Mälardalen University Doctoral Thesis
No.171

Preservation of Extra-Functional Properties in Embedded Systems Development

Mehrdad Saadatmand

February 2015

MÄLARDALEN UNIVERSITY
SWEDEN

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

ISSN 1651-4238
ISBN 978-91-7485-151-9
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

Abstract

The interaction of embedded systems with their environments and their resource limitations make it important to take into account properties such as timing, security, and resource consumption in designing such systems. These so-called Extra-Functional Properties (EFPs) capture and describe the quality and characteristics of a system, and they need to be taken into account from early phases of development and throughout the system's lifecycle. An important challenge in this context is to ensure that the EFPs that are defined at early design phases are actually preserved throughout detailed design phases as well as during the execution of the system on its platform.

In this thesis, we provide solutions to help with the preservation of EFPs, targeting both system design phases and system execution on the platform. Starting from requirements, which form the constraints of EFPs, we propose an approach for modeling Non-Functional Requirements (NFRs) and evaluating different design alternatives with respect to the satisfaction of the NFRs. Considering the relationship and trade-off among EFPs, an approach for balancing timing versus security properties is introduced. Our approach enables balancing in two ways: in a static way resulting in a fixed set of components in the design model that are analyzed and thus verified to be balanced with respect to the timing and security properties, and also in a dynamic way during the execution of the system through runtime adaptation. Considering the role of the platform in preservation of EFPs and mitigating possible violations of them, an approach is suggested to enrich the platform with necessary mechanisms to enable monitoring and enforcement of timing properties. In the thesis, we also identify and demonstrate the issues related to accuracy in monitoring EFPs, how accuracy can affect the decisions that are made based on the collected information, and propose a technique to tackle this problem. As another contribution, we also show how runtime

monitoring information collected about EFPs can be used to fine-tune design models until a desired set of EFPs are achieved. We have also developed a testing framework which enables automatic generation of test cases in order verify the actual behavior of a system against its desired behavior.

On a high level, the contributions of the thesis are thus twofold: proposing methods and techniques to 1) improve maintenance of EFPs within their correct range of values during system design, 2) identify and mitigate possible violations of EFPs at runtime.

Sammanfattning

Interaktionen mellan inbyggda system och dess miljöer gör det viktigt att ta hänsyn till egenskaper såsom timing, säkerhet, och tillförlitlighet vid utformning av sådana system. Dessa så kallade Extra-Funktionella Egenskaper (EFE:er) innefattar och beskriver kvalitet och egenskaper hos ett system, och måste beaktas tidigt i utvecklingsfasen samt under hela systemets livscykel. En central utmaning i detta sammanhang är att säkerställa att de EFE:er som är definierade i tidiga designfaser bevaras genom de detaljerade konstruktionsfaserna så väl som under exekveringen av systemet på sin plattform.

I denna avhandling tillhandahåller vi lösningar för att stödja bevarandet av EFE:er, med inriktning på både systemkonstruktionsfaserna och plattformsexekvering. Med utgångspunkt från kraven, som begränsar EFE:erna, föreslår vi en strategi för modellering av Icke-Funktionella Krav (IFK) och utvärdering av olika designalternativ med avseende på uppfyllandet av IFK. Med kopplingen och avvägningen emellan EFE:er i åtanke, introduceras ett tillvägagångssätt för att balansera tidsegenskaperna gentemot säkerhetsaspekterna. Vår metod gör det möjligt att balansera på två sätt: statiskt, som resulterar i en fast uppsättning komponenter i konstruktionsmodellen som analyseras och därigenom verifieras som balanserade med avseende på tids och säkerhetsegenskaper, samt dynamiskt genom anpassning under systemexekvering. Med hänsyn till plattformens roll i bevarandet av EFE:er samt mildrande av eventuella kränkningar, föreslås ett sätt för att berika en plattform med nödvändiga mekanismer för att möjliggöra övervakning samt säkerställande av tidsegenskaper. I avhandlingen både identifierar och demonstrerar vi problematiken med noggrannhet i övervakning av EFE:er, och hur noggrannheten kan påverka de beslut som fattas grundat på insamlad information, samt föreslår en teknik för att lösa detta problem.

Som ytterligare bidrag visar vi också på hur information om EFE:er som insamlats genom övervakning under drift kan användas för att finjustera designmodeller tills en önskad uppsättning EFE:er uppnås. Vi har också utvecklat ett ramverk för testning som möjliggör automatisk generering av testfall för kontrollera att faktiskt beteende hos ett system överensstämmer med önskat beteende.
Som helhet är således avhandlingens bidrag dubbelt: föreslagna metoder och tekniker till att 1) förbättra bevarandet av EFE:er inom tillåtet intervall under systemdesign, 2) identifiera och mildra eventuella överträdelser av EFE:er under körning. Ur detta perspektiv bidrar våra lösningar till att producera inbyggda system med bättre kvalitetssäkring.

Dissertation Opponent:

- Assoc. Prof. Vittorio Cortellessa - University of L'Aquila, Italy.

Grading Committee:

- Prof. Antonia Bertolino - National Research Council (CNR), Italy.
- Prof. Jan Bosch - Chalmers University of Technology, Sweden.
- Adj. Prof. Tiberiu Seceleanu - ABB Corporate Research, Sweden.

Grading Committee Reserve:

- Prof. Kristina Lundqvist - Mälardalen University, Sweden.

PhD Advisors:

- Prof. Mikael Sjödin - Mälardalen University, Sweden.
- Dr. Antonio Cicchetti - Mälardalen University, Sweden.

To my dear family,

Massoud, Forough, Mahnaz, Farshid & Farshad.

“A PhD is someone who knows everything about something & something about everything” - anonymous.

Acknowledgements

The journey towards a PhD degree is full of joy, ups and downs, excitements, and challenges. There have been many people who have been with me throughout this journey; people from which I have learned a lot, shared our joys and excitements together, those that together we worked countless hours to solve problems and tackle challenges, those who provided support and made the progress smoother and easier, and those people whose mere acquaintance and presence have been a great source of inspiration and motivation.

Hereby, I would like to thank my supervisors Mikael Sjödin and Antonio Cicchetti for their support, encouragement, and all the things that I learned from them helping me become a better researcher. Thanks to Radu Dobrin, Jan Carlson and Cristina Seceleanu for their invaluable comments and tips. I had the pleasure of sharing the office with Federico and Antonio with whom I have also great and unforgettable memories from all the travels that we did together. I would also like to thank my managers and colleagues at Alten and Enea, particularly Detlef Scholle.

The success of Mälardalen Real-Time Research Centre (MRTC) at IDT with its friendly, pleasant and enriching environment is due to the hard work and presence of many great people and researchers and I am glad that I have had the chance to be part of such an environment. My studies at MDH also gave me the opportunity to meet new friends and work with many wonderful people. I would like to thank them all here for the all the great moments. Thanks also to the ITS-EASY graduate school staff for their support, and the nice educational events they organized.

My deepest gratitude to my family who have always been there for me. Without them I could have never reached this far.

Mehrdad Saadatmand
Västerås, February 2015

List of Publications

Papers Included in the PhD Thesis[1]

Paper A *Model-Based Trade-off Analysis of Non-Functional Requirements: An Automated UML-Based Approach.* Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin. Journal of Advanced Computer Science, Vol. 3, No. 11, November, 2013.

Paper B *Managing Timing Implications of Security Aspects in Model-Driven Development of Real-Time Embedded Systems.* Mehrdad Saadatmand, Thomas Leveque, Antonio Cicchetti, Mikael Sjödin. International Journal On Advances in Security, Vol. 5, No. 3&4, December, 2012.

Paper C *Monitoring Capabilities of Schedulers in Model-Driven Development of Real-Time Systems.* Mehrdad Saadatmand, Mikael Sjödin, Naveed Ul Mustafa. 17th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA), Krakow, Poland, September, 2012.

Paper D *An Automated Round-trip Support Towards Deployment Assessment in Component-based Embedded Systems.* Federico Ciccozzi, Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin. 16th International Symposium on Component-Based Software Engineering (CBSE), Vancouver, Canada, June, 2013.

[1]The included articles have been reformatted to comply with the PhD thesis layout.

Paper E *Towards Accurate Monitoring of Extra-Functional Properties in Real-Time Embedded Systems.* Mehrdad Saadatmand, Mikael Sjödin. 19th Asia-Pacific Software Engineering Conference (APSEC), Hong Kong, December, 2012.

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

Paper G *Testing of Timing Properties in Real-Time Systems: Verifying Clock Constraints.* Mehrdad Saadatmand, Mikael Sjödin. 20th Asia-Pacific Software Engineering Conference (APSEC), Bangkok, Thailand, December, 2013.

Related Publications not Included in the PhD Thesis

Licentiate Thesis[2]

1. *Satisfying Non-Functional Requirements in Model-Driven Development of Real-Time Embedded Systems.* Mehrdad Saadatmand, Licentiate Thesis, ISSN 1651-9256, ISBN 978-91-7485-066-6, May, 2012.

Conferences & Workshops

1. *A Fuzzy Decision Support Approach for Model-Based Tradeoff Analysis of Non-Functional Requirements.* Mehrdad Saadatmand, Sahar Tahvili. 12th International Conference on Information Technology : New Generations (ITNG), Las Vegas, USA, April, 2015.

2. *Mapping of State Machines to Code: Potentials and Challenges.* Mehrdad Saadatmand, Antonio Cicchetti. The Ninth International Conference on Software Engineering Advances (ICSEA), Nice, France, October, 2014.

3. *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 (SEAA), Verona, Italy, August, 2014.

4. *Runtime Verification of State Machines and Defect Localization Applying Model-Based Testing.* Mehrdad Saadatmand, Detlef Scholle, Cheuk Wing Leung, Sebastian Ullström, Joanna Fredriksson Larsson. First Workshop on Software Architecture Erosion and Architectural Consistency (SAEroCon) (ACM) (Co-located with WICSA 2014), Sydney, Australia, April, 2014.

5. *Run-Time Monitoring of Timing Constraints: A Survey of Methods and Tools.* Nima Asadi, Mehrdad Saadatmand, Mikael Sjödin. The Eighth International Conference on Software Engineering Advances (ICSEA), Venice, Italy, October 27 - November 1, 2013.

[2] A licentiate degree is a Swedish graduate degree halfway between M.Sc. and Ph.D.

6. *On Combining Model-Based Analysis and Testing.* Mehrdad Saadatmand, Mikael Sjödin. 10th International Conference on Information Technology : New Generations (ITNG), Las Vegas, USA, April, 2013.

7. *Toward Model-Based Trade-off Analysis of Non-Functional Requirements.* Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin. 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Cesme-Izmir, Turkey, September, 2012.

8. *Modeling Security Aspects in Distributed Real-Time Component-Based Embedded Systems.* Mehrdad Saadatmand, Thomas Leveque. 9th International Conference on Information Technology : New Generations (ITNG), Las Vegas, USA, April, 2012.

9. *Design of Adaptive Security Mechanisms for Real-Time Embedded Systems.* Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin. 4th International Symposium on Engineering Secure Software and Systems (ESSoS), Eindhoven, The Netherlands, February, 2012.

10. *UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems.* Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin. The Sixth International Conference on Software Engineering Advances (ICSEA), Barcelona, Spain, October, 2011

11. *On Generating Security Implementations from Models of Embedded Systems.* Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin. The Sixth International Conference on Software Engineering Advances (ICSEA), Barcelona, Spain, October, 2011.

12. *Enabling Trade-off Analysis of NFRs on Models of Embedded Systems.* Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin. 16th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA), WiP session, Toulouse, France, September, 2011.

13. *A Methodology for Designing Energy-aware Secure Embedded Systems.* Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin. 6th IEEE International Symposium on Industrial Embedded Systems (SIES), Västerås, Sweden, June, 2011.

14. *Toward a Tailored Modeling of Non-Functional Requirements for Telecommunication Systems.* Mehrdad Saadatmand, Antonio Cicchetti, Diarmuid Corcoran, Mikael Sjödin. 8th International Conference on Information Technology : New Generations (ITNG), Las Vegas, USA, April, 2011.

15. *On the Need for Extending MARTE with Security Concepts.* Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin. 2nd International Workshop on Model Based Engineering for Embedded Systems Design (M-BED), Grenoble, France, March, 2011.

Contents

I

Thesis

Chapter 1

Introduction

Once upon a time programmers used to get excited when they found out that the programs they had punched on a card just worked and could perform the expected calculations and functions. Although the computers back then also had their own limitations in terms of memory and processing capacity, getting the right functionality done was a big enough challenge in itself, which also meant not needing to redo programming on another punchcard. With respect to complexity, computer systems today are rapidly becoming more and more complex. There are several factors that contribute to this complexity. One factor is the growing demands and expectations on the services provided by these systems. For instance, more tasks are delegated to software and electric/electronic components in a car nowadays, which used to be performed solely by mechanical and hydraulic parts. On the other hand, the complexity of each service itself is increasing as well. Another aspect that contributes to the complexity issue is the complexity and variety of the infrastructure and platforms on which systems are deployed; i.e., different operating systems and frameworks (e.g., Android, iOS, .NET, JVM), different hardware (e.g., single core, multicore, 32/64-bit processors). In case of embedded systems, limitations and constraints on available resources also add another dimension to the complexity issue. This means that such systems should operate within certain constraints. In other words, other concerns than just logical correctness of operations can play an important role in determining the success and correctness of embedded systems. These limitations and constraints are captured in the form of

Non-Functional Requirements (NFRs) and Extra-Functional Properties (EFPs). However, EFPs like NFRs [1] cannot be considered individually as they have interdependencies and mutual impacts and tuning one EFP can affect another one.

The ultimate goal in the design of a software system is to deliver a product which satisfies all the requirements of different stakeholders. To assess the fulfillment of this goal, not only it is needed to be able to verify certain properties in the end product (e.g., timing properties), but also it is important from early design phases to consider a set of system components, among different alternatives, that have desired properties in line with and contributing to the overall satisfaction of system requirements and not violating them. One important challenge in this context is that as we build a system and go down the abstraction levels and finally get to its execution, the properties of interest which are related and relevant to the satisfaction of system requirements should be *preserved* and not deviate from their desired values. For example, if at runtime the execution time of a task exceeds a certain threshold and value, it can impact the satisfaction of a timing requirement on end-to-end response time in the system.

Considering the aforementioned points, two challenges with respect to EFPs in embedded systems can be identified: 1-coming up with a system design (e.g., models) which respects the desired set of properties (considering their interdependencies and trade-offs); and 2-making sure that those properties remain valid in the final product and during execution. The solutions that are provided in this thesis under the title of preservation of extra-functional properties mainly tackle these two issues. The importance of the contributions of this thesis lies in the fact that regardless of the type and amount of analyses done in building a system, if there are any deviations between the expected EFPs and the actual ones at runtime, the developed system might be then a failure.

1.1 Background and Motivation

The number of computer systems that we use in our daily life which are embedded as part of other devices and systems is rapidly growing. Examples of such systems are microwave ovens, automobiles, TV sets, digital cameras, and refrigerators. Embedded computer systems are systems that are designed basically to control and operate as part of other

devices. These systems are usually designed for specific and dedicated operations, and interact with their external environment through sensors and actuators [2, 3]. This interaction often brings along additional requirements such as real-time and safety. However, besides such requirements, resource constraints in these systems also introduce other requirements with respect to power consumption, processing capacity, memory usage, etc. Due to resource constraints that these systems have, their correctness depends not only on providing the right functionality but also respecting the constraints that they have. For this reason, it is of great importance to be able to evaluate EFPs and preserve them within their acceptable range of values mitigating possible violations of them at runtime.

In this context, requirements serve as a means for capturing and expressing the needs of different stakeholders of the system. Considering the variety of stakeholders, requirements can be originated from different sources; such as customers and end users, standards and regulations, developers and refinement of higher level requirements, development tools, operational context and environment of the system, and so on. As for Non-Functional Requirements (NFRs), they have specific challenges which can make their satisfaction in a system a complicated task [4, 5, 6]. For instance, NFRs cannot usually be considered in isolation as they are interconnected and have dependencies, and also can span different aspects and parts of a system. Therefore, in satisfying an NFR, its impacts on other NFRs should also be taken into account, and trade-off analysis among NFRs needs to be done [7, 8]. A similar procedure needs to be done for EFPs when adjusting and tuning different properties of the system.

There is a clear and important (but sometimes misunderstood) relationship between NFR and EFP. For example, 'response time of a component should not exceed 2ms' is a requirement, while 'response time of component A never exceeds 2ms' or 'response time of component B is equal to 1ms' are expressions of properties. It is only in relation to a requirement that it becomes possible to then talk about validity/invalidity or 'goodness/badness' of the value of a property. For instance, just knowing that a software component has the worst-case execution time of 200ms does not help in determining if it is suitable to be used in building a particular system or not. Only when the requirements are taken into account, this value gets meaning in the sense that if it is good for that system and context or not; in which case, another component

with a different worst-case execution time might be adopted. Similarly, when monitoring and evaluating EFPs, requirements are needed in order to determine if the value of an EFP is valid and acceptable or not. Balancing trade-offs among requirements can also incur adjusting system properties that are related to those requirements. Therefore, there is a relationship between an NFR and EFPs in a system and in order to satisfy an NFR, its related extra-functional properties should have valid values. For example, to satisfy performance requirements in a real-time system, execution and response time values of tasks (among others) should remain within a valid range.

To tackle the design complexity issue of embedded systems, Model-Driven Development (MDD) is a promising approach. It aids by raising abstraction level and reducing design complexity, which also enables analysis at earlier phases of development. This helps with the identification of problems before the implementation phase [9, 10], noting that the cost of problems when found at later phases, especially in the code and at runtime, grows exponentially [11, 12, 13]. The implementation of the system can also be generated from the design models through (a set of) model transformations. As more abstraction levels are introduced in designing a system, it becomes also important to verify consistency of design artifacts and their properties at each level. This means that for each transformation, input properties should be preserved in the output of the transformation. This also includes the system execution at runtime which is the result of executing the generated source code, implying that the execution platform should be able to actively monitor and preserve extra-functional properties at runtime. To this end, the execution platform also requires to be *semantically aware* of the specified properties and related events in order to monitor them and detect their deviations. In the context of MDD, preservation of EFPs also gains special importance and interest as it is ultimately the object code which is executing and controlling a system, not models per se. Moreover, in terms of timing properties, for instance, "models are only approximation of system implementations" and therefore, "inevitable time deviations of the implementation from the model appear" [14, 15].

