system and for

the generation of implementation code. The former approach prescribes the use of UML for modelling the software components while MARTE is used for modelling hardware and software to hardware allocations. Moreover, timing verification is accomplished by running simulations on the automatically generated code. The latter approach instead targets component-based system deployment. Components allocations are derived by means of code generation, which is based on high level description models conforming to MARTE. In [44], the authors propose a technique to specify tasks and their allocation to cores. The technique is based MARTE and allows to perform simulation and task allocation optimisation based on the execution. AADL [45] is an architecture description language initially tailored to the avionic domain but currently used for modelling embedded systems in general. AADL supports the design of multi-core embedded software by means of separation of concerns between software and hardware elements. The software architecture is described in terms of, e.g., Processes and Threads, that is at a lower level of abstraction if compared to RCM.

11.7.3 Support for design-space exploration

Design-space exploration (DSE) typically involves the generation, analysis, and optimisation of multiple design alternatives [46]. The step-by-step expansion of design alternatives illustrated in this article can be classified as rule-based DSE complying to the model generation pattern. In fact, the space of solutions is represented by means of models and the corresponding alternatives are generated through model transformations [47, 48]. Moreover, an exhaustive derivation of models at the implementation level is performed by enriching design level (EAST-ADL) models with timing and allocation details, constrained by the system architecture and domain-specific rules [49]. Such a derivation process is quality-driven [47, 50], in the sense that JTL model transformations generate all the viable (timing/allocation) solutions for a certain system architecture, while do not aim at automatically discovering optimisation opportunities at design level [51]. Kang et al. introduce in [52] a DSE tool called FORMULA. FORMULA permits the user to define equivalence classes over alternatives, such that only non-isomorphic solutions are generated in the exploration phase. Their aim is to enhance the cost-effectiveness of DSE by avoiding the exploration of design alternatives not relevant to a certain development stage. Indeed, some alternatives might look equivalent to the user whom, due to the maturity of the design, might not yet be concerned with some of the details about the system that are changed. Several DSE mecha-

nisms cope with the search of input model configurations such that to achieve an optimised system in terms of certain properties of interest. Since the search space is typically huge, research efforts are devoted to generating candidates in an effective way (e.g., close to the optimal solution). Optionally, user's inputs and/or heuristics are exploited to drive the exploration and prune alternatives. DESERT [53] is a tool that provides DSE based on exploration and pruning rules manually defined by the user. Interestingly, also DESERT is based on a compact representation of viable design alternatives, in particular by means of ordered binary decision diagrams. Differently to the approach proposed by MoVES, DESERT forces the user to perform an element-by-element selection of the available options to reduce the space of solutions to a single one. Shaetz et al. [47] proposed a rule-based DSE mechanism tailored to embedded systems development. The exploration is realised by means of model transformations specified as Prolog programs. In particular, transformation rules both define the generation of alternatives and constrain the space of solutions. However, there is no explicit visualisation technique to reveal available alternatives to the user. Therefore, the user has to foresee possible design alternatives and write appropriate exploration/pruning rules. A number of additional techniques target multi-criteria optimisations. In this respect, one precondition to be met by the DSE mechanism is a generic representation of the solution space, which has to be compatible with multiple exploration approaches. In general these techniques leverage intermediate formats over which several explorations/optimisations can be run. Notably, in [54] Saxena and Karsai introduce an extension of a domain-specific language devoted to DSE. Such extension is exploited by disparate constraint solvers to compute multiple explorations/optimisations. Similarly, Octopus [55] supports DSE for software intensive embedded systems. A DSE tailored intermediate representation is exploited to implement an iterative refinement process based on analyses, searches, and diagnostics over the space of available solutions. In MoVES, the uncertainty points can be combined to address multiple DSE needs. Moreover, the order of resolutions (i.e., solution exploration) can be exploited to set priorities over properties.

During the last decades, several approaches for the effective software development of vehicular embedded systems were introduced. In general, this problem requires the consideration of a number of models which can rapidly become unbearable to handle manually. In this work, we tackled this problem by employing i) the interplay of two modelling languages, ii) a fully automated mechanism and iii) model-based timing analysis. While the interplay of the two modelling languages allows for the explicit modelling of the vehicular's software architecture and its execution platform, the automated mechanism

provides for the automatic generation of all the model alternatives meaningful from a timing perspective.

11.8 Conclusion and Future Work

The work presented in this paper describes a timing-aware model-driven methodology for the software development of distributed vehicular embedded systems on single- and multi-core. In particular, it tackles the problem of guiding the engineer in taking timing-aware design decisions at design level when modifications on the vehicular embedded system are generally more cheaper than modifications at the implementation level. Generally, this requires the consideration of a number of implementation alternatives which are hard to handle without an automated support. We proposed to solve this by introducing automation in terms of six model-transformations which describe precise relationships between EAST-ADL (the language used at design level) and RCM (the language used at implementation level). We exploited the properties of a constraint-based transformation language, JTL, to automatically derive all the possible RCM models entailing meaningful and unique timing and allocation configurations. Eventually, we leveraged model-based timing analysis for the timing verification of the generated RCM models. The case study we conducted together with our industrial partners in the automotive domain demonstrated i) promising results in terms of reduction of late modifications as well as the i) applicability of the methodology.