Regardless of the applied development method and despite all types of analyses done at earlier phases, the ultimate goal is to have a system which behaves correctly and as intended at runtime and during its execution. Moreover, for performing static analysis different assumptions are taken into account. At runtime, situations may still occur that lead to

the violation of those assumptions which in turn can mean invalidation of analyses' results [16, 17]. This again emphasizes the need for preservation of EFPs which constitute such assumptions, such as worst-case execution times of tasks, etc. For instance, it is very common nowadays that modern CPUs have Dynamic Voltage and Frequency Scaling (DVFS) support for power management purposes. When DVFS is applied, the CPU can go down to a lower frequency which then affects the execution times of tasks. In such a scenario, the results of timing analysis done assuming certain execution times of tasks based on a different CPU frequency may not be valid anymore. Of course, in an ideal case, all such scenarios should be considered in the analysis. However, such extensive and exhaustive analyses may not always be economical or even possible to perform, particularly in complex systems.

1.2 Problems and Contributions Overview

To provide support for preservation of EFPs, we first start from NFRs, and considering the relationships between different NFRs we introduce a method to evaluate their interdependencies and mutual impacts in a generic manner. This step is necessary considering the relationship of NFRs and EFPs as discussed in the previous section. Besides, early analysis of NFRs is also important considering that different sets of NFRs on the same set of functional requirements can result in different design decisions and implementations. For instance, to sort some data under a specific timing constraint, only certain sorting algorithms may be suitable to use. However, if that timing constraint is relaxed but instead there is a constraint on maximum memory usage, a different set of sorting algorithms can then be considered as suitable. Moreover, due to resource limitations, the NFRs that are defined for an embedded system by different stakeholders need to be balanced and trade-off analysis among them should be performed. As the next step in building a system which respects its specified constraints, we introduce an approach for establishing balance among different EFPs in an embedded system. This approach is demonstrated by looking particularly into the relationship between security and timing.

Another challenge is collecting information about each EFP in the system. In other words, appropriate mechanisms for monitoring EFPs are needed in order to determine whether a violation with respect to the

constraints of an EFP has occurred or not. Such monitoring information can then be used for different purposes: for example, to perform runtime adaptation, enforcing the system constraints, testing EFPs, or simply providing feedback on EFPs values collected during the system execution. Along with this goal, we have developed a method for collecting necessary information about the timing behavior of real-time systems to identify violations such as deadline misses and execution time overruns of real-time tasks. In this thesis, we also discuss and demonstrate a method how such monitoring capabilities and the EFPs information collected using them can help with adjusting system models towards reaching deployment configurations which ultimately result in a desired set of EFPs' values at runtime.

Moreover, we demonstrate how monitored information can be used to perform runtime adaptation and also test a system with respect to its timing properties. For the former, a runtime adaptation mechanism has been developed to balance timing aspects of a system versus its security level by adopting different encryption algorithms with different timing properties. For the latter, we have developed a testing approach to verify whether the timing constraints (e.g., specified in the form of clock constraints) in a system get violated at runtime or not.

As can be understood so far, one fundamental feature needed in achieving preservation of EFPs and to mitigate their possible violations is the ability to monitor and collect information about EFPs. While such capability is assumed as provided in many previous works, it has its own unique challenges. For instance, EFPs are of different nature and kind, and therefore, the way that necessary information is collected for each one is a challenge in itself. Also accuracy of the collected information plays an important role when deciding if an EFP is within an acceptable range (as specified by an NFR). As another contribution of the thesis, we demonstrate the importance of this issue, how it can affect taking correct decisions about EFPs in a system, identify the factors that can contribute to the accuracy of the collected information (or lack thereof), and provide a solution for mitigating it.

In summary, the main goal of the thesis is to highlight the challenges with respect to the preservation of EFPs and provide techniques and methods applicable at different levels of abstraction to increase our confidence that at the final step, namely runtime and during execution, EFPs do not deviate from their expected values and the system behaves as intended. While the proposed solutions have the potential to be ap-

plied for various EFPs, our main focus in this work is particularly on timing properties.

1.3 Thesis Outline

The thesis consists of two parts: **Part I** includes five chapters. Chapter 1 provided an introduction to the thesis, described the background, and main problems that this thesis addresses as well as an overall description of the thesis contributions. Chapter 2 elaborates the research context, goals, and process. Contributions of the thesis are described in more detail in Chapter 3. Chapter 4 discusses the related work and in Chapter 5, the work is summarized and future directions are mentioned. **Part II** of the thesis contains the published peer-reviewed papers that constitute and present the technical contributions of the thesis in detail, which are organized in Chapters 6-12.

Chapter 2

Research Context

The main objective in this research work is to introduce methods for preservation of extra-functional properties in embedded systems development and to mitigate their possible violations at runtime. This goal is based on the following principles and assumptions:

- For each EFP to be preserved, a set of values containing acceptable and valid values for that EFP can be considered. Values that fall outside of this set are considered invalid, and thus violate the constraints of the EFP (considering a specific context).

- Analysis can be performed on system models in order to evaluate satisfaction feasibility of its non-functional requirements and to evaluate/calculate extra-functional properties of the system and its components, such as response time. Analyses are based on some assumptions and preconditions. Violation of these assumptions can lead to having invalid and erroneous analysis results.

- At runtime, several factors such as transient loads, difference between the ideal execution environment (taken into account for analysis) and the actual one, can lead to the violation of the assumptions that were used to perform analysis [16].

- It may not be practical and/or economical to perform analysis on all types of extra-functional properties. In such cases, runtime monitoring becomes even more important.

- As we go down the abstraction levels, an EFP may be refined and translated into one or more EFPs at the next levels. In that case, preservation of the former can incur preservation of the latter at a different abstraction level.

In this context, the term *preserve* that we use in our work implies that there is some knowledge about valid and invalid values (e.g., range of values) that an extra-functional property can have in a specific context and system; i.e., we actually preserve the *validity* of the value, rather than the actual value.

2.1 Research Goals

To achieve the objective of the work, the following main research goals have been defined:

- **G1: Define an approach for establishing balance among NFRs at the model level:** The relationship between NFRs and EFPs makes it important to start from NFRs and evaluate and balance them. For example, a requirement on end-to-end deadlines and response times in a system might require use of certain components with specific timing properties. If that requirement is changed, another set of components with different timing properties might be needed.
- **G2: A model-based EFP-aware method resulting in property preserving system designs:** Model-Based Development can enable identification of problems earlier in the development phase. By exploiting this feature, system models can be analyzed and adjusted to have extra-functional properties within their desired and acceptable range of values, and thus, support EFP preservation from earlier phases of development.
- **G3: A framework for monitoring and testing of extra-functional timing properties:** Without monitoring EFPs at runtime we cannot be sure whether they are preserved or not (somehow analogous to the Schrödinger's cat experiment [18]). A monitoring mechanism is also needed if a system requires to apply enforcement of EFPs (for instance, to preempt a task before it exceeds its worst-case execution time). Moreover, it should be

possible to test a system with respect to its EFPs, which requires to obtain and know the value of an EFP. In this research, we limit ourselves to timing properties while an industrial tool should support a wider range of EFPs.

2.2 Research Process

The main steps that have taken place in performing this research work are summarized and illustrated in Figure 2.1.

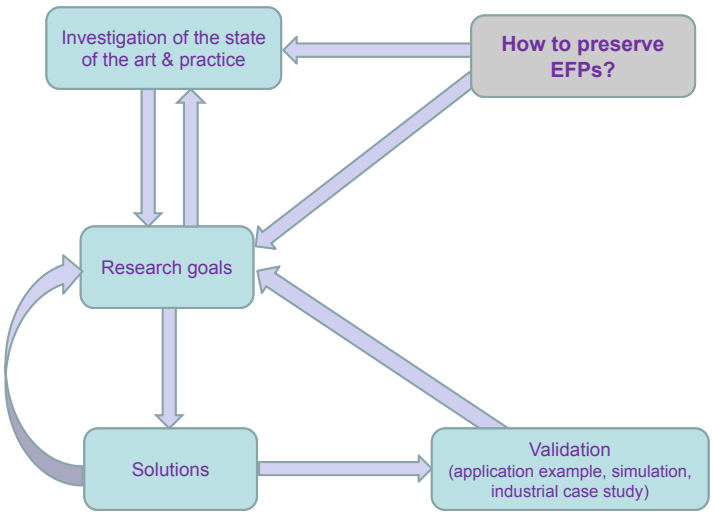

Figure 2.1: Research Steps

To achieve the overall objective of the thesis, namely preservation of EFPs, several research goals have been defined. Moreover, the state of the art and practice have been consulted, which resulted in additional research (sub-)goals or modification of already defined ones. To fulfill the research goals, solutions have been proposed. These solutions were

validated in different ways: by an application example to demonstrate how, for instance, a method is applied; by simulation; or by performing an industrial case study. The arrow in the figure from `Validation` to `Research Goals` indicates that in validating the solutions, new research goals or modification of existing ones have also formed and occurred.

Chapter 3

Contributions

In this section we provide the description of the main contributions of this work:

C1) Trade-off analysis and evaluation of conflicts among NFRs: To model interdependencies of NFRs and evaluate their impacts on each other, we have proposed a UML profile. The profile offers necessary concepts to generically model an NFR along with its refinements which can include one or several other NFRs as well as functional parts that contribute to its satisfaction. This way, it enables to create a hierarchy of NFRs, form child-parent relationship among them, and also establish relationships to the functional elements in the model that provide realization and implementations for NFRs. To enable trade-off analysis of NFRs in a quantitative manner, we introduce numerical properties as part of the defined stereotypes in the profile. This allows to calculate the satisfaction level of an NFR by taking into account both the contribution degree of each of its children NFRs and any impacts that other NFRs in the system may have on it. To automate and perform the analysis, an in-place model transformation is used to traverse model elements, perform necessary calculations based on the algorithms that are implemented as part of the transformations, and then update the respective properties of model elements with the calculated values. Based on the analysis results, it can be determined if the planned decision designs to fulfill NFRs are acceptable or not.

In the profile, we also introduce the concept of *deviation indicator*

(in Paper A in the thesis) for requirements, which helps to identify parts of a system which have greater degree of deviation with respect to satisfaction of an NFR. This provides for several interesting features, such as to prioritize test cases and focus testing activities on parts of the system with higher deviation indicator values (i.e., probably having more 'severe' problems) as demonstrated in a separate work of ours in [19]. This is particularly interesting and beneficial considering that there is only limited amount of resources (e.g., time or money) available for performing testing activities. Such prioritization based on the deviation indicator values allows for more efficient use of available resources for testing.

Another variation of our approach based on the NFR profile is presented in [20]; in which we have used fuzzy logic and decision support systems to identify best design alternatives (e.g., from a set of available components with different properties) in order to construct a system design which is optimized with respect to the overall satisfaction level of its NFRs. Application of fuzzy logic helps to relax the need for providing accurate quantified values for relationships among NFRs, as is expected in the original approach.

C2) Establishing balance between security and timing properties: The choice of security mechanisms in real-time embedded systems where timing requirements are critical may not be trivial. For example, performing encryption using a strong encryption algorithm may take longer time than using a weaker (but faster) encryption algorithm and this may lead to the violation of timing requirements. Therefore, in implementing security requirements, timing implications of the chosen security mechanisms should also be considered.

In this thesis, we provide two approaches to tackle this challenge. In the first one, we address it at the model level by identifying parts of the system (i.e., sensitive data) that need to be protected. For such parts, appropriate security features (in here, encryption and decryption components) to protect them are added to the original component model of the system. Then timing analysis is performed on the derived component model to ensure that the security features which are added do not violate the timing requirements. In this approach, the original component model of the system is used as input for a transformation that considers the sensitive data flows in the input model and adds appropriate security components considering their timing properties. This approach leads to

a static design in the sense that a fixed and particular security mechanism which is analyzed and thus, known to respect its allowed time budget is always used in each execution. Our approach also provides a form of separation of concerns in the sense that instead of defining security on the architectural (i.e., component) model, security engineers (or system designers) can now focus and annotate the data model, based on which the component model is then updated (through model transformation) to include appropriate security mechanisms. Moreover, in this way we also enable to bring security concerns into earlier phases of development. Considering security at earlier phases of development is a need which is getting more and more attention in computer systems today, particularly in the embedded domain where security has its own unique challenges (such as timing implications as we already discussed) [21].

However, the aforementioned solution may not be practical for systems with high complexity which are hardly analyzable or systems with unknown timing properties of their components. For such systems, an adaptive approach to select appropriate security mechanisms based on the state of the system can be used to adapt its behavior at runtime and stay within the timing constraints. To this end, as a second and complementary way to establish balance between security and timing, we have suggested an approach for selecting appropriate encryption algorithms (in terms of their timing behaviors) at runtime in an adaptive fashion. The approach works by keeping a log for each execution of the encryption process. Based on the logged information, if a timing violation is observed, a more suitable encryption algorithm (in terms of timing properties) is adopted in the next execution. In other words, the system adapts itself at runtime to keep the timing constraints and reduce the number of timing violations. The idea behind this adaptation mechanism is that when it is detected that an executing encryption algorithm is exceeding its allowed time budget, it can be more costly to terminate it in the middle of the encryption process and restart encrypting the data with a less time-consuming encryption algorithm. Instead we let it finish its job and then use another encryption algorithm with a lower execution time in the next invocation of the encryption process.

In summary, the former approach aims to help with establishing balance between security and timing properties at the design phase when system models are being created. In the second approach, this balance is established in an adaptive way at a later phase, namely during system

execution and at runtime. These two approaches can of course be used together in building a system in order to provide a higher degree of assurance to keep timing and security properties balanced and at acceptable levels.

C3) Runtime monitoring and enforcement of EFPs: The capabilities of platforms play an important role in preservation of extra-functional properties and mitigation of possible violations. To be able to actively monitor and enforce EFPs, the platform needs to be semantically aware of them. With respect to timing properties, this awareness can include properties of a real-time task such as activation pattern (i.e., periodic, sporadic, aperiodic), deadline, execution time, and so on. The platform needs to be aware of such properties so that, for example, it can detect deadline misses or execution time overruns. This also implies that the platform needs to have respective monitoring mechanisms for different timing properties. We have constructed a framework that brings such awareness about several timing properties to the platform and provides mechanisms for runtime monitoring and enforcement of such properties.

Moreover, in the context of a model-based development method, it is important to verify that the actual values of EFPs at runtime are in compliance with those expected and defined at the model level. The importance of such capability becomes more clear remembering that the end goal of all different development methods and techniques is to have a system which executes and behaves as expected at runtime with respect to both functional and extra-functional aspects. For this purpose, we show how having a monitoring framework enables to also build and provide a round-trip support in the development chain, constituting from design models down to code and its execution on the platform and back again to the models. In such a round-trip chain, monitoring information is propagated back to the design models where the expected values are compared with the actual values. If necessary, refinements and modifications are done to the model(s) in order to finally have a system with the desired set of EFPs. This is done by performing the process several times until the refined models and the code that is generated from them result in a system with satisfactory set of EFPs. This approach may well be used towards optimizing design models for specific properties. We have applied and validated our proposed round-trip method on the Ericsson's ATM Adaptation Layer 2 (AAL2) subsystem.

C4) Testing of timing properties and runtime model verification: Another challenge with respect to timing properties is how to test them. Testing EFPs, such as timing, can generally be a tricky task. One factor contributing to this issue is that testers need to collect and have necessary information about timing properties of a system so that they can actually then test it. This brings us back to the previous point about the capability to monitor and observe timing properties. To improve testability of a system with respect to its timing properties, we introduce a method and a testing framework that by exploiting monitoring capabilities of the platform enables testing of timing properties. Of course, testing per se does not guarantee absence of errors, but by detecting failures and fixing them it can help to increase our confidence in the correctness of a system. Our suggested testing approach enables automatic generation of Concrete Test Cases (CTC) (i.e., executable test scripts) from abstract ones (ATC: Abstract Test Cases) which are derived from Timed Automata (TA) models, executing them against the target, and determining timing issues and violations. This is done by implementing a parser which reads in ATCs and generates Python scripts based on them. This way, we not only enable testing of timing properties but also do so in an automatic way. We have applied and validated our approach on the Brake-by-Wire (BBW) use-case provided by Volvo.

The result of testing also indicates if there is any deviation between the intended behavior captured by models and the actual behavior of the system at runtime, which can also be useful in verifying architectural deviations and inconsistencies. This is important as models are also used to perform model-based analysis. Therefore, if they do not correctly represent a system's behavior, the analyses that are done based on them may also not be valid for that system.

Our proposed testing framework is covered by papers F and G in the thesis. In the former, the whole testing framework and test case generation methodology are explained with a focus on functional requirements. In the latter, the extensions to also enable testing of timing requirements are described (with a minor difference in how the System Under Test (SUT) code is implemented for mapping states and transitions to code [22], and a slightly different TA model for describing the internal behavior of system components). In our testing approach, timed auomata are used to describe the internal behavior of the components of the target system which is modeled in EAST-ADL [23]. Figure 3.1 shows the EAST-ADL model of the BBW system.

Figure 3.1: The EAST-ADL model of the BBW system.

The TA model of the Anti-lock Braking System (ABS) component in the BBW system, without its timing and clock constraints, is depicted in Figure 3.2.

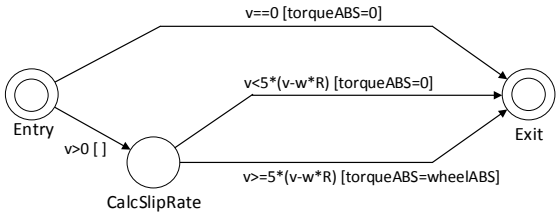

Figure 3.2: The TA description of the ABS function (without timing).

The TA models are analyzed and verified in UPPAAL PORT which also produces trace information constituting a path in the TA model. The trace information is parsed and transformed into executable test scripts which basically check behavioral conformance by verifying that the same set of states and transitions (or nodes and edges) are taken and visited at runtime and in the same order. To take into account timing properties, the time point at which each state is visited at runtime is timestamped. Doing so, it becomes possible to then check, for instance, how long it has taken to go from one state to another. Having such information as part of the test result reports, comparisons can be done against clock constraints and other timing annotations in the TA model, such as the model shown in Figure 3.3 (in which x represents a clock).

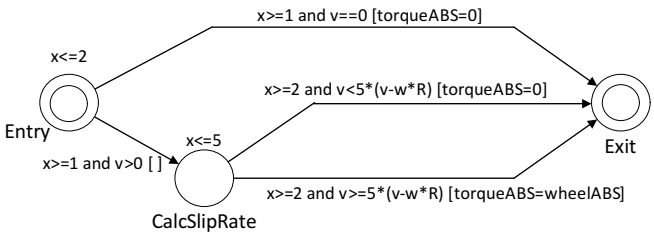

Figure 3.3: The TA description of the ABS function.

Part of a sample internal output (the actual output shown to the user is visualized and represented as HTML) of executing a test script showing the timestamps at each visited state is illustrated in Listing 3.1.

Listing 3.1: Sample internal output from a test script

```
Running tests...
------------------
Checking the behaviour of GlobalBrakeController
Transition 0:
        timestamp = 0:0 (1398176868:107184)
        state = 0
        rpm1 = 8
        rpm2 = 8
        reqTorque = 0
        whlTorque = 0
        v = 0
        rpm3 = 16
        rpm4 = 16
        R = 1

Transition 1:
        timestamp = 0:1 (1398176868:107185)
        state = 1
        rpm1 = 8
        rpm2 = 8
        reqTorque = 0
        whlTorque = 0
        v = 6
        rpm3 = 16
        rpm4 = 16
        R = 1

Transition 2:
        timestamp = 0:4 (1398176868:107188)
        state = 4294967295
        rpm1 = 8
        rpm2 = 8
        reqTorque = 0
        whlTorque = 0
        v = 6
        rpm3 = 16
        rpm4 = 16
        R = 1

....
```

This technique also enables to test other timing properties such as end-to-end response times in the systems. For instance, if there is a requirement on end-to-end response time from the moment that the brake pedal is pressed until the moment that the brake force is applied, the logged timestamp information can be used to calculate the actual end-to-end response time and determine if it is in compliance with the requirement or not. Listing 3.2 shows an excerpt of an Abstract Test Case (ATC) generated from TA models, including a sample requirement on end-to-end response time (specified as 'EndtoEnd = 3').

Listing 3.2: A sample ATC for BBW system

```
##############################################
# [TestCaseSpecification]
# atc1
#
# [RequirementSpecification]
# [Functional]
# E<> C2.VehicleSpeed_kmph_FL#==0 or C2.VehicleSpeed_kmph_FL#<5*(C2.
    VehicleSpeed_kmph_FL#-C2.WheelSpeed_rpm_FL#*C2.R/2) and C2.ABSTorque_FL
    #==0
#
# [RequirementSpecification]
# [Characteristics]
# EndtoEnd = 3
#
# [Purpose]
# -
#
# [Description]
# If VehicleSpeedIn == 0 or slip rate > ABSSlipRateThreshhold, then
    ABSBrakeTorqueOut shall be set to 0Nm.
#
# [PassCriteria]
# PASS if parameter values in the return signal are the same as the expected
     values.
#
# [Environment]
# Local host
#
# [Prerequisites]
# N/A
#
# [AbstractTestCaseDateGenerated]
# 06/05/2014
#
# [GeneratingToolAndVersion]
# UPPAAL PORT v0.48
#
##############################################
[SHORTNAME=atc1]
[ENDTOEND=3]

State:
( GlobalBrakeController.idle ABSFL.idle )
GlobalBrakeController.x=0 ABSFL.x=0 GlobalBrakeController.rpm1=0
      GlobalBrakeController.rpm2=0 GlobalBrakeController.reqTorque=0
      GlobalBrakeController.whlTorque=0 GlobalBrakeController.v=0
      GlobalBrakeController.rpm3=16 GlobalBrakeController.rpm4=16
      GlobalBrakeController.R=1 ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=0
      ABSFL.v=0 ABSFL.R=1

Transitions:
  GlobalBrakeController.idle->GlobalBrakeController.Entry { reqTorque := 0,
       rpm1 := 8, rpm2 := 8, x := 0 }

State:
( GlobalBrakeController.Entry ABSFL.idle )
GlobalBrakeController.x=0 ABSFL.x=0 GlobalBrakeController.rpm1=8
      GlobalBrakeController.rpm2=8 GlobalBrakeController.reqTorque=0
      GlobalBrakeController.whlTorque=0 GlobalBrakeController.v=0
      GlobalBrakeController.rpm3=16 GlobalBrakeController.rpm4=16
      GlobalBrakeController.R=1 ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=0
      ABSFL.v=0 ABSFL.R=1

Delay: 2

State:
( GlobalBrakeController.Entry ABSFL.idle)
GlobalBrakeController.x=2 ABSFL.x=2 GlobalBrakeController.rpm1=8
      GlobalBrakeController.rpm2=8 GlobalBrakeController.reqTorque=0
      GlobalBrakeController.whlTorque=0 GlobalBrakeController.v=0
      GlobalBrakeController.rpm3=16 GlobalBrakeController.rpm4=16
      GlobalBrakeController.R=1 ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=0
      ABSFL.v=0 ABSFL.R=1

...
```

Our testing approach also provides for other interesting features such as defect localization. It means that when a problem is identified in the system, it can be observed in the test result report on the transition between which two states a deviation from the expected behavior has occurred. This way, it provides hints and information about the vicinity of a problem and helps with determining where the root cause of a problem could be in the system.

3.1 Overview of the Included Papers

The main contributions of this thesis are organized and included as a set of published papers as mentioned below. Other papers which can strengthen the contributions of the thesis, but are not included here, were mentioned at the beginning of the thesis; some of which, such as [19, 20], were briefly discussed and cited in the thesis.

- Paper A: Model-Based Trade-off Analysis of Non-Functional Requirements: An Automated UML-Based Approach

  Abstract: One common goal followed by software engineers is to deliver a product which satisfies the requirements of different stakeholders. Software requirements are generally categorized into functional and Non-Functional Requirements (NFRs). While NFRs may not be the main focus in developing some applications, there are systems and domains where the satisfaction of NFRs is even critical and one of the main factors which can determine the success or failure of the delivered product, notably in embedded systems. While the satisfaction of functional requirements can be decomposed and determined locally, NFRs are interconnected and have impacts on each other. For this reason, they cannot be considered in isolation and a careful balance and trade-off among them needs to be established. We provide a generic model-based approach to evaluate the satisfaction of NFRs taking into account their mutual impacts and dependencies. By providing indicators regarding the satisfaction level of NFRs in the system, the approach enables to compare different system design models and also identify parts of the system which can be good candidates for modification in order to achieve better satisfaction levels.

Contribution: I have been the initiator and main author of the paper.

- Paper B: Managing Timing Implications of Security Aspects in Model-Driven Development of Real-Time Embedded Systems

  Abstract: Considering security as an afterthought and adding security aspects to a system late in the development process has now been realized to be an inefficient and bad approach to security. The trend is to bring security considerations as early as possible in the design of systems. This is especially critical in certain domains such as real-time embedded systems. Due to different constraints and resource limitations that these systems have, the costs and implications of security features should be carefully evaluated in order to find appropriate ones which respect the constraints of the system. Model-Driven Development (MDD) and Component-Based Development (CBD) are two software engineering disciplines which help to cope with the increasing complexity of real-time embedded systems. While CBD enables the reuse of functionality and analysis results by building systems out of already existing components, MDD helps to increase the abstraction level, perform analysis at earlier phases of development, and also promotes automatic code generation. By using these approaches and including security aspects in the design models, it becomes possible to consider security from early phases of development and also identify the implications of security features. Timing issues are one of the most important factors for successful design of real-time embedded systems. In this paper, we provide an approach using MDD and CBD methods to make it easier for system designers to include security aspects in the design of systems and identify and manage their timing implications and costs. Among different security mechanisms to satisfy security requirements, our focus in this paper is mainly on using encryption and decryption algorithms and consideration of their timing costs to design secure systems.

  Contribution: I have been the initiator and main author of the paper.

- Paper C: Monitoring Capabilities of Schedulers in Model-Driven Development of Real-Time Systems

  Abstract: Model-driven development has the potential to reduce

the design complexity of real-time embedded systems by increasing the abstraction level, enabling analysis at earlier phases of development, and automatic generation of code from the models. In this context, capabilities of schedulers as part of the underlying platform play an important role. They can affect the complexity of code generators and how the model is implemented on the platform. Also, the way a scheduler monitors the timing behaviors of tasks and schedules them can facilitate the extraction of runtime information. This information can then be used as feedback to the original model in order to identify parts of the model that may need to be re-designed and modified. This is especially important in order to achieve round-trip support for model-driven development of real-time systems. In this paper, we describe our work in providing such monitoring features by introducing a second layer scheduler on top of the OSE real-time operating system's scheduler. The goal is to extend the monitoring capabilities of the scheduler without modifying the kernel. The approach can also contribute to the predictability of applications by bringing more awareness to the scheduler about the type of real-time tasks (i.e., periodic, sporadic, and aperiodic) that are to be scheduled and the information that should be monitored and logged for each type.

Contribution: I have been the initiator and main author of the paper. The implementation was majorly done by Naveed Ul Mustafa.

- Paper D: An Automated Round-trip Support Towards Deployment Assessment in Component-based Embedded Systems

  Abstract: Synergies between model-driven and component-based software engineering have been indicated as promising to mitigate complexity in development of embedded systems. In this work we evaluate the usefulness of a model-driven round-trip approach to aid deployment optimization in the development of embedded component-based systems. The round-trip approach is composed of the following steps: modelling the system, generation of full code from the models, execution and monitoring the code execution, and finally back-propagation of monitored values to the models. We illustrate the usefulness of the round-trip approach exploiting an industrial case-study from the telecom-domain. We use a code-generator that can realise different deployment strategies, as well as special monitoring code injected into the generated code, and

monitoring primitives defined at operating system level. Given this infrastructure we can evaluate extra-functional properties of the system and thus compare different deployment strategies.

Contribution: I have been the second author of the paper, responsible for writing and implementing the monitoring framework part.

- Paper E: Towards Accurate Monitoring of Extra-Functional Properties in Real-Time Embedded Systems

  Abstract: Management and preservation of Extra-Functional Properties (EFPs) is critical in real-time embedded systems to ensure their correct behavior. Deviation of these properties, such as timing and memory usage, from their acceptable and valid values can impair the functionality of the system. In this regard, monitoring is an important means to investigate the state of the system and identify such violations. The monitoring result is also used to make adaptation and re-configuration decisions in the system as well. Most of the works related to monitoring EFPs are based on the assumption that monitoring results accurately represent the true state of the system at the monitoring request time point. In some systems this assumption can be safe and valid. However, if in a system the value of an EFP changes frequently, the result of monitoring may not accurately represent the state of the system at the time point when the monitoring request has been issued. The consequences of such inaccuracies can be critical in certain systems and applications. In this paper, we mainly introduce and discuss this practical problem and also provide a solution to improve the monitoring accuracy of EFPs.

  Contribution: I have been the initiator and main author of the paper.

- Paper F: A Model-Based Testing Framework for Automotive Embedded Systems

  Abstract: Architectural models, such as those described in the EAST-ADL language, represent convenient abstractions to reason about automotive embedded software systems. To enjoy the fully-fledged advantages of reasoning, EAST-ADL models could benefit from a component-aware analysis framework that provides, ideally, both verification and model-based test-case generation capabilities. While different verification techniques have been developed

for architectural models, only a few target EAST-ADL. In this paper, we present a methodology for code validation, starting from EAST-ADL artifacts. The methodology relies on: (i) automated model-based test-case generation for functional requirements criteria based on the EAST-ADL model extended with timed automata semantics, and (ii) validation of system implementation by generating Python test scripts based on the abstract test-cases, which represent concrete test-cases that are executable on the system implementation. We apply our methodology to analyze the ABS function implementation of a Brake-by-Wire system prototype.

Contribution: I have been the second author in this paper, responsible for the parts regarding generation of executable test scripts (called as Concrete Test Case), and their execution on the platform and producing test result reports. I also participated and contributed to the development of the overall testing methodology.

- Paper G: Testing of Timing Properties in Real-Time Systems: Verifying Clock Constraints

  Abstract: Ensuring that timing constraints in a real-time system are satisfied and met is of utmost importance. There are different static analysis methods that are introduced to statically evaluate the correctness of such systems in terms of timing properties, such as schedulability analysis techniques. Regardless of the fact that some of these techniques might be too pessimistic or hard to apply in practice, there are also situations that can still occur at runtime resulting in the violation of timing properties and thus invalidation of the static analyses' results. Therefore, it is important to be able to test the runtime behavior of a real-time system with respect to its timing properties. In this paper, we introduce an approach for testing the timing properties of real-time systems focusing on their internal clock constraints. For this purpose, test cases are generated from timed automata models that describe the timing behavior of real-time tasks. The ultimate goal is to verify that the actual timing behavior of the system at runtime matches the timed automata models. This is achieved by tracking and time-measuring of state transitions at runtime.

  Contribution: I have been the initiator and main author of the paper.

Table 3.1 shows how the papers cover research goals and contributions of the thesis.

Thesis Paper	Contribution	Research Goal
A	C1	G1
B	C2	G2
C	C3 & C4	G3
D	C3	G2
E	C3 & C4	G3
F	C4	G3
G	C4	G3

Table 3.1: Mapping of papers to the research goals and contributions of the thesis

Chapter 4

Related Work

Property preservation: The importance of property preservation is acknowledged and discussed in different contexts in building software systems. In [24], the authors confirm that "the model can only be an approximation of the implementation w.r.t. the timing behavior. It is difficult to guarantee that the issuing time of an event in the implementation is exactly the same as that in the model". Based on this observation, in [24], they provide an approach and demonstrate that the real-time properties of the implemented system can be predicted from the properties of its (timed state/action sequences) model, when the time deviation is bounded. As an extension of this work, in [25] the authors introduce an approach for strengthening property preservation between model and implementation by imposing urgency on the execution of observable actions (than the execution of unobservable ones). They define the notion of distance "as a metric to express the strength of observable property preservation between model and implementation" [25]. Using this metric, they show that by applying the aforementioned approach and executing observable actions before unobservable ones, a smaller distance to the model than any other implementation of the same model can be obtained. From this aspect, their work can also be relevant and applicable for the issue of accuracy in monitoring EFPs that we have discussed in this thesis and the technique we introduced to tackle it [26].

One scenario where deviation of system properties can occur is in performing model transformations (horizontal or vertical). In this thesis we did not focus on property preserving model transformations and

consider it as a future work to complement the contributions of the thesis. There are, however, various works in the literature that discuss and provide solutions for this problem and for verifying the correctness of transformations. In [27], Vallecillo et al. discuss the importance of model transformation correctness and the issues related to specification and testing of transformations. They introduce the concept of *tract* as a generalization of model transformation contracts and a mechanism to specify and capture the expected behavior of a model transformation. Using tracts, they then generate test cases and perform testing of model transformations in a black-box fashion. Based on the concept of tracts, in [28], a static and white-box fault localization method is presented which helps to identify problematic rules in model transformations. In [29], an investigation on techniques based on finite model theory is done to show the use of algebraic co-limits (from category theory) in preservation of certain logical properties (consistency related) in model merging transformations. REFINER [30] is a tool consisting of a set of techniques for verification of behavioral transformations of formal models of concurrent systems. The tool can analyze transformations to determine if semantics of the input model and also given safety and liveness properties are preserved or not. Another work that can be consulted for semantic preserving model transformations is [31] by Mathias Hülsbusch et al. in which a direct bisimulation proof and borrowed context technique are used and compared as two different ways in order to show semantic preservation for a transformation.

NFR Framework: One of the fundamental works in addressing NFRs in development of systems, identifying their impacts and conflicts, and performing trade-off analysis on them, is the NFR Framework [32]. In this framework, NFRs are represented as *softgoals* which are to be *satisficed*. The notion of softgoal is used as a looser notion of goals to indicate the absence of a clear-cut criterion for satisfaction of non-functional requirements [33]. Similarly, the term satisfice is used to indicate that there is sufficient positive evidence and little negative evidence for the satisfaction of an NFR. An NFR is considered *unsatisficeable* when the opposite of the above condition holds. Development techniques for achieving NFRs are defined as *operationalization* which include (but are not limited to) operations, functions and data. Besides NFR softgoals and operationalizating softgoals, NFR framework introduces *claim softgoals* which convey the rationale and argument for or

against a design decision. In refining and decomposing softgoals, the relationships between them are established in the form of 'AND' and 'OR' contributions. An 'AND' contribution means that all the sub-softgoals are needed in order to achieve a softgoal at a higher level in the hierarchy. The structure that is produced as the result of the decomposition and refinement process is called Softgoal Interdependency Graph (SIG). The NFR UML profile that we introduced in this thesis is inspired by the NFR Framework and the concept of *feature* in our NFR profile is similar to what NFR Framework calls *operationalization*. One major difference, though, is that NFR Framework can only support a qualitative form of analysis of NFRs while our profile along with its automated analysis mechanism enables to do so in a quantitative manner. Moreover, the concepts we have suggested as part of the NFR Profile enable to capture more detailed information for each NFR in the system, such as the rationale behind having a requirement as well as numerical properties such as deviation indicator value that are calculated as the result of automated analysis of NFRs.

MARTE: The UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [34] is one of the recent and major efforts on modeling real-time embedded systems and their extra-functional properties. It was introduced as the successor of UML profile for Schedulability, Performance, and Time (SPT). MARTE includes concepts and semantics for UML-based description of real-time and embedded systems. The core concepts in MARTE are categorized in two parts: modeling and analysis. The intent in the analysis part is not to define new analysis techniques, but to provide a framework to annotate models with the necessary extra-functional properties and information in order to support different kinds of analyses in the real-time domain, particularly performance and schedulability. One of the main characteristics of MARTE is that it provides a common way to model both the hardware and software aspects of real-time systems. This improves the communication between developers and helps to include both hardware and software characteristics in making predication and analysis of the systems. In this thesis, we did not deal with how EFPs can be specified and represented, for which MARTE can serve as one solution. There is, however, the potential to use and gain from the capabilities of MARTE in our suggested solutions. For instance, the concepts for modeling EFPs in MARTE can be used together with our NFR profile to annotate and

include in the model of NFRs the EFPs of different *feature* elements that are considered for satisfying the NFRs.