Despite the generation of the RCM models entailing different timing and allocation configurations is transparent to the engineer and it can be guided with logic constraints, issues about scalability and performance may remain open. In this respect, the main future investigation direction encompasses the study of a smarter generation process for narrowing and clustering the space of the generated RCM models and the use of further non functional properties for pruning the set of the generated RCM models. In addition, we are planning to equip the proposed methodology with the compact notation discussed in [7] for enabling the visualisation of multiple RCM models as a single RCM model equipped with uncertainty points. Another line of future investigation encompasses the extension of the proposed methodology to the higher EAST-ADL structural abstraction levels.

Acknowledgments

The work in this paper is supported by the Swedish Knowledge Foundation (KKS) through the PreView and MOMENTUM projects, and by the Swedish Research Council (VR) through the SynthSoft project. We thank our industrial partners Arcticus Systems, Volvo Construction Equipment and BAE Systems Hägglunds, Sweden.

Bibliography

- [1] Robert N Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.
- [2] D C Schmidt. Guest Editor’s Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 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] AUTOSAR Technical Overview, Version 4.3, The AUTOSAR Consortium, Dec., 2016. <http://autosar.org>.
- [5] 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.
- [6] Daniel Galin. *Software quality assurance: from theory to implementation*. Pearson Education India, 2004.
- [7] Romina Eramo, Alfonso Pierantonio, and Gianni Rosa. Managing uncertainty in bidirectional model transformations. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 49–58. ACM, 2015.
- [8] Tao Zan, Hugo Pacheco, and Zhenjiang Hu. Writing bidirectional model transformations as intentional updates. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 488–491. ACM, 2014.

- [9] 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.
- [10] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *Software, IEEE*, 20(5):42–45, 2003.
- [11] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: the state of the practice. *Software, IEEE*, 20(6):61–69, 2003.
- [12] Alessio Bucaioni, Saad Mubeen, Federico Ciccozzi, Antonio Cicchetti, and Mikael Sjödin. Technology-preserving transition from single-core to multi-core in modelling vehicular systems. In Springer, editor, *13th European Conference on Modelling Foundations and Applications*, July 2017.
- [13] ISO 26262-1:2011: Road vehicles in Functional safety. <http://www.iso.org/>.
- [14] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2):117–134, 1994.
- [15] 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.
- [16] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Communications-Oriented Development of Component-Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture*, 60(2):207–220, 2014.
- [17] 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.

- [18] Alessio Bucaioni, Antonio Cicchetti, and Mikael Sjödin. Towards a meta-model for the rubus component model. In *1st International Workshop on Model-Driven Engineering for Component-Based Software Systems*, September 2014.
- [19] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Saad Mubeen, and Mikael Sjödin. A metamodel for the rubus component model: Extensions for timing and model transformation from east-adl. *Journal of IEEE Access*, 5(1), December 2016.
- [20] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, and Mikael Sjödin. Anticipating implementation-level timing analysis for driving design-level decisions in east-adl. In *International Workshop on Modelling in Automotive Software Engineering*, September 2015.
- [21] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. volume 88, pages 1070–1080. MIT Press, 1988.
- [22] Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT). OMG Group.
- [23] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.
- [24] Pär Berggren. Autonomous cruise control for chalmers vehicle simulator. 2008.
- [25] 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*. IEEE, April 2014.
- [26] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Saad Mubeen, Mikael Sjödin, and Alfonso Pierantonio. Handling uncertainty in automatically generated implementation models in the automotive domain. In *42nd Euromicro Conference series on Software Engineering and Advanced Applications*, September 2016.
- [27] Arcticus System Research. <http://www.arcticus-systems.com/research/>.
- [28] TIMMO-2-USE. <https://itea3.org/project/timmo-2-use.html>.

- [29] Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, 2012. Available at: <http://www.timmo-2-use.org/pdf/T2UBrochure.pdf>. 2012.
- [30] 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.
- [31] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.
- [32] Bran Selic and Leo Motus. Using models in real-time software design. *Control Systems, IEEE*, 23(3):31–42, June 2003.
- [33] Saad Mubeen, Thomas Nolte, Mikael Sjödin, John Lundbäck, and Kurt-Lennart Lundbäck. Supporting Timing Analysis of Vehicular Embedded Systems through the Refinement of Timing Constraints. *Accepted for publication in the Journal of Software and Systems Modeling*, DOI: 10.1007/s10270-017-0579-8, 2017.
- [34] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded system design for automotive applications. *Computer*, 40(10):42–51, October 2007.
- [35] Antonio Cicchetti, Federico Ciccozzi, Silvia Mazzini, Stefano Puri, Marco Panunzio, Tullio Vardanega, and Alessandro Zovi. Chess: a model-driven engineering tool environment for aiding the development of complex industrial systems. In *27th International Conference on Automated Software Engineering (ASE 2012)*, September 2012.
- [36] Patricia Derler, Edward A. Lee, and Alberto Sangiovanni-Vincentelli. Modeling cyber-physical systems. *Proceedings of the IEEE (special issue on CPS)*, 100(1):13 – 28, January 2012.
- [37] Pieter J. Mosterman and Hans Vangheluwe. Computer automated multi-paradigm modeling: An introduction. *SIMULATION*, 80(9):433–450, 2004.
- [38] J. C. Jensen, D. H. Chang, and E. A. Lee. A model-based design methodology for cyber-physical systems. In *2011 7th International Wireless Communications and Mobile Computing Conference*, pages 1666–1671, July 2011.