Real-Time Specification for Java (RTSJ): Many of the operating systems and also programming languages today provide support for measuring the CPU time that a runnable entity (i.e., thread, etc.) consumes to perform its function. However, the monitoring facilities and event handling mechanisms provided by these platforms are not usually integrated with their scheduling facilities [35]. As a result, the platform cannot enforce and ensure real-time properties of threads such as their allowed execution times and deadlines. This can lead to the violation of the result of timing analyses performed before runtime. Real-Time Specification for Java (RTSJ) is an attempt to integrate scheduling of threads with the execution time monitoring facilities and enforce execution budgets on them. By monitoring the execution times of threads and comparing them with the specified timing requirements, it can detect and ensure that a thread which is about to exceed its execution time budget does not impair the execution of other tasks as expected and predicted through schedulability analysis [35].

The concept of the second layer scheduler that we introduced and implemented (in paper C) to enable detailed monitoring of real-time tasks is similar to the main idea behind RTSJ. Our approach, however, provides more types of monitoring information such as on periodicity of tasks, deadline misses, execution time overruns. It also generates extensive log information which can be analyzed for different purposes such as to detect that a task is getting closer and closer to missing its deadline in each execution and take some measures to prevent it from missing its deadline before it occurs. In addition, our solution enables to configure and use different scheduling policies in the scheduler.

Ada Ravenscar: Ravenscar profile for Ada which was introduced in the 8th International Real-Time Ada Workshop (IRTAW) [36] is a subset of the tasking model in Ada. It is defined to provide the level of predictability and determinism that is needed in safety-critical hard real-time systems by removing constructs that contain non-deterministic behavior and implementation dependancies [37, 38]. It offers features that cover the programming needs of statically defined real-time systems. Ravenscar profile enables specification of a safety-critical system in a way to be certifiable to the highest integrity levels. Such a verifiable system

is achieved by:

- providing a task concurrency model for real-time systems along with several concurrency-related checks,
- enforcement of constraints that are required to enable and preserve results of schedulability analysis and other types of static analysis techniques,
- defining bounds on memory usage and allowing only a static task set (number of tasks are fixed),
- enabling creation of small and highly efficient runtime implementations (by reducing the size and complexity of the required runtime system) [37].

These features facilitate temporal verification of Ravenscar programs and gaining confidence in its behavior. In [39], several mechanisms in Ada Ravenscar profile that contribute to the preservation of timing properties, such as WCET and period/MIAT, are introduced and discussed.

CHESS project: The CHESS project [40] which stands for Composition with Guarantees for High-integrity Embedded Software Components Assembly aims to provide model-based solutions for addressing extra-functional concerns in embedded systems while guaranteeing correctness of the system at runtime. This is done by modeling systems using a cross-domain UML profile called CHESS ML which adopts from the modeling concepts of MARTE [34] (for modeling of extra-functional properties) and SysML [41] (for high-level modeling of requirements) UML profiles, and also extends some of the concepts defined in these profiles to cover the modeling needs of telecommunication, space, railway, and to some extent automotive domains. In CHESS, Model-Driven Architecture (MDA) methodology that is recommended by Object Management Group (OMG) is used [42]. Different types of analyses are performed at different levels and throughout the transformation chain for code generation to ensure correctness of the design. The idea here is to provide a read-only Platform-Specific Model (PSM) to the user, and also make the manual editing of the code unnecessary. This is important in order to maintain design consistency. As a consequence, the results of the analyses are propagated back to the Platform-Independent Model

(PIM), so that the user can identify parts of the model that need to be modified in order to achieve the desired behavior.

CHESS project is basically defined as an extension of the ASSERT (Automated proof-based System and Software Engineering for Real-Time systems) [43] project following two main goals: to establish correctness as early as possible and to actively preserve it throughout the whole development process down to deployment and execution. Towards these goals, ASSERT adopted an MDE methodology targeting high-integrity software systems that are used in space domain and was based on a DSL. CHESS extended the approach covering other types of systems from telecommunication and railway (and partially automotive) domains. The modeling language that was used in CHESS, called CHESSML, was based on UML profiling approach consisting of concepts from UML, SysML and MARTE. Some of the contributions of this thesis have been formulated in the scope of the CHESS project (mentioned in the respective papers).

Others: In [44] by Sentilles, a framework for management of extra-functional properties in component models of embedded systems is offered. The framework enables specification and attachment of multi-valued and context-aware EFPs to architectural elements of component models. In this work, it is also highlighted and emphasized that different values for an EFP can exist which originate from different sources. Moreover, by gaining more information and knowledge about a system, refined and more accurate values for an EFP can be considered along the development of the system. As part of this work, ProCom component model is also introduced. In summary, it focuses mainly on capturing and representation of EFPs in component models of embedded systems.

The work done by Ciccozzi in [45] is a model-driven approach towards achieving preservation of properties. It introduces model-driven techniques such as full-code generation from (analyzed) system models and also a back propagation mechanism to facilitate preservation of properties. The back propagation mechanism enables to establish a feedback loop between monitoring results at runtime and system models. This way, it helps to inform designers about actual values of properties at runtime versus their expected values specified in the model. This work assumes that the platform for which code is generated has necessary mechanisms for monitoring, enforcement and basically preservation of EFPs. Moreover, it does not deal with how balance between EFPs can

be established, except its suggested round-trip support (which is common between that work and this thesis) which can be used as a means towards establishing such a balance. In that work however, the focus in the round-trip solution is on the model-based techniques to propagate back monitored information to the design models, while in this thesis, we focused on the monitoring part and the role of the platform in the round-trip chain.

REMES which is a Resource Model for Embedded Systems is introduced and applied in [46]. REMES enables modeling and reasoning about the functional and extra-functional behaviors of a system and its components. It introduces a formal language (a state-machine based behavioral language) for modeling resources. The main intention with REMES is to express resource usage and perform usage analysis. In a REMES model, resources are considered as global quantities of finite size. The analysis of resources is based on multi-priced timed automata and enables to solve, for instance, feasibility, trade-off and optimal/worst-case resource analysis problems [47].

An example of works that deal with EFPs at the hardware and chip level is [48] by Hatami which provides a method for analysis and prediction of hardware related EFPs at early design phases. It takes into account various hardware parameters such as voltage, frequency, and a few others to determine chip level EFPs such as dynamic energy for a transistor. In [49] by Sapienza, an approach is provided for making decisions based on EFPs for optimal partitioning of an embedded system into software and hardware parts.

Chapter 5

Conclusion and Future Directions

In this thesis, we addressed the important issue of preservation of EFPs in embedded systems. We started by discussing the relationship between NFRs and EFPs which is necessary in understanding how constraints over EFPs form and where they originate from. It is based on such a relationship that it then becomes possible to talk about if the EFPs of a system element or component is acceptable in the context of that system or not. In embedded systems where such EFP constraints can be determinant in success or failure of the product, ensuring that those constraints are not violated and EFPs are preserved within their constraints is crucial.

For modeling NFRs and performing trade-off analysis among them, we introduced a UML profile which enables capturing of NFRs in a generic way and automatic analysis of their trade-offs using model transformation techniques. In this step, mutual impacts and conflicts among NFRs are understood and they can then be balanced to achieve an acceptable overall satisfaction level for all the NFRs before continuing with the rest of the development process. As the next step, to build a system which respects the constraints defined over its EFPs, we introduced an approach for balancing security versus timing constraints which enables to produce a component model of the system including added security features while respecting and operating within its timing constraints. Although in the approach we only focused on security and timing prop-

erties, the suggested solution has the potential to be adapted and used for other EFPs as well.

As mentioned before, the ultimate goal in designing a software product is that the system performs as expected at runtime and when it is in operation. However, regardless of which development method is used and the amount of analysis performed, situations may still occur at runtime that lead to the violation of EFPs. To monitor for such cases for timing properties, we suggested the second layer scheduler concept. It basically adds to the execution platform the necessary mechanism for monitoring of timing events. Of course, adding monitoring features has its own overhead. In terms of the second layer scheduler, the solution we have suggested adds the monitoring capabilities without modifying the kernel of the OS, which is particularly interesting in cases where, for instance, the source code is not available, legacy systems, or when more flexibility and portability is desired. By including the monitoring features as part of the OS kernel and scheduler, the overhead can be reduced to some extent. In the context of model driven development, we also demonstrated how the collected monitoring information about EFPs can be used to propagate back to the models in order to identify inconsistencies. The models can then be modified and new code can be generated until the desired set of EFPs value at runtime is achieved. Such a round-trip support is particularly useful in optimizing the system with respect to EFPs, which deserves a separate study and investigation as a future work.

When a system is built, the next step is to test it before shipping it to customers. For testing timing properties, we also introduced a model-based testing framework which enables testing the actual behavior of the system at runtime against its desired behavior captured in the form of timed automata models. This was achieved in our approach by automatic generation and execution of test cases and use of timestamps at runtime for each state transition. As a future extension, the approach may also be well modified and used for testing of other EFPs such as memory usage, if memory constraints are also captured as part of the model. For instance, analogous to a timed automata model, if in a state machine model the information on allowed memory usage in each state is specified, the approach can easily be modified to check the memory consumption at runtime at each state transition and verify if it matches the specified constraints in the model. One point to note here with respect to testing in general is that testing and passing test cases does

not necessarily guarantee the absence of errors and bugs in a system, but serves as a means to gain more confidence in the quality of the product that is developed.

In this thesis, we also investigated the issue of the accuracy of collected monitoring information and demonstrated a technique for mitigating this issue. Accuracy plays an important role, for example, when the collected information is used to make decisions such as for runtime adaptation. If the information is not accurate (or fresh and up-to-date), it can lead to taking wrong decisions, which in turn might have drastic consequences, for instance, in safety critical systems such as a pacemaker. In the thesis, mainly single core platforms were considered and it would be interesting as another future work to investigate monitoring and testing of EFPs in multicore systems which would have their unique challenges. For instance, there could be a requirement on end-to-end response time of a set of tasks which are allocated and run on different cores. Extending the monitoring and testing methods introduced in this work to such scenarios would be another future direction and extension of this work.

As mentioned in the thesis, an EFP may well be refined and translated into one or more fine-grained EFPs at lower abstraction levels. The refinement and granularity level at which preservation of an EFP is applied can depend on several factors such as need and interest to preserve an EFP at a certain level and also monitoring feasibility at that level. For instance, if there is a composite component with a constraint on its maximum execution time, preservation of its execution time property may be enforced at the level of the composite component itself or at the level of its child components. It should, however, be noted that in the latter case, property preservation can be more restrictive, which might actually be necessary in certain systems and contexts. On the other hand, in the former case there should be a mechanism to be able to monitor the execution time of the composite component or derive it from the execution times of its child components which are feasible to monitor.

The solutions introduced in this thesis contribute to preserving EFPs at different abstraction levels and development phases, particularly in the context of model-based development. Therefore, together they can form and be part of a methodology and act as a set of design techniques for building a system with property preservation considerations and support, from the requirements analysis phase down to its deployment on the

platform, and execution. Application of such a methodology as a whole and its validation against one single system and use-case is planned as another future work. Overall, we believe that the solutions proposed in this thesis can help and serve as a set of means in building embedded systems with better quality assurance.

Bibliography

[1] Martin Glinz. On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference*, pages 21–26, New Delhi, India, October 2007.

[2] S. Heath. Embedded Systems Design. EDN series for design engineers, ISBN: 9780750655460. Newnes, 2003.

[3] Thomas Henzinger and Joseph Sifakis. The Embedded Systems Design Challenge. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2006.

[4] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. Toward Model-Based Trade-off Analysis of Non-Functional Requirements. In *38th Euromicro Conference on Software Engineering and Advanced Applications(SEAA)*, September 2012.

[5] Mehrdad Saadatmand. *Satisfying Non-Functional Requirements in Model-Driven Development of Real-Time Embedded Systems, Licentiate Thesis*. Number 150. Mälardalen University, May 2012.

[6] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems. In *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, Barcelona, Spain, October 2011.

[7] Lawrence Chung and Julio Cesar Prado Leite. Conceptual Modeling: Foundations and Applications. chapter On Non-Functional

Requirements in Software Engineering, pages 363–379. Springer-Verlag, Berlin, Heidelberg, 2009.

[8] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Non-functional requirements: From elicitation to conceptual models. In *IEEE Transactions on Software Engineering*, volume 30, pages 328–350, 2004.

[9] Bran Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20:19–25, September 2003.

[10] M. Torngren, DeJiu Chen, and I. Crnkovic. Component-based vs. model-based development: a comparison in the context of vehicular embedded systems. In *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*, pages 432 – 440, aug.-3 sept. 2005.

[11] B.W. Boehm and P.N. Papaccio. Understanding and controlling software costs. *Software Engineering, IEEE Transactions on*, 14(10):1462–1477, 1988.

[12] Barry Boehm and Victor R. Basili. Software Defect Reduction Top 10 List. *Computer*, 34(1):135–137, January 2001.

[13] G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology, May, 2002.

[14] O. Florescu, Jinfeng Huang, J. Voeten, and H. Corporaal. Strengthening Property Preservation in Concurrent Real-Time Systems. In *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, pages 106–109, 2006.

[15] Oana Florescu, Jinfeng Huang, Jeroen Voeten, and Henk Corporaal. Towards Stronger Property Preservation in Real-Time Systems Synthesis (Technical Report). `http://repository.tue.nl/710999`, 2006.

[16] S.E. Chodrow, F. Jahanian, and M. Donner. Run-time monitoring of real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 74 –83, dec 1991.

[17] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. Design of adaptive security mechanisms for real-time embedded systems. In *Proceedings of the 4th international conference on Engineering Secure Software and Systems*, ESSoS'12, pages 121–134, Eindhoven, The Netherlands, 2012. Springer-Verlag.

[18] Schrödinger's cat Experiment. `http://www.http://en.wikipedia.org/wiki/Schr%C3%B6dinger%27s_cat`, Accessed: December 2014.

[19] Mehrdad Saadatmand and Mikael Sjodin. On Combining Model-Based Analysis and Testing. In *Information Technology: New Generations (ITNG), 2013 Tenth International Conference on*, pages 260–266, Las Vegas, NV, USA, April 2013.

[20] Mehrdad Saadatmand and Sahar Tahvili. A Fuzzy Decision Support Approach for Model-Based Tradeoff Analysis of Non-Functional Requirements. In *12th International Conference on Information Technology : New Generations (ITNG)*, Las Vegas, Nevada, USA, April 2015.

[21] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. Security in Embedded Systems: Design Challenges. *ACM Trans. Embed. Comput. Syst.*, 3(3):461–491, August 2004.

[22] Mehrdad Saadatmand and Antonio Cicchetti. Mapping of State Machines to Code: Potentials and Challenges. In *The Ninth International Conference on Software Engineering Advances (ICSEA)*, pages 247–251, Nice, France, October 2014.

[23] EAST-ADL Specification. `http://www.atesst.org`, Accessed: December 2014.

[24] Jinfeng Huang, Jeroen Voeten, and Marc Geilen. Real-time Property Preservation in Concurrent Real-time Systems. In *In: Proc. of 10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, 2004.

[25] O. Florescu, Jinfeng Huang, J. Voeten, and H. Corporaal. Strengthening Property Preservation in Concurrent Real-Time Systems. In *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, pages 106–109, 2006.

[26] Mehrdad Saadatmand and Mikael Sjodin. Towards Accurate Monitoring of Extra-Functional Properties in Real-Time Embedded Systems. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, pages 338–341, Dec 2012.

[27] Antonio Vallecillo, Martin Gogolla, Loli Burgueño, Manuel Wimmer, and Lars Hamann. Formal Specification and Testing of Model Transformations. In Marco Bernardo, Vittorio Cortellessa, and Alfonso Pierantonio, editors, *Formal Methods for Model-Driven Engineering*, volume 7320 of *Lecture Notes in Computer Science*, pages 399–437. Springer Berlin Heidelberg, 2012.

[28] L. Burgueno, J. Troya, M. Wimmer, and A. Vallecillo. Static Fault Localization in Model Transformations. *Software Engineering, IEEE Transactions on*, PP(99):1–1, 2015.

[29] Mehrdad Sabetzadeh, Shiva Nejati, Marsha Chechik, and Steve Easterbrook. Reasoning about Consistency in Model Merging. In *Proceedings of 3rd Workshop on Living With Inconsistency in Software Development (Co-located with ASE2010)*, Antwerp, Belgium, September 2010.

[30] Anton Wijs and Luc Engelen. REFINER: Towards Formal Verification of Model Transformations. In JuliaM. Badger and KristinYvonne Rozier, editors, *NASA Formal Methods*, volume 8430 of *Lecture Notes in Computer Science*, pages 258–263. Springer International Publishing, 2014.

[31] Mathias Hülsbusch, Barbara König, Arend Rensink, Maria Semenyak, Christian Soltenborn, and Heike Wehrheim. Showing Full Semantics Preservation in Model Transformation - A Comparison of Techniques. In Dominique Méry and Stephan Merz, editors, *Integrated Formal Methods*, volume 6396 of *Lecture Notes in Computer Science*, pages 183–198. Springer Berlin Heidelberg, 2010.

[32] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*, volume 5 of *International Series in Software Engineering*. Springer, 1999.

[33] John Mylopoulos, Lawrence Chung, and Eric Yu. From object-oriented to goal-oriented requirements analysis. *Commun. ACM*, 42:31–37, January 1999.

[34] MARTE specification. `http://www.omgmarte.org`, Accessed: December 2014.

[35] Andy J. Wellings, Gregory Bollella, Peter C. Dibble, and David Holmes. Cost Enforcement and Deadline Monitoring in the Real-Time Specification for Java. In *7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, pages 78–85. IEEE Computer Society, 12-14 May 2004.

[36] Lars Asplund, B. Johnson, Kristina Lundqvist, and Alan Burns. Session Summary: The Ravenscar Profile and Implementation Issues. ACM Press, July 1999.

[37] Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the Ada Ravenscar Profile in high integrity systems. *Ada Lett.*, XXIV:1–74, June 2004.

[38] Kristina Lundqvist and Lars Asplund. A Ravenscar-Compliant Run-time Kernel for Safety-Critical Systems*. *Real-Time Systems*, 24:29–54, January 2003.

[39] Enrico Mezzetti, Marco Panunzio, and Tullio Vardanega. Preservation of Timing Properties with the Ada Ravenscar Profile. In Jorge Real and Tullio Vardanega, editors, *Reliable Software Technologies – Ada-Europe 2010*, volume 6106 of *Lecture Notes in Computer Science*, pages 153–166. Springer Berlin Heidelberg, 2010.

[40] CHESS Project: Composition with Guarantees for High-integrity Embedded Software Components Assembly. `http://chess-project.ning.com/`, Accessed: December 2014.

[41] OMG SysML Specifcation. `http://www.sysml.org/specs.htm`, Accessed: December 2014.

[42] Model-Driven Architecture (MDA). `http://www.omg.org/mda/`, Accessed: December 2014.

[43] ASSERT Project: Automated proof-based System and Software Engineering for Real-Time systems. `http://http://cordis.europa.eu/projects/rcn/71564_en.html/`, Accessed: December 2014.

[44] Séverine Sentilles. *Managing Extra-Functional Properties in Component-Based Development of Embedded Systems.* PhD thesis, Mälardalen University, Västerås, Sweden, June 2012.

[45] Federico Ciccozzi. *From models to code and back: A round-trip approach for model-driven engineering of embedded systems.* PhD thesis, Mälardalen University, Västerås, Sweden, January 2014.

[46] Aneta Vulgarakis. *A Resource-Aware Framework for Designing Predictable Component-Based Embedded Systems.* PhD thesis, Mälardalen University, June 2012.

[47] C. Seceleanu, A. Vulgarakis, and P. Pettersson. REMES: A Resource Model for Embedded Systems. In *Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on*, pages 84–94, 2009.

[48] Nadereh Hatami Mazinani. *Multi-level analysis of non-functional properties (PhD Thesis).* PhD thesis, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2014.

[49] Gaetana Sapienza. *Multiple Property-Based Partitioning for Embedded Applications, Licentiate Thesis.* Number 176. Mälardalen University, May 2014.

II

Included Papers

Chapter 6

Paper A: Model-Based Trade-off Analysis of Non-Functional Requirements: An Automated UML-Based Approach

Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin
In Journal of Advanced Computer Science, Vol. 3, No. 11, November, 2013.

Abstract

One common goal followed by software engineers is to deliver a product which satisfies the requirements of different stakeholders. Software requirements are generally categorized into functional and Non-Functional Requirements (NFRs). While NFRs may not be the main focus in developing some applications, there are systems and domains where the satisfaction of NFRs is even critical and one of the main factors which can determine the success or failure of the delivered product, notably in embedded systems. While the satisfaction of functional requirements can be decomposed and determined locally, NFRs are interconnected and have impacts on each other. For this reason, they cannot be considered in isolation and a careful balance and trade-off among them needs to be established. We provide a generic model-based approach to evaluate the satisfaction of NFRs taking into account their mutual impacts and dependencies. By providing indicators regarding the satisfaction level of NFRs in the system, the approach enables to compare different system design models and also identify parts of the system which can be good candidates for modification in order to achieve better satisfaction levels.

6.1 Introduction

In software engineering, there are different types of programming languages and development methods that have been introduced to develop software systems in different domains. There is one common goal that is inherent in all these different development tools and methodologies, and that is to help build a software system which satisfies the set of requirements that are defined for it. While the focus has usually been mainly on functional requirements [1, 2], inadequate attention and improper handling of Non-Functional Requirements (NFRs) has been identified as one of the important factors for failure of many project [3, 4]. In spite of this fact, NFRs are still rarely taken into account so seriously as functional requirements and not considered as first-class entities in software architecture [5]. Part of this is due to the fact that NFRs are usually defined at a high abstraction level and specified in an informal way [6, 5]. Therefore, there need to be appropriate tools and methods to incorporate them at earlier phases of development and in design models along with functional requirements. Integration of NFRs and FRs is especially important considering that having a different set of NFRs for the same FRs can result in different architectural decisions and implementations [6, 7].

While NFRs might receive less attention and degree of importance in certain systems such as desktop applications, however, they can be critical in certain domains such as in real-time and embedded systems. In these systems, there are different set of constraints and limitations on available resources and therefore, a successful design and implementation depends heavily on how it can satisfy the non-functional requirements of the system [8]. Examples of such limitations that get formulated in the form of NFRs can be limited amount of available memory, limited energy resources, and so on. Therefore, it is important to be able to evaluate different design models and alternatives with respect to the satisfaction of NFRs. For example, in one design, to fulfill security requirements, a stronger encryption algorithm might be used than another design alternative. However, using a stronger encryption algorithm, may lead to consuming more memory or processing capacity and CPU time, and this way, it impacts memory and performance requirements (if there are any defined). This brings us to the next challenge with respect to NFRs and it is that NFRs are interconnected and have dependencies and for this reason, cannot be considered in isolation. Therefore, designers should be

able to carefully identify how satisfying and fulfilling one requirements can impair the satisfaction of other NFRs in the system. Establishing and maintaining such interdependencies during the development process and the lifecycle of the product is also an important point taking into account the evolution of software architecture and introduction of new requirements or modifying existing ones. Moreover, not only NFRs can have impacts on each other, but also an NFR usually crosscuts different parts of a system. For example, achieving security in a system, requires design decisions for different parts of a system spanning from user interfaces (e.g., what a user can enter as input), database backends, communication protocols, network topology and so on.

Model-based development (MBD) is a promising approach to cope with the design complexity of systems such as those in real-time embedded domain. It helps to raise the abstraction level (and hiding unnecessary details and complexities in each viewpoint), perform analysis at earlier phases of the development and also enables automatic generation of code from the models [9]. By providing views of the system at a high abstraction level, MBD concepts can also be used to model NFRs, which as stated are usually defined at a high abstraction level, and incorporate them with other parts of the system. Analysis of NFRs can then be performed on the model and also the model of NFRs can be maintained as a development artifact.

In this paper we introduce a UML profile [10, 11] for modeling NFRs in a generic way and regardless of their type (i.e., performance, availability, and so on), to enable performing trade-off analysis on them. By including important information about each requirement in the model, such as its priority and also its relationships to other requirements and functional parts of the system, the dependencies and impacts of NFRs are analyzed to provide system designers with information about how good a system design is in terms of the satisfaction of its NFRs. It also helps to identify parts of the system in which violations and deviations have occurred that deserve more attention. Based on this information, system designers can also compare different design models and alternatives. Another approach for modeling NFRs could be to define a Domain-Specific Language (DSL) from scratch (i.e., non-UML based approaches), however, using UML and its profiling mechanism to extend it and define new modeling semantics has some advantages. Introducing a DSL requires extra efforts on training the developers, while most developers may already be familiar or even using UML. For this rea-

son, it can also serve as a unifying factor between different development teams (e.g., to communicate design decisions). Moreover, there is a big variety of different UML tools which are already available and can be used off-the-shelf. Also, integrating NFRs with functional parts of the systems will be more straightforward, such as when there already exist UML models of the system and the model of NFRs based on our introduced profile can be constructed and integrated with them (e.g., legacy systems). A comparison of advantages and disadvantages of using DSLs and UML profiles for defining new modeling semantics are discussed in detail in [10, 12].

Using the suggested profile for modeling NFRs, not only NFRs can be modeled and integrated with already existing functional models of the system, but it is also intended to be used for constructing the NFR model at the beginning of the development process and to perform analysis of their trade-offs, especially when enough information about their impacts and dependencies are available. The model may then gradually grow, be integrated with functional parts as they get designed, and automatic analysis of NFRs can be done iteratively when any changes that can affect NFRs are made.

The remainder of the paper is structured as follows. In Section 6.2, NFRs and their characteristics are introduced and the challenges related to NFRs during the development process are identified and discussed. In Section 6.3, we formulate and summarize the characteristics that different solutions for managing the trade-offs of NFRs should be able to provide to cope with the identified challenges. Section 6.4 describes in the detail the suggested UML profile and its modeling semantics including the rules and formulas that are defined for performing trade-off analysis on the models of NFRs. An application of the profile and how analysis is performed on NFRs is provided in Section 6.5 using a selected part of the NFR model of a mobile phone. Discussion of different aspects of the proposed approach is offered in Section 6.6. In Section 6.7, related works are investigated and finally in Section 6.8, a summary of the work is provided and conclusions are drawn.

6.2 Non-Functional Requirements

6.2.1 Definitions

A requirement is basically an expression of a need [13] and in developing software systems, there can be different stakeholders with their own specific requirements [14]. Some requirements, such as those related to the user interface, may originate from the customer or end-user side, while some other requirements may be due to the selection of a particular development process (e.g., agile or model-based development). Also, there are different standards and regulations that may need to be followed in the development of a software system which bring along additional sets of requirements. Examples of such standards could be different safety standards that a safety-critical system should conform to, for instance, in avionics, automotive, and medical systems. In systems engineering, requirements are usually categorized as *functional* and *non-functional* [13]. Simply stated, functional requirements state what a system should do and are sometimes identified as capabilities of a software product [14], whereas non-functional requirements define how the system should perform or as mentioned in [15] a non-functional requirement is "an attribute or a constraint on a system". A list of different definitions for non-functional requirements are collected in [16]. An example for functional requirements could be that a system should be able to read input from a text file. A non-functional requirement could be that the process of reading the input file should not take more than 10 milliseconds; this requirement is basically an expression of a performance need in the system.

The IEEE standards, 610.12-1990 and ISO/IEC/IEEE 24765:2010(E) [17, 18] provide the following definitions for requirement, and functional and non-functional requirements (quoted):

- Requirement:

    1. a condition or capability needed by a user to solve a problem or achieve an objective.
    2. a condition or capability that must be met or possessed by a system, system component, product, or service to satisfy an agreement, standard, specification, or other formally imposed documents.

3. a documented representation of a condition or capability as in (1) or (2).
4. a condition or capability that must be met or possessed by a system, product, service, result, or component to satisfy a contract, standard, specification, or other formally imposed document.

- Functional Requirement:

1. a statement that identifies what a product or process must accomplish to produce required behavior and/or results.
2. a requirement that specifies a function that a system or system component must be able to perform.

- Non-Functional Requirement: a software requirement that describes not what the software will do but how the software will do it (i.e., design constraints). Examples: software performance requirements, software external interface requirements, software design constraints, and software quality attributes. Non-functional requirements are sometimes difficult to test, so they are usually evaluated subjectively.

Moreover, a requirement can be refined (into smaller, more detailed and fine-grained ones) and this way a hierarchy of requirements can be created. The term *derived requirement* is also offered by the IEEE standards, 610.12-1990 and ISO/IEC/IEEE 24765:2010(E), which is defined as:

- Derived Requirement:

1. a lower-level requirement that is determined to be necessary for a top-level requirement to be met.
2. a requirement that is not explicitly stated in customer requirements, but is inferred from contextual requirements (such as applicable standards, laws, policies, common practices, and management decisions) or from requirements needed to specify a product or service component.

In this context, the term extra-functional is also used at times as an equivalent of non-functional to change the focus and take away and

replace the negative aspect that is inherent in 'non'. On the other hand, there is the concept of non-functional/extra-functional property (NFP/EFP), which is often confused with NFRs. As a type of requirements, NFRs are also expression of a need which are generally stated in an informal way, while a property is a statement that can be asserted formally, and therefore, it can be analyzed and proven. An example of extra-functional properties could be the worst-case execution time of a component in a system which may be calculated statically or measured. Therefore, saying that "the worst-case execution of component A is 5ms" or that "the execution time of component A never exceed 10ms" are actually expression of properties. On the other hand, "the execution time of component A should never exceed 10ms" is a non-functional requirement and an expression of a need. The key point here is that a property per se does not tell us much about its validity, and it is only when it is considered along with its related requirement(s) that we can determine whether it is acceptable and good for a specific design or not. In other words, if we know that the worst-case execution time of a component is 5ms, we cannot determine whether it can be considered a good value or not, unless we check it against the requirements. While for one system this value of 5ms could be acceptable, for other systems this may be considered as problematic and lead to the violation of requirements. Considering such a relationship between an NFR and an extra-functional property, to satisfy an NFR, its related extra-functional properties should have valid values. For example, to satisfy performance and schedulability requirements in a real-time system, execution and response time values (among others) should remain within a valid range. Understanding the differences between these two terms is important in some works (such as this paper), while in other contexts, their differences can be ignored and using these two terms as equivalents can be safe. In [19], NFP is used instead of NFR when talking about the final product implying that the requirement has been concretized and become an actual property of it.

6.2.2 Characteristics and Challenges

Addressing NFRs in the development of a software product is a challenging tasks. Aside from the fact that often times NFRs are expressed in a natural language and informally, they have some characteristics that makes their consideration in the development process complicated. In

contrast to FRs which are typically realized locally and implemented one by one and step by step in an incremental manner while the software product is buing built, NFRs do not follow such a pattern. In this respect, NFRs can be considered as specification of global constraints on the software product, such as security, performance, availability and so on [5] which can crosscut different parts of a system. Also in satisfying NFRs, the dependencies among them should not be neglected, as satisfying one NFR can affect and impair the satisfaction of other NFRs in the system. Therefore, performing trade-off analysis to establish balance among NFRs and identify such mutual impacts is necessary.

There are also other issues that contribute to the complexity of managing NFRs in the development process. For example, organizational structures of companies and the way they are divided into different development departments and sub-departments usually fit functional requirements; as these requirements can be (more easily) implemented in separation from each other and then integrated to satisfy a parent requirement (considering a hierarchy of requirements consisting of refinements of each) [20, 21]. On the other hand, a non-functional requirement such as security, availability, or user-friendliness crosscuts different parts of the system and requires a more holistic view and a top-down approach [21]. Another problem which is mostly observed in large organizations is that different teams may have different interpretations of an NFR, or vice versa, refer to one NFR using different terms [20]. Therefore, a coherent way of representing and defining NFRs, and also establishing and maintaining traceability links among them can be helpful to mitigate such problems. Issues related to traceability between NFRs can also occur easily during the development process [22]. For example, code tweaks that one development team may do to improve performance, which may affect security or memory consumption, can become hidden and lost to other teams.

Considering that NFRs are usually specified in an informal and abstract way [5, 19], providing a more formal approach using model-based development which enables to raise the abstraction level can help with the treatment of NFRs during the development process. Dealing explicitly with NFRs and incorporating them in different phases of development becomes more important especially considering the increasing number of systems in which NFRs are critical such as real-time embedded systems. Moreover, an explicit treatment of NFRs facilitates the predictability of the system in terms of the quality properties of the

final product in a more reliable and reasonable way [19].

Sometimes the approaches for the explicit treatment of NFRs are categorized into two groups: product-oriented and process-oriented [23]. The former approaches try to formalize NFRs in the final product in order to perform evaluation on the degree to which requirements are met. In the latter approaches, NFRs are considered along with functional requirements to justify design decisions and guide and rationalize the development process and construction of the software in terms of its NFRs [23, 19].

6.3 Addressing the Challenges of NFRs

Considering the nature of NFRs and to cope with the challenges that have been discussed so far in managing and treatment of them in the development process, we formulate here the key features that are required in order to model NFRs and enable performing trade-off analysis among them to evaluate a system design with respect to the satisfaction of its NFRs.

Traceability of design decisions related to an NFR: An NFR can crosscut different parts of a system and there needs to be a mechanism to identify the parts that contribute to its satisfaction. Establishing such a relationship is especially important after performing trade-off analysis in order to identify which parts of the system should be replaced or modified in order to improve the satisfaction of an NFR. On the other hand, in maintaining a system, it is important to find out which requirement(s) a specific part of a system is related to and as a result of which requirement(s) that part has been implemented. Such information can easily become lost in complex systems and also as the system ages.

Traceability between an NFR and its refinements: as mentioned before, during the whole development process, high level NFRs get refined into more fine-grained ones which leads to the formation of a hierarchy and tree-structure of NFRs and parent-child relationships among them. Therefore, in order to evaluate the satisfaction of one NFR in the system, it is necessary to keep track of its refinements and the children requirements originated from it at lower levels of requirements hierarchy. The evaluation of an NFR, is thus, performed recursively by evaluating to what degree its refinements have been satisfied. As an example of such refinements, we can name security as an NFR which can then be refined

into lower level and more concrete requirements such as encryption of data and access control mechanisms.

Impact of an NFR on other NFRs: Due to the impacts that NFRs have on each other and the interdependencies among them, an NFR cannot be considered in isolation in a system in order to satisfy and achieve it. System designers should be able to identify the impacts that a system feature and design decision that is made to satisfy one NFR can have on other NFRs. Examples of such impacts can be more tangible in embedded systems. For instance, performing heavy computations by an encryption component in an embedded system can lead to consuming more battery. Therefore, it is important to be able to identify and include such impacts and side effects as part of the system design models.

Priority of an NFR: In a system, different NFRs can have different levels of importance. It is necessary to know the importance of each NFR to be able to compare them and resolve conflicts among them (reduce the impact of one NFR in favor of another) to improve the overall satisfaction of NFRs. Considering priorities for NFRs is also important to capture the preferences of customers. Similarly, priorities can also be considered for different features implemented to satisfy an NFR.

Satisfaction level of an NFR: To enable comparison of a system design against the specifications of the system and customer requirements and also to compare different design alternatives, it is needed to evaluate, specify and represent the satisfaction degree/level of an NFR in the system. The end goal is that system designers should be able to get an idea to what extent each NFR is satisfied and how good a system design is in terms of the satisfaction of its NFRs. After analyzing the dependencies and impacts of NFRs and determining their satisfaction levels, as the next step, it can be judged whether the satisfaction level of an NFR is acceptable or not. This phase can probably be done by checking and consulting with the stakeholders, if needed.

Coherent terms for NFRs: It was discussed that especially in large organizations, it can happen that different departments and development teams may have their own interpretations for each NFR or use different terms to refer to an NFR. By providing a coherent and consistent representation and notation for NFRs and also establishing traceability links for them (to other NFRs as well as to design elements implementing each), it becomes possible to mitigate such inconsistency problems. This problem can be very subtle and easily remain unnoticed [20].