-
- [39] The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, 2010. OMG Group, January 2010.
- [40] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Rabie Ben Atitallah, Anne Etien, Philippe Marquet, and Jean-Luc Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(4):39, 2011.
- [41] Pao-Ann Hsiung, Shang-Wei Lin, Yean-Ru Chen, Nien-Lin Hsueh, Chih-Hung Chang, Chih-Hsiung Shih, Chorng-Shiuh Koong, Chao-Sheng Lin, Chun-Hsien Lu, Sheng-Ya Tong, Wan-Ting Su, and William C. Chu. Model-driven development of multi-core embedded software. In *Proceedings of the 2009 ICSE Workshop on Multicore Software Engineering, IWMSE '09*, pages 9–16, Washington, DC, USA, 2009. IEEE Computer Society.
- [42] Federico Ciccozzi, Tiberiu Seceleanu, Diarmuid Corcoran, and Detlef Scholle. Uml-based development of embedded real-time software on multi-core in practice: Lessons learned and future perspectives. *Journal of IEEE Access*, 2(1):1–12, September 2016.
- [43] A. Nicolas, H. Posadas, P. Peñil, and E. Villar. Automatic deployment of component-based embedded systems from uml/marte models using mcapi. In *Design of Circuits and Integrated Systems*, pages 1–6, Nov 2014.
- [44] Federico Ciccozzi, Juraj Feljan, Jan Carlson, and Ivica Crnkovic. Architecture optimization: Speed or accuracy? both! *Software Quality Journal*, 22(1):1–28, October 2016.
- [45] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (aadl): An introduction. Technical report, DTIC Document, 2006.
- [46] Matthias Gries. Methods for evaluating and covering the design space during early design development. *Integr. VLSI J.*, 38(2):131–183, December 2004.
- [47] B. Schätz, F. Hözl, and T. Lundkvist. Design-space exploration through constraint-based model-transformation. In *Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on*, pages 173–182, March 2010.

- [48] Hani Abdeen, Dániel Varró, Houari Sahraoui, András Szabolcs Nagy, Csaba Debreceni, Ábel Hegedüs, and Ákos Horváth. Multi-objective optimization in rule-based design space exploration. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14*, pages 289–300, New York, NY, USA, 2014. ACM.
- [49] 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.
- [50] Mauro Luigi Drago, Carlo Ghezzi, and Raffaella Mirandola. Towards quality driven exploration of model transformation spaces. In *Procs. of the 14th Int. Conf. on Model Driven Engineering Languages and Systems, MODELS'11*, pages 2–16, Berlin, Heidelberg, 2011. Springer-Verlag.
- [51] Martin Walker, Mark-Oliver Reiser, Sara Tucci-Piergiovanni, Yiannis Papadopoulos, Henrik Lnn, Chokri Mraidha, David Parker, DeJiu Chen, and David Servat. Automatic optimisation of system architectures using east-adl. *Journal of Systems and Software*, 86(10):2467–2487, 2013.
- [52] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems: 16th Monterey Workshop 2010, Redmond, WA, USA, March 31-April 2, 2010, Revised Selected Papers*, chapter An Approach for Effective Design Space Exploration, pages 33–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [53] Sandeep Neema, Janos Sztipanovits, Gabor Karsai, and Ken Butts. *Embedded Software: Third International Conference, EMSOFT 2003, Philadelphia, PA, USA, October 13-15, 2003. Proceedings*, chapter Constraint-Based Design-Space Exploration and Model Synthesis, pages 290–305. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
- [54] Tripti Saxena and Gabor Karsai. *Model Driven Engineering Languages and Systems: 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*, chapter MDE-Based Approach for Generalizing Design Space Exploration, pages 46–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [55] Twan Basten, Martijn Hendriks, Nikola Trčka, Lou Somers, Marc Geilen, Yang Yang, Georgeta Igna, Sebastian Smet, Marc Voorhoeve, Wil Aalst,

Henk Corporaal, and Frits Vaandrager. *Model-Based Design of Adaptive Embedded Systems*, chapter Model-Driven Design-Space Exploration for Software-Intensive Embedded Systems, pages 189–244. Springer New York, New York, NY, 2013.