Coherent measurements of NFRs: To enable the comparison of dif-

ferent NFRs and performing trade-off analysis among them, specification of the satisfaction level and impact values of NFRs should follow a coherent representation. This means that the criteria or metrics that are used should be such that to allow pair-wise comparison of NFRs (e.g. using the same types, scales and units, or a convertible format).

6.4 Suggested Approach

This section is devoted to the illustration of the proposed UML profile enabling the modeling of NFRs and hence their trade-off analysis. Therefore, in the following we first introduce some basic concepts about UML profiles that underpin the technicalities of our proposal.

6.4.1 UML Profiles

As mentioned before in this article, thanks to MBD the early evaluation of quality attributes can dramatically save development time and verification and validation costs. The underlying assumption is that the adopted modeling means are capable of carrying by enough details to perform reliable evaluations.

Historically there have been two different ways of addressing language expressiveness limitations, either UML profiling or designing a new DSL from scratch. The former exploits a possibility given by the UML to extend itself, while the latter prescribes building a new modeling language specifically tailored to the domain taken into account. Both approaches have their own advantages and drawbacks [10, 12], the discussion of which goes beyond the scope of this article. However, it is worth noting that, especially in industrial settings, UML profiles are typically preferred due to multiple (practical) reasons: UML is a *de facto* standard for modeling industrial software systems, therefore it is expectable the existence of a "legacy" including models, tools, skilled personnel, and so forth; UML profiles, as will be discussed below, are still UML models, thus compatible with other models, and even more important, with existing UML tool formats. We opted for a UML profile as the means for supporting the modeling of NFRs details to enable their trade-off analysis. Nonetheless, there are no limitations from the expressiveness perspective preventing the realization of the same kind of modeling support by adopting the DSL solution.

UML has been conceived from the beginning as a general purpose language, therefore it does not contain any domain-specific concept. On the contrary, it allows to model any kind of reality abstraction thanks to its expressiveness. Preservation of generality comes at the cost of lack of precise semantics and ambiguities that can be fixed by exploiting UML profiles. It is worth mentioning that the UML language can be refined by adding, removing, and changing the available concepts, thus creating a new DSL [10]. However, models created by means of such a new language would be not compatible with other UML models and tools. Consequently, UML has been equipped with modeling concepts able to specialize the language itself, i.e. profiles [11].

A *Profile* is a specialization of an existing UML modeling concept; for instance, profiles can be created not only for classes and relationships, but also for states in Activity Diagrams, actors in Use Cases, messages in Sequence Diagrams, and so forth. *Interface* is a famous example of profile for *Class*. When exploited, the profile allows users to recognize that what they have in their hands is not a regular UML Class but an Interface, and act appropriately (that is, give a precise semantic to the kind of object taken into account). Profiles can be also enriched by adding new attributes and properties, called *Tagged Values* (simply referred to as properties in this work). In this way, information can be provided as specifically pertaining to the introduced profile. In the next section, we show how this powerful concept can be used to store NFRs information in order to enable trade-off analysis at the design level of abstraction.

6.4.2 NFR Profile

Based on the challenges identified in Section 6.3, we have created a UML profile to define NFRs as model elements and include necessary information (in the form of properties of model elements and different relationships among them) to enable performing trade-off analysis and evaluating the design with the respect to the satisfaction of NFRs. The structure of the defined profile is depicted in Figure 6.1.

The profile consists of several key stereotypes and properties that are described as follows:

`System:` In the hierarchy of NFRs, the root node will represent the system itself which can have several different NFRs represented in the model at the lower levels of the hierarchy as children model elements.

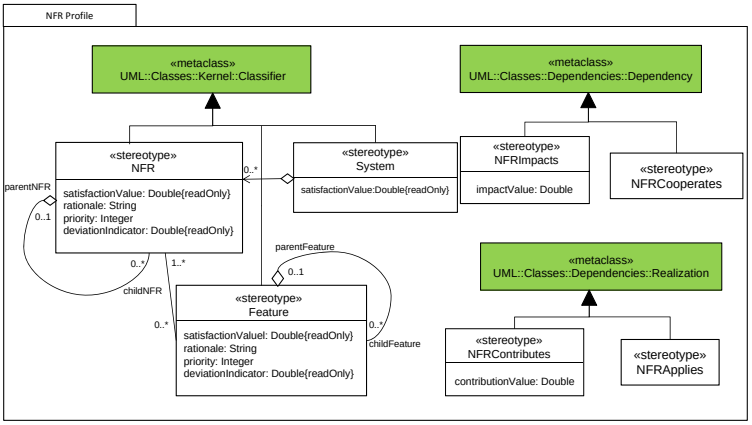

Figure 6.1: NFR Profile

The System stereotype is used to annotate this root model element as the system. The system is also considered as the context of the analysis.

`SatisfactionValue:` This property is used to represent the satisfaction degree of the model element it belongs to and to what extent it has been fulfilled. As can be seen in the Figure 6.1, several stereotypes have this property. In case of the System stereotype, the value of this property shows the total calculated satisfaction value for the system (described later). This value is calculated and set by the analysis engine and the users cannot set it.

`NFR:` NFRs in the system are stereotyped and annotated with this defined stereotype. Since NFRs can have other NFRs as refinements and thus as children nodes, an associtation relationship to itself (reflexive aggretation) has been defined for it.

`Feature:` A feature in the system that is defined to satisfy an NFR is identified by using this stereotype. It is basically the equivalent of *Operationalization* concept in NFR framework and Softgoal Interdependency Graph (SIG) [24] or *tactics* as used in [1] (described later in the work).

`NFRContributes:` This stereotype is used to indicate that an NFR or Feature contributes directly to the satisfaction of another one. It has a property called contributionValue that specifies the degree of this contribution.

`NFRImpacts:` This is similar to NFRContributes stereotype but is used to include the impact of a model element on other NFRs in the system in a quantitative manner. In other words, this stereotype is defined to capture the side effects of features and NFRs. ImpactValue property of this stereotype shows the degree of the impact. A positive value for the ImpactValue implies a positive side effect, and a negative one implies a negative side effect accordingly.

`NFRCooperates:` When there are more than one element that are defined to work together in satisfying an NFR, this stereotype is used to annotate and show such a cooperation relationship between them. This concept is similar to the AND relation in the NFR framework and SIG (another reason to provide this stereotype to explicitly specify such cooperation relationships is to help with the extensibility of the suggested approach in future to include different design alternatives in the form of OR relationships in the same design model, when needed).

`NFRApplies:` This stereotype is defined to enable the possibility to relate the NFR model to functional model elements (e.g. an NFR that applies to a component). For instance, if there is already a UML model of the system available (e.g., a class diagram), with this stereotype it can be specified to which part of that model an NFR or Feature applies and is related to.

`Rationale:` The rationale behind having an NFR or Feature and any other description about it can be captured and specified in this property. Both NFR and Feature stereotypes have this property.

`Priority:` This property which exists in both NFR and Feature stereotypes captures the preferences of customers (and also developers priorities when relevant and applicable) and their priorities in terms of the relative importance of NFRs and Features.

`DeviationIndicator:` By taking into account the priority and the satisfaction value of an NFR or Feature, a value for this property is calculated (as will be described soon) and provided which indicates to the designer the importance and magnitude of how much the satisfaction of an NFR or Feature has deviated or been violated. The deviation indicator value basically shows and helps to identify which parts of the system have deviated more from the specification (i.e., from being fully satisfied) and may need to be modified to achieve a better satisfaction level. This value is also calculated and set by the analysis engine and the users cannot set it. While the satisfaction value does not reflect user preferences and priorities, the deviation indicator value identifies

to the designers which parts need to be considered first with respect to the preferences and priorities of the customers. This is especially helpful and beneficial for identifying such parts in complex systems.

To use the profile and perform calculations, there are several rules that are defined on model elements and their relationships and how to set and calculate values for different properties:

- The priority for an element can be set to one of the following values: 1 (very low), 2 (low), 3 (medium), 4 (high), 5 (very high).
- The satisfaction value for each leaf node is always considered to be 1.
- The contribution value of the NFRContributes link connecting a child node to its parent can be set as a positive value between 0 and 1, but the sum of the contribution values of the links connecting children nodes (refinement/lower level elements) to their parent should always be less or equal to 1.
- The contribution of a child node to its parent is calculated by multiplying the satisfactionValue of the child node by the contributionValue of the NFRContributes link that connects it to the parent.
- For NFRImpacts links, the allowed range of values is between -1 and 1. A negative value on the NFRImpacts relationship shows the negative impact of the source element on the target.
- The total impact value of other nodes on a node (denoted as I) is calculated as follows: if the sum of all impact values is positive and not greater than 1, then the total impact value will be this sum, however, if the sum is greater than 1, then the total impact value on the node will be 1. On the other hand, if the sum of all impact values is negative and not less than -1, then the total impact value will be this sum, however, if the sum is less than -1 (e.g., -1.5 or -2), then the total impact value on the node will be set as -1. Note that the value of I in this calculation will always be between -1 and 1. This is summarized by the following formula, considering that i_j is the impact value of another node on the node for which we want to calculate the total impact value:

$$I = \begin{cases} Min(\sum i_j, 1) & if \sum i_j >= 0 \\ Max(\sum i_j, -1) & if \sum i_j < 0 \end{cases} \quad (6.1)$$

- To calculate the satisfactionValue of a node, first the total contributions from all of its children nodes are calculated, and then the total impact value is also taken into account. If s_k is the satisfaction value for each child node of a node, l_k is the value on the link that connects the child node k to its parent node (NFRContributes relationship), and I is the total impact value, the satisfaction value of the parent node is calculated as:

$$S = \begin{cases} Min((\sum s_k * l_k) + I, 1) & if (\sum s_k * l_k) + I >= 0 \\ 0 & if (\sum s_k * l_k) + I < 0 \end{cases} \quad (6.2)$$

  Considering the above rules and formulas, the satisfaction value of a node will be in the range of 0 and 1. To perform these calculations, nodes are navigated and traversed starting from leaf nodes (considering that the satisfaction of leaf nodes is 1) and values are calculated using the above formulas upwards toward the top element which is the system.

- The DeviationIndicator is calculated after the calculation of satisfaction value using the following formula:

$$\begin{aligned} &DeviationIndicator \\ &Priority - Priority * SatisfactionValue \end{aligned} = \quad (6.3)$$

  Based on this calculation and considering that the SatisfactionValue is always between 0 and 1 and priority is an integer value between 1 and 5, the value of DeviationIndicator will be in the range of $[0, 5]$. The perfect situation is when the DeviationIndicator value is 0, and the more this value increases the more is the deviation from the desired design, and thus, it indicates a bigger and more severe problem.

6.4.3 Implementation

The profile and its concepts that were described are implemented using MDT Papyrus [25] in Eclipse [26]. To navigate and transform a model that is annotated with our suggested UML profile, a model-to model (M2M) transformation is also developed using QVT Operational language (QVT-O)[27]. The transformation incorporates all the rules for performing calculations and reads as input a UML model annotated with our profile, traverses the nodes and calculates satisfaction and deviation values and writes the results back in the same model. This means that we use an *in-place transformation* (i.e. input and output models are the same) to perform the analysis on the model. A recursive algorithm is executed as part of the transformation which starts from the System node. To calculate the total satisfaction value of the system, it first retrieves the children NFRs of the system node and recursively performs calculations on each of them based on the defined formulas and rules; meaning that all the children of that node are again retrieved and this continues until it reaches a leaf node whose satisfaction value will be considered 1. In other words, for each node, first all the links that are stereotyped with NFRContributes or NFRImpacts are retrieved. A node which does not have such a link is then considered a leaf node, while for other nodes, the source node of the link is retrieved (which will be another node); hence the recursion.

6.5 Usage Example

In this section we show the applicability of the approach and how it is used for modeling NFRs and performing analysis on them to evaluate the *satisfiability* (by this term we mean the ability to satisfy the NFRs) of a model and also compare it with other design alternatives. Figure 6.2 shows NFRs that are defined for part of a mobile phone system using our profile in Papyrus. One NFR is defined for the quality of the pictures that are taken by the mobile phone. This NFR which can for example state that the quality of the picture should not be below a certain level is represented in the model simply as `Camera Picture Quality`. Similarly, another NFR is defined to represent the requirement on efficient use of battery and energy consumption in the mobile phone, denoted as `Battery Life` NFR in the model. To satisfy the `Camera Picture Quality` NFR, the possibility to use flash for taking pictures,

and also a specific type of lens have been considered (modeled as `Flash` and `Lens` features). To satisfy and achieve the requirement related to the battery life of the mobile phone, automatic adjustment of brightness level and also automatic standby mode (e.g., when the phone is in idle state) have been designed.

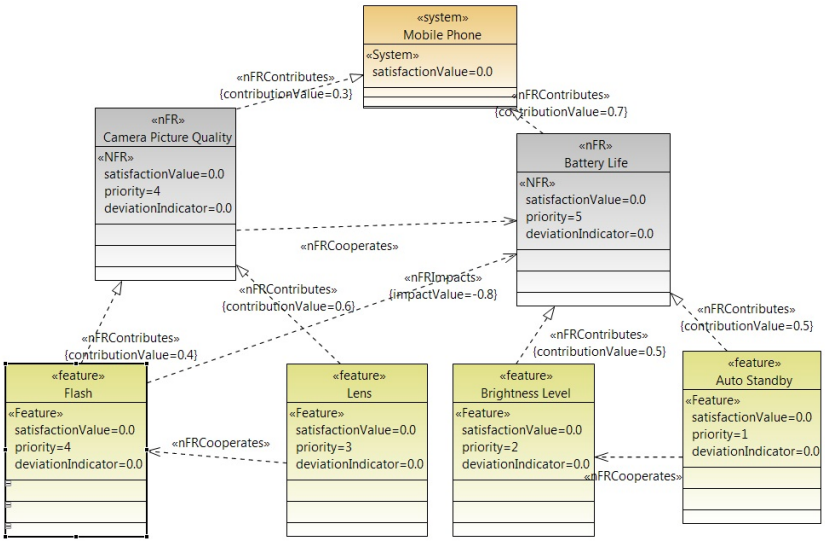

Figure 6.2: NFRs for the mobile phone system (before analysis)

NFRContributes stereotype is used to annotate the relationship between each feature and the NFR to which it contributes. Moreover, the dependencies and impacts of NFRs and features on each are modeled using the NFRImpacts stereotype, which as mentioned before can have positive or negative values. Since the use of the flash has a negative impact on the battery level and consumes energy, the value of the NFRImpacts relationship between the `Flash` feature and `Battery Life` NFR, which shows the magnitude of this impact is specified as a negative number. Importance of different NFRs and features for the customer and his/her preferences are captured by the priority property. The initial values of `satisfactionValue` and `deviationIndicator` properties are zero indicating that no calculation has been done on the model yet.

To analyze the model and perform calculations based on the formulas

defined for the profile (which are implemented as part of the transformation code), the model is fed as input to the transformation. The calculations are done using the recursive algorithm that was described before. In case of the mobile phone example here, the `Flash` and `Lens` features will be identified as leaf nodes and thus their satisfaction values are set to 1. The satisfaction value of `Camera Picture Quality` is calculated as the satisfaction value of `Flash` multiplied by the contribution value of the NFRContributes links that connects it to the `Camera Picture Quality` plus the same multiplication done on the `Lens` and its NFRContributes link: $1 * 0.4 + 1 * 0.6 = 1$.

The same calculations are done to obtain the satisfaction value for `Battery Life`, however, in this case there is an impact from the use of the `Flash` feature. Therefore its satisfaction value is calculated as: $1 * 0.5 + 1 * 0.5 - 0.8 = 0.20$. Figure 6.3 shows the analyzed model of the system. The discrepancy that is observed in the calculated satisfaction value for `Battery Life`, that is 0.1999... instead of being 0.20, is due to the OCL implementation of real numbers that are used in QVT.

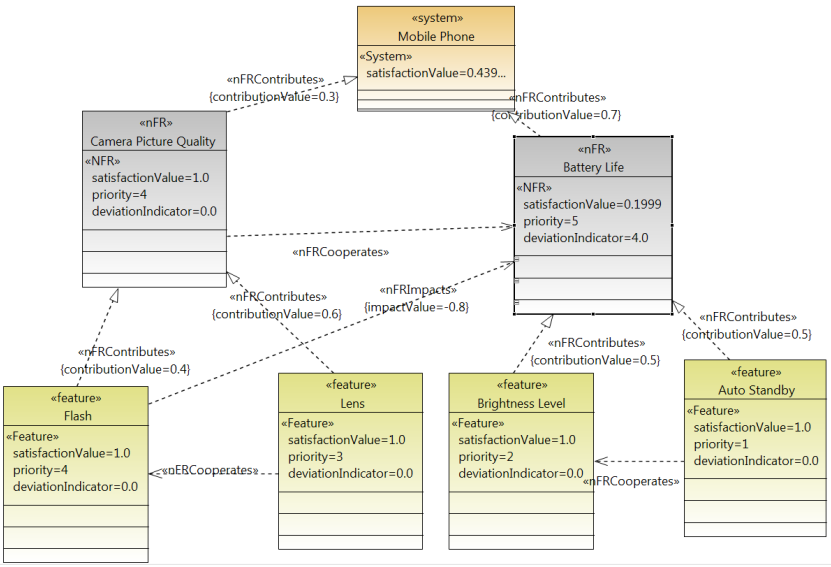

Figure 6.3: Analyzed model of the system

The total satisfaction value which is calculated for the `System` node is therefore: $1 * 0.3 + 0.2 * 0.7 = 0.44$. Having the satisfaction values of NFRs and features in the model, the deviation indicator values can now be calculated using Formula 6.3. The deviation indicator value for the leaf nodes will always result in 0 as their satisfaction values are set to 1. For the `Camera Picture Quality` whose satisfaction value is also 1 the deviation indicator value will be $4 - 4 * 1.0 = 0$ as well. However for `Battery Life`, this value will be $5 - 5 * 0.2 = 4$. This high deviation indicator value (compared to other parts) in the model shows the designers that this part of the model requires a more careful attention. Such parts could be good candidates for modification and refactoring in order to improve the satisfiability of the system. Considering the deviation indicator value of the `Battery Life`, and by investigating the elements that have impacts on it (here only the `Flash` feature), it can imply that the type of the flash that is selected to be used in this system and model of mobile phone is not good enough in terms of energy consumption and a more energy efficient flash can be used to improve the satisfiability of the system. In this rather simple example, we could have also guessed the issue with the type of flash that is used, based on the magnitude of the impact that it has on the `Battery Life` in the system; especially that it is the only impact on `Battery Life` (there could, for example, exist other NFRs and features with positive or negative impacts on it as well). However, in more complex systems with lots of dependencies and mutual impacts and taking into account the priorities of the customers, identifying the parts that have quite major (negative) effects on the satisfiability of the system and thus are of utmost importance to be re-considered could be a real staggering challenge.

Figure 6.4 shows the model of the system but without the `Flash` feature, which could represent a different model and family of mobile phones. By performing analysis on this model, the total satisfaction value of 0.88 is calculated for this design of the mobile phone; versus 0.44 in the previous model which included the flash. On the other hand, removing the flash, as can be seen from the analyzed model, has led to some deviaiton (1.6) in the `Camera Picture Quality` NFR.

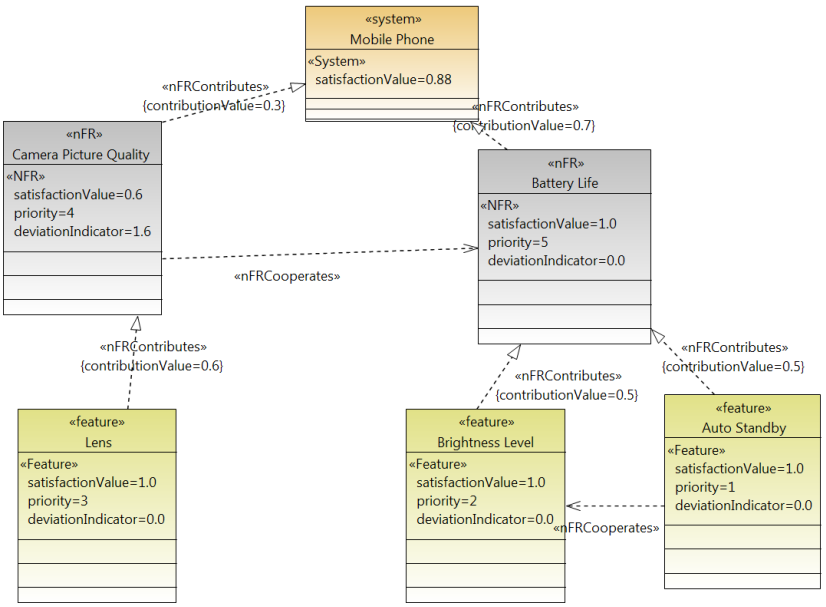

Figure 6.4: Analyzed model of the system without the Flash feature

6.6 Discussion

As was demonstrated in the previous section, our suggested approach enables designers to compare different design alternatives with respect to the satisfaction of NFRs by taking into account interdependencies and impacts of NFRs as well as the features that are designed to satisfy and fulfill each. This can help the designers in making decisions when building a system. Moreover, the approach provides for several other interesting features which we discuss here. Considering that we can now evaluate the satisfiability of a system design and compare different design alternatives, it becomes also possible to use the suggested approach in optimization of design models with respect to their NFRs. For example, in the mobile phone system, if there is a kind of repository of NFRs and features to choose from, it becomes possible to perform a series of analysis in order to find a set of NFRs and features which lead to the highest possible satisfaction value for the `Battery Life` NFR, for

instance (or even the whole system). However, this may not be as simple as it sounds due to the famous state-space explosion problem [28] that can happen in bigger and more complex systems.

Another use of the suggested approach could be to support runtime adaptation and building re-configurable systems. For instance, in case of power consumption in the mobile phone system example, if at runtime it is detected that the battery level has fallen beyond a certain level, an analysis can be performed using the introduced approach to find alternatives and identify a set of features that incur minimum impact on the battery consumption and then replace active components in the system accordingly to make the system go into a power-saving mode. To reach such an adaptive behavior, the analysis part may or may not be done at runtime. In other words, different design alternatives may have been considered and analyzed offline, and then based on desired Quality-of-Service (QoS) levels at runtime, a different architecture may be adopted to re-configure the system (similar to design diversity techniques [29]).

To enable performing a quantitative type of analysis which in turn gives designers the possibility to more carefully evaluate a model as well as different parts of it and also compare it with other alternatives, it was assumed that the designers can specify the necessary values (in this case, contribution and impact values). There are some methods that help with providing such quantitative information (as will be discussed in the related work section), however, as also mentioned in [18, 1], deciding on these values is usually a subjective task, whose precisions can be improved and increased through the use of the different methods. On the other hand, our suggested approach is deemed more suitable in Component-Based Design (CBD) of systems [30], where a system is built by composing and as an assembly of already existing components, and thus, more information and knowledge about the characteristics and behaviors of the different constituting features of the system are available. Such information could be memory usage, execution time, energy consumption and similar properties which help designers to specify more accurate quantified values in the NFR model. For example, if there is an NFR which specifies that the actual throughput should not be lower than a certain level, however, a protocol is used to satisfy security requirements which is known to double the amount of transmitted packets due to the transmission of security related information, then the impact of this feature on the bandwidth NFR can be specified as -0.5 indicating that it consumes half of the bandwidth to pass the additional informa-

tion. Also, in this work we assumed that the satisfaction values of leaf nodes are always 1, meaning that they are/will be fully implemented. If, for any specific reasons, the system needs to be analyzed using not-fully implemented features, then this assumption and rule can be relaxed to also enable specifying values between 0 and 1 for leaf nodes.

6.7 Related Work

One of the fundamental works in the field of non-functional requirements is the NFR Framework which is proposed in [24]. It is a process- and goal-oriented approach which makes use of Softgoal Interdependency Graphs (SIG) to represent NFRs. In this approach NFRs are refined into other fine-grained NFRs and also entities that function to satisfy NFRs which are termed as *Operationalization*. The dependencies and contributions of NFRs are specified using *make*, *hurt*, *help*, *break* and *undetermined* relationship types. Besides NFR softgoals, and operationalizating softgoals, NFR framework also introduces *claim* softgoals which convey the rationale and argument for or against a design decision. In addition, it provides notations to mark critical NFRs in the graph as a way to specify priorities on NFRs, and also an evaluation procedure to determine the satisfaction and conflicts of NFRs. NFR Framework is basically a qualitative approach for evaluation of NFRs and their impacts and dependencies, which although is quite useful for capturing NFRs and their relationships, but evaluating the satisfaction of NFRs is not easy [1] and hard to automate. Moreover, the criticality concept in NFR framework may be more suitable for developers and does not convey enough information for prioritization of NFRs particularly from the customer's perspective and also for performing trade-off analysis. In [1], QSIG is introduced which is basically a quantified version of SIG. It enables to perform quantitative evaluation of impacts and trade-offs among NFRs. Our work is inspired by the QSIG approach in the sense that the structure of the UML model that is built is similar to that of QSIG, and as in QSIG, we also defined a set of rules for calculations of different values, although our rules are different to be more suitable for complex systems where, for example, an NFR may be impacted by several different NFRs. We also introduced the concept of deviation indicator which is especially useful in such situations in complex systems to identify problematic parts of them. Also in QSIG, there is no explicit

concept of priority for capturing customers preferences and the impact of one NFR on another is assumed to also convey priorities. This is also another fundamental difference as we believe the concept of impact and priority should be separate, considering that the impact of an NFR on another one should be evaluated per se, while the customer priority for that the latter NFR can show the designers the meaning and importance of such impact especially when the deviation indicator is also taken into account. Moreover, in [1], no automation mechanism for the calculations is discussed, and while the QSIG graph is used to make decisions as a separate document with no connection to the functional parts, the integration of NFRs with functional parts are actually done at the code level through the notation of classpects [31] and irrespective of the constructed graph. In contrast, we enable the integration of NFRs with functional parts at the model level and the analysis of NFRs is also done automatically. In the case that the code is to be generated from the models later on, the concept of classpects could be considered as an interesting method for the integration of NFRs in the implementation code, if the code is based on an aspect-oriented and object-oriented language (as classpects is basically a concept unifying classes and aspects for such languages). [2] introduces FQQSIG which is a fuzzy quantitative and qualitative softgoal interdependency graph representation for analysis of NFRs in trustworthy software, however it offers no solution for the integration of NFRs with other parts of the system. On the other hand, although both QSIG and FQQSIG approaches provide solutions for evaluating different design alternatives, one subtle but important difference that our suggested modeling solution has is that the main idea in our work is to maintain the NFR model throughout the development process and perform analysis whenever and as many times as needed, such as when a new requirement is added or an existing one is modified, as well as when a new design model is created which should be evaluated and compared with the old one in terms of the satisfiability of its NFRs. Such an approach and vision on NFRs is important in managing NFRs throughout the development lifecycle, particularly, considering all the related challenges of NFRs which we discussed in this paper.

Another important work in the area of evaluation of different systems designs and architectures, and identifying the trade-offs of competing quality attributes is the Architecture Trade-off Analysis Model (ATAM) [7]. It is a spiral model of design and risk mitigation process that helps to find the dependencies among quality attributes which are

referred in ATAM as trade-off points. These trade-off points are considered to be caused and derived from architectural elements that are important for and affected by multiple attributes. This method is helpful at the beginning of development process to evaluate different designs and architectures and select one, however, it does not help that much to address the challenges of NFRs that we discussed in this paper such as integration with functional requirements, and its usefulness also decreases when a more fine-grained analysis is needed [1]. Automation of this analysis approach and thus its applicability for large and complex systems is another weakness of this method, particularly, in cases where trade-off analysis might need to be done several times during the development process and lifecycle of a product.

While deciding on the satisfaction of NFRs is mainly considered to be subjective, there are several works that try to provide quantifications for NFRs to ease their evaluation and analysis. Kassab et al. in [3, 32] offer a method to quantify NFR size in a software project based on the functional size measurement method to help with the estimation of effects of NFRs on the effort of building the software in a quantitative manner. In a more recent work in [33], Kassab also proposes to incorporate Analytical Hierarchy Process (AHP) with the NFR framework. AHP is a mathematical based trade-off technique whose combination with the NFR framework enables to quantitatively deal with ambiguities, trade-offs, priorities and interdependencies among NFRs and operationalizations. An approach is introduced in [4] which makes use of Requirements Hierarchy Approach (RHA) as a quantifiable method to measure and manipulate the effects that NFRs have on a system. It does so by capturing the effects of functional requirements. In [34], an approach for quantifying NFRs based on the characteristics of and information from execution domain, application domain and component architectures is suggested. Moreover, an interesting quantitative approach for discovering the dependencies of quality metrics and identifying their impacts in the architecture of a system is provided in [35].

While models used to be thought mainly just as another form of documentation during the development process, with the introduction of model-based development and further maturation of this field, models have got a more important role as in the automatic generation of code and performing different types of analysis at earlier phases of development, and thus saving time and effort by identifying problems earlier. Aligned with this direction, there are several works that provide dif-

ferent forms of solutions for modeling requirements. For modeling SIG and concepts of NFR framework to represent NFRs as UML elements, a UML profile is provided in [36] to help with integration of the graph of NFRs with functional parts of the system (that are modeled in UML). Considering that NFRs and design decisions are usually specified in an informal way and as a separate document with poor or no traceability to architectural elements, [37] offers two UML profiles for modeling design decisions and NFRs as first-class entities in software architecture and to maintain traceability between them and architectural elements in the system. The profile for modeling NFRs in this work, offers six stereotypes for modeling reliability, security, performance, modifiability, and scalability each with their own specific and different set of fixed properties, such as a property called 'effort' for modifiability requirement, and 'response_time' for performance. In contrast, in our work, we have tried to provide a generic way for modeling for all NFRs regarding of their specificities (i.e., performance or security, etc.), and more importantly, with the goal of enabling designers to perform trade-off analysis on them.

In the telecommunication domain, the Telecommunication Standardization Sector (ITU-T) [38], has suggested User Requirements Notation (URN) for modeling requirements which consists of Goal-Oriented Requirement Language (GRL) and Use Case Maps (UCM). GRL is basically defined to models goals and non-functional requirements in the form goals and sub-goals, while UCM is used to describe functional scenarios. There are also some works done to define these languages as UML profiles such as [39] for GRL. As another example, for modeling security requirements, UMLsec [40] is suggested that comes with an analysis suite which enables performing analysis on the model to identify violations of security requirements. SysML [41] which is both an extension and subset of UML 2 was offered by Object Management Group (OMG) for system engineering. SysML enables to represent requirements as first-class model elements by providing a package for generic modeling of requirements (both NFRs and FRs) and the relationships among them. Different types of associations which are provided in SysML to model the relationships between the requirements include: *copy*, *deriveReqt*, *satisfy*, *verify*, *refine* and *trace*. While SysML does not specifically focus on NFRs and analysis of them, our approach and SysML can be used together to complement each other. , EAST-ADL[42] which is developed for modeling software architecture and electronic parts of automotive systems, makes use of SysML requirements semantics for modeling requirements and special-

izes them to match the needs of automotive domain (e.g., definition of timing, delay and safety requirements). In relation to our discussion on non-functional requirements and the difference between a requirement and a property, it is worth here to also mention the UML profile for Modeling and Analysis of Real-time Embedded Systems (MARTE) [43] which offers a rich set of semantics for modeling non-functional properties and supporting analysis of them, such as schedulability analysis.

6.8 Summary and Conclusion

In this paper, we introduced a UML-based approach for generic modeling of NFRs and performing trade-off analysis on them. By identifying and discussing different challenges related to the treatment of NFRs during the development process, we formulated what information is required to be incorporated in the models of NFRs to include them as first-class entities as part of a system's architecture and enable their trade-off analysis. Through an example, it was demonstrated how the approach can be applied and how it helps to evaluate a system design with respect to the satisfaction of its NFRs. It was also shown that using the suggested approach designers can evaluate different design alternatives and get a better idea of the satisfiability of each. Moreover, the analysis highlights problematic parts of the system through the deviation indicator value which hints to the designers which parts of the system need to be reconsidered and are good candidates for improvement, taking into account the preferences of the customers. As another contribution of this work, we applied a model transformation technique to provide support for automatic analysis of the model. The possibility to analyze models of NFRs in an automatic way is particularly essential for large and complex systems and also to ease performing the analysis as many times as needed. The latter is also useful in the evolution of software architecture [44] as requirements and features are modified or new ones are added during the lifecycle of a software product and thus analysis of NFRs (including different design alternatives) may need to be performed again and again.

It was also discussed how the introduced approach can be extended and used in other contexts and as part of other solutions such as in optimizing a system design in terms of the satisfaction of its NFRs and also for providing runtime adaptation mechanisms and to manage different

QoS levels of a system. As future directions of this work, quantification of NFRs and how to evaluate and provide more accurate values for them is an interesting research topic in order to reduce possible inaccuracies related to their subjective specifications. Extending our approach to incorporate other available methods such as FQQSIG [2] in which NFRs and their related relationships are specified in a qualitative manner and then through a fuzzification process quantitative values are determined for them could also be another possible direction of this work. Along with this goal, it would be interesting to include several algorithms and methods in the analysis engine which the user may then select to use, and offer the approach as a complete tool suite. One point to remember though is that since the evaluation of NFRs and quality attributes is basically a subjective task, the methods and tools provided for this purpose serve actually as helpers for system designers to make better and more accurate evaluations and decisions.

6.9 Acknowledgements

This work has been partially supported by the Swedish Knowledge Foundation (KKS) through the ITS-EASY industrial research school [45] and by Xdin AB [46] in the scope of the MBAT European Project [47].

Bibliography

[1] Tegegne Marew, Joon-Sang Lee, and Doo-Hwan Bae. Tactics based approach for integrating non-functional requirements in object-oriented analysis and design. *The Journal of Systems and Software*, 82:1642–1656, October 2009.

[2] Ming-Xun Zhu, Xin-Xing Luo, Xiao-Hong Chen, and Desheng Dash Wu. A non-functional requirements tradeoff model in Trustworthy Software. *Elsevier Journal of Information Sciences*, 191:61–75, May 2012.

[3] Mohamad Kassab, Olga Ormandjieva, Maya Daneva, and Alain Abran. Software Process and Product Measurement. chapter Non-Functional Requirements Size Measurement Method (NFSM) with COSMIC-FFP, pages 168–182. Springer-Verlag, Berlin, Heidelberg, 2008.

[4] Andrew J. Ryan. An Approach to Quantitative Non-Functional Requirements in Software Development. In *Proceedings of the 34th Annual Government Electronics and Information Association Conference*, 2000.

[5] N.S. Rosa, P.R.F. Cunha, and G.R.R. Justo. ProcessNFL: a language for describing non-functional properties. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3676 – 3685, jan. 2002.

[6] Yi Liu, Zhiyi Ma, and Weizhong Shao. Integrating Non-functional Requirement Modeling into Model Driven Development Method. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 98 –107, December 2010.

[7] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The architecture tradeoff analysis method. In *Engineering of Complex Computer Systems, 1998. ICECCS '98. Proceedings. Fourth IEEE International Conference on*, pages 68 –78, Aug 1998.

[8] Thomas Henzinger and Joseph Sifakis. The Embedded Systems Design Challenge. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg.

[9] Bran Selic. The Pragmatics of Model-Driven Development. *IEEE Software Journal*, 20:19–25, September 2003.

[10] Bran Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07. 10th IEEE International Symposium on*, pages 2 –9, May 2007.

[11] Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno. An Introduction to UML Profiles. In Rafael F. Calvo, editor, *The European Journal for the Informatics Professional - UML and Model Engineering*, volume V, April 2004.

[12] Ingo Weisemöller and Andy Schürr. A Comparison of Standard Compliant Ways to Define Domain Specific Languages. pages 47–58, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] G. Kotonya and I. Sommerville. Requirements engineering with viewpoints. *Software Engineering Journal*, 11(1):5 –18, jan 1996.

[14] Pete Sawyer and Gerald Kotonya. Chapter 2-Software Requirements. In *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, May 2001.

[15] Martin Glinz. On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference*, pages 21–26, New Delhi, India, October 2007.

[16] Lawrence Chung and Julio Cesar Prado Leite. Conceptual Modeling: Foundations and Applications. chapter On Non-Functional Requirements in Software Engineering, pages 363–379. Springer-Verlag, Berlin, Heidelberg, 2009.

[17] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, 1990.

[18] Systems and software engineering – Vocabulary (IEEE Standard). *ISO/IEC/IEEE 24765:2010(E)*, 15 2010.

[19] Nelson S. Rosa, George R. R. Justo, and Paulo R. F. Cunha. A framework for building non-functional software architectures. In *Proceedings of the 2001 ACM symposium on Applied computing*, SAC '01, pages 141–147, New York, NY, USA, 2001. ACM.

[20] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems. In *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, October 2011.

[21] Andreas Borg, Angela Yong, Pär Carlshamre, and Kristian Sandahl. The Bad Conscience of Requirements Engineering : An Investigation in Real-World Treatment of Non-Functional Requirements. In *Third Conference on Software Engineering Research and Practice in Sweden (SERPS'03), Lund*, 2003.

[22] Andreas Borg, Mikael Patel, and Kristian Sandahl. Good Practice and Improvement Model of Handling Capacity Requirements of Large Telecommunication Systems. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference*, Washington, DC, USA, 2006.

[23] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: a process-oriented approach. *Software Engineering, IEEE Transactions on*, 18(6):483 –497, jun 1992.

[24] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*, volume 5 of *International Series in Software Engineering*. Springer, October 1999.

[25] MDT Papyrus. `http://www.eclipse.org/modeling/mdt/papyrus/`, Accessed: July 2012.

[26] Eclipse Modeling Framework Project (EMF). `http://www.eclipse.org/modeling/emf/`, Accessed: July 2012.

[27] QVT Operational Language. `http://www.eclipse.org/m2m/`, Accessed: July 2012.

[28] Antti Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets*, pages 429–528, London, UK, UK, 1998. Springer-Verlag.

[29] John P. J. Kelly, Thomas I. McVittie, and Wayne I. Yamamoto. Implementing Design Diversity to Achieve Fault Tolerance. *IEEE Software Journal*, 8:61–71, July 1991.

[30] I. Crnkovic, M. Chaudron, and S. Larsson. Component-Based Development Process and Component Lifecycle. In *Software Engineering Advances, International Conference on*, page 44, Oct 2006.

[31] Hridesh Rajan and Kevin J. Sullivan. Classpects: unifying aspect- and object-oriented language design. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 59–68, New York, NY, USA, 2005. ACM.

[32] M. Kassab, M. Daneva, and O. Ormandjieva. Early Quantitative Assessment of Non-Functional Requirements, June 2007.

[33] M. Kassab. An integrated approach of AHP and NFRs framework. In *Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on*, pages 1–8, May 2013.

[34] Raquel Hill, Jun Wang, and Klara Nahrstedt. Quantifying Non-Functional Requirements: A Process Oriented Approach. In *Proceedings of the Requirements Engineering Conference, 12th IEEE International*, pages 352–353, Washington, DC, USA, 2004. IEEE Computer Society.

[35] Anakreon Mentis, Panagiotis Katsaros, Lefteris Angelis, and George Kakarontzas. Quantification of interacting runtime qualities in software architectures: Insights from transaction processing in client-server architectures. *Information and Software Technology Journal*, 52(12):1331–1345, December 2010.

[36] Sam Supakkul. A UML profile for goal-oriented and use casedriven representation of NFRs and FRs. In *In Proceedings of the 3rd International Conference on Software Engineering Research, Management and Applications*, pages 112–121, 2005.

[37] Liming Zhu and Ian Gorton. UML Profiles for Design Decisions and Non-Functional Requirements. In *Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, SHARK-ADI '07, pages 8–, Washington, DC, USA, 2007. IEEE Computer Society.

[38] Telecommunication Standardization Sector (ITU-T). `http://www.itu.int/en/pages/default.aspx`, Accessed: July 2012.

[39] Muhammad R. Abid, Daniel Amyot, Stéphane S. Somé, and Gunter Mussbacher. A UML profile for goal-oriented modeling. In *Procs. of SDL'09*, 2009.

[40] Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 412–425, London, UK, 2002. Springer-Verlag.

[41] OMG SysML Specifcation V1.2. `http://www.sysml.org/specs.htm`, Accessed: July 2012.

[42] EAST-ADL Specification V2.1. `http://www.atesst.org`, Accessed: June 2012.

[43] OMG. MARTE specification version 1.1. `http://www.omgmarte.org`, Accessed: July 2012.

[44] Hongyu Pei-Breivold, Ivica Crnkovic, and Magnus Larsson. A systematic review of software architecture evolution research. In *Journal of Information and Software Technology*. Elsevier, doi:10.1016/j.infsof.2011.06.002, July 2011.

[45] ITS-EASY post graduate industrial research school for embedded software and systems. `http://www.mrtc.mdh.se/projects/itseasy/`, Accessed: June 2013.

[46] Xdin AB. `http://xdin.com/`, Accessed: July 2012.

[47] MBAT Project: Combined Model-based Analysis and Testing of Embedded Systems. http://www.mbat-artemis.eu/home//, Accessed: July 2012.

Chapter 7

Paper B: Managing Timing Implications of Security Aspects in Model-Driven Development of Real-Time Embedded Systems

Mehrdad Saadatmand, Thomas Leveque, Antonio Cicchetti, Mikael Sjödin
International Journal On Advances in Security, Vol. 5, No. 3&4, December, 2012.

Abstract

Considering security as an afterthought and adding security aspects to a system late in the development process has now been realized to be an inefficient and bad approach to security. The trend is to bring security considerations as early as possible in the design of systems. This is especially critical in certain domains such as real-time embedded systems. Due to different constraints and resource limitations that these systems have, the costs and implications of security features should be carefully evaluated in order to find appropriate ones which respect the constraints of the system. Model-Driven Development (MDD) and Component-Based Development (CBD) are two software engineering disciplines which help to cope with the increasing complexity of real-time embedded systems. While CBD enables the reuse of functionality and analysis results by building systems out of already existing components, MDD helps to increase the abstraction level, perform analysis at earlier phases of development, and also promotes automatic code generation. By using these approaches and including security aspects in the design models, it becomes possible to consider security from early phases of development and also identify the implications of security features. Timing issues are one of the most important factors for successful design of real-time embedded systems. In this paper, we provide an approach using MDD and CBD methods to make it easier for system designers to include security aspects in the design of systems and identify and manage their timing implications and costs. Among different security mechanisms to satisfy security requirements, our focus in this paper is mainly on using encryption and decryption algorithms and consideration of their timing costs to design secure systems.

7.1 Introduction

To cope with the specific challenges of designing security for real-time embedded systems, appropriate design methods are required. Due to resource constraints in these systems, the implications of introducing security and its impacts on other aspects and properties of the system should be carefully identified as early as possible and the methods used for designing these systems should provide such a feature [1]. Timing properties are of utmost importance in real-time embedded systems. In this paper, we introduce an approach using Model-Driven and Component-Based Development (MDD & CBD) methods for designing secure embedded systems to bring security aspects into early phases of the development and take into account their timing costs and implications.

This work provides an implementation and a methodology for the generic idea that we discussed in [1] and extends it with the result of our works in [2, 3]. In this work we provide a more complete approach and methodology, compared to the two aforementioned works, based on their combination and synergy and discuss how this approach can cover more issue regarding the timing implications of security mechanisms in real-time embedded systems.

The approach basically works by identifying and annotating sensitive data in the component model of the system, and then deriving automatically a new component model which includes necessary security components for the protection of the data. Our main focus in this paper will be on using encryption and decryption algorithms as security mechanisms. The derivation of the new component model is based on a set of pre-defined strategies. Each strategy defines a different set of possible encryption and decryption algorithms to be used as the implementation of the security components. In this approach, since the derived component model conforms to the original meta model, the same timing analysis and synthesis as for the original component model can be used and applied for the derived one.

With the increasing role of computer systems in our daily lives, there is hardly any software product developed these days that does not have to deal with security aspects and protect itself from malicious adversaries [4]. Also with the exponentially growing number of connected and networked devices and more integration between different tools and services that store and exchange different types of data, not only new types of

attacks are constantly emerging but also the risks and consequences of security breaches have become more drastic. Even some simple software products and applications which do not store any sensitive information and therefore may not seem to need to care about security aspects can, for example, be the target of buffer overflow attacks [5] and thus help attackers in gaining access to a system. All these points emphasize that security aspects cannot be taken into account just as an afterthought and added feature to an already developed system [6], but instead should be considered at different phases of development from early phases such as requirements engineering to deployment [4]. What is needed is that instead of adding security features in an "eggshell approach", security should be designed intrinsically and inseparable from the application to be able to address the threats that target the application itself [6].

Considering security from early phases of development is especially critical in the design of real-time embedded systems. These systems typically have limited amount of resources (e.g., in terms of available memory, CPU and computational power, energy and battery) and therefore, implications of security features should also be taken into account. This is basically because of the fact that Non-Functional Requirements (NFRs), such as security, are not independent and cannot be considered in isolation and satisfying one can affect the satisfaction of others [7]. Therefore, costs and implications of security features should be identified to analyze the trade-offs and establish balance among different non-functional requirements of the system. Such costs can be in the form of impacts on timing, schedulability and responsiveness of the system, as well as memory usage, energy consumption, etc. In real-time embedded systems, satisfaction of timing requirements is critical for the successful behavior of the system, therefore, choice of security mechanisms should be done considering their timing characteristics and impacts.

Model-driven development is a promising approach to cope with the design complexity of real-time embedded systems. It helps to raise the abstraction level, enables analysis at earlier phases of development and automatic generation of code [8, 9]. Component-based development, on the other hand, is another discipline in software engineering and a software development method in which systems are built out of already existing components as opposed to building them from scratch [10, 11]. In other words, it promotes developing a system as an assembly of components by reusing already existing software units (components). Model-driven development and component-based development approaches can

be used orthogonally to complement and reinforce each other to alleviate the design complexity of real-time embedded systems [10].

In this context, including security aspects in design models helps with achieving the two goals mentioned so far: bringing security aspects into earlier phases of development and enabling analysis of security implications. Moreover, model-based security analysis (not the focus of this paper) in order to identify possible violations of security requirements [12] becomes possible and also system designers with lower levels of expertise and knowledge in security domain can also include and express security concerns [2]. The latter is due to the fact that code level implementation of security features requires detailed security knowledge and expertise, while at the model level, system designers can use modeling concepts and annotations for expressing security concerns (which in turn may also be used for automatic generation of security implementations).

By constructing the model of the system including security features, timing analysis can then be done on the model to evaluate whether the model meets the timing requirements or not. If so, the implementation of the system can then be generated from the model(s). This leads to a fixed set of security mechanisms that are already analyzed as part of the whole system in terms of their timing behaviors and are thus known to operate within the timing constraints of the system. However, there are situations where such a guarantee in terms of timing behaviors cannot be achieved. For example, in performing analysis some assumptions are taken into account, such as worst-case execution times of tasks. If these assumptions are violated at runtime, the analysis results will not hold anymore. Moreover, in complex real-time systems where timing analysis is not practical/economical or not much information about the timing characteristics of each individual task is available, other approaches are needed in order to tackle the timing issues [13]. One solution is to have runtime adaptation to mitigate timing violations and keep the execution of tasks within their allowed time budgets.

The remainder of the paper is structured as follows. In Section 7.2, we discuss the issue of security and its challenges in embedded systems in general. In Section 7.3, the automatic payment system is described as the motivating example of this paper and also as an example of distributed real-time embedded systems with security requirements. The suggested approach and its implementation are described in Sections 7.4 and 7.5. In Section 7.6, we introduce a runtime adaptation mechanism to mitigate the violations of timing constraints at runtime. Practical aspects of the

introduced approach and other related issued are discussed in Section 7.7. Section 7.8 discusses the related work and finally in Section 7.9 conclusions are made.

7.2 Security in Embedded Systems

In the design of embedded systems, security aspects have often been neglected [14]. However, the use of embedded systems in critical applications such as aviation systems, controlling power plants, vehicular systems control, and medical devices makes security considerations even more important. This is due to the fact that there is now a tighter relationship between safety and security in these systems (refer to [15] for the definitions of security and safety and their differences).

Also because of the operational environment of embedded systems, they are prone to specific types of security attacks that might be less relevant for other systems such as a database server inside a bank which is physically isolated and protected, in contrast to smart cards and wireless sensor networks which are physically exposed. Physical and side channel attacks [16] are examples of these types of security issues in embedded systems that bring along with themselves requirements on hardware design and for making systems tamper-resistant. Examples of side channels attack could be the use of time and power measurements and analysis to determine security keys and types of used security algorithms.

Increase in the use and development of networked and connected embedded devices also opens them up to new types of security issues. Features and devices in a car that communicate with other cars (e.g., the car in front) or traffic data centers to gather traffic information of roads and streets, use of mobile phones beyond just making phone calls and for purposes such as buying credits, paying bills, and transferring files (e.g. pictures, music,etc.) are tangible examples of such usages in a networked environment.

Besides physical and side channel attacks, often mobility and ease of access of these devices also incur additional security issues. For example, sensitive information other than user data, such as proprietary algorithms of companies, operating systems and firmwares, are also carried around with these devices and need protection.

Because of the constraints and resource limitations in embedded systems, satisfying a non-functional requirement such as security requires

careful balance and trade-off with other requirements and properties of the systems such as performance and memory usage. Therefore, introducing security brings along its own impacts on other aspects of the systems. This further emphasizes the fact that security cannot be considered as a feature that is added later to the design of a system and needs to be considered from early stages of development and along with other requirements. From this perspective, there are many studies that discuss the implications of security features in embedded systems such as [16], in which considering the characteristics of embedded systems, major impacts of security features are identified to be on the following aspects:

- Performance: Security protocols and mechanisms incur heavy computational demands on a system that the processing capacity of an embedded system might not be able to satisfy easily. For example, using encryption and decryption algorithms not only have high computational complexity but also require good amount of memory. In systems that need to handle heavy input loads, such as routers and many systems that are used in telecommunication domain to handle calls and data traffics, these security features can consume lots of processing capacity of the system and result in missed deadlines of other tasks, dropped throughput level, and overall transaction and data rate of the system.

- Power Consumption: In embedded systems with limited power sources, any resource-consuming feature impacts the operational life of the system. In this regard, security features with their heavy computational and memory demands, as discussed above, require careful considerations. There are studies that investigate this issue and compare power consumption of different encryption/decryption algorithms such as [17] that looks at this issue in wireless sensor networks. The issue of power consumption is especially interesting knowing that the growth of battery capacities are a lot slower and far behind the ever-increasing power requirements of security features. This has also led to investigating optimized security protocols for embedded systems and hardware security solutions [16].

- Flexibility and Maintainability: Flexibility of security features and possibility to adapt them according to new requirements is also a

challenge in embedded systems. For example, embedded devices such as mobile phones that are used in different operational modes and environments need to support a variety of security protocols. Moreover, security solutions need to be updated in order to be protected against emerging hacking methods. Therefore, flexibility of security design decisions is important for maintaining the security of the system to apply updates and patches.

- Cost: Cost is also a limiting factor in the design of embedded systems. Considering the issues mentioned above, using a faster and more expensive CPU or adding more memory modules to cope with the demands of security requirements can add to the total cost of an embedded system. Taking into account that these devices are often produced in large amounts (e.g. mobile phones and vehicular systems), a small increase in cost can affect overall revenues and competitive potentials of a product in the market. Therefore, the security features that are implemented in embedded systems should be balanced with hardware requirements and consequently cost limits.

7.3 Motivation Example: Automatic Payment System

Figure 7.1 shows the Automatic Payment system which is an example of distributed embedded systems with real-time and security requirements. The main goal in the design of this system is to allow a smoother traffic flow and reduce waiting times at tolling stations (as well as parkings).

For each toll station, a camera is used to detect a vehicle that approaches the station (e.g. at 100/200 meter distance), and scans and reads its license plate information. This information is passed to the payment station subsystem which then sends the toll fee to the vehicle through a standardly defined wireless communication channel. This amount is shown to the driver in the vehicle through its User Interface (UI) and the driver inserts a credit card and accepts the payment to be done. The credit card number is then sent securely to the payment station which then performs the transaction on it through a (third party) merchant (e.g., via a wired Internet connection at the station). The driver is then notified about the success of the transaction and receives

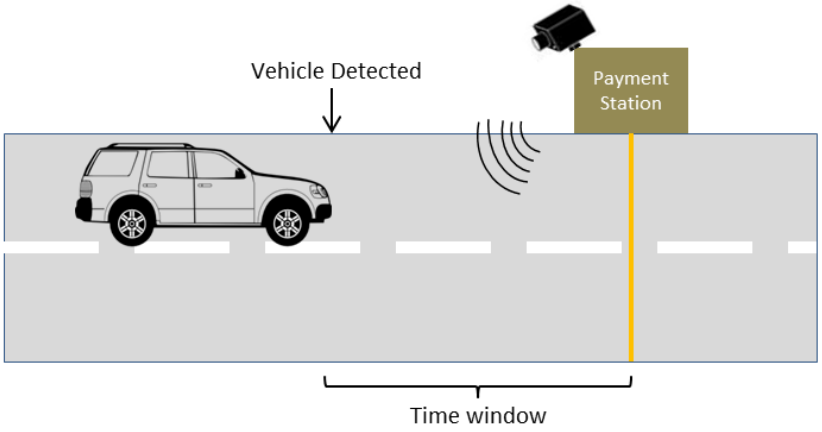

Figure 7.1: Automatic Payment System for Toll Roads.

an *OK* message to go accordingly. The interactions between different objects in this system are shown in Figure 7.2.

To allow a smooth traffic flow, all these operations should be done in a certain time limit. Such time constraints can be calculated considering the specifications of camera and its required time for the detection of an approaching vehicle, traffic and safety regulations (e.g, allowed speed), and other similar factors. For example, if the vehicle is detected at 100 meter distance from the station, and the allowed speed at that point is 20 km/h, then the system has a strict time window during which it should be able to store the vehicle information, establish communication, and send the payment information to it. Different scenarios can happen in this system. For example, it could happen that the driver/vehicle fails to provide credit card information, or the credit card is expired. In this case, the system can log the vehicle information in a certain database and send the bill later to the owner, or even it can be set to not open the gate for the vehicle to pass and also show a red light for other cars approaching that toll station to stop. Besides the mentioned timing constraints that exist in this system, the communication between different nodes and transfer of data need to be secured and protected. In this system, we have the following security requirements:

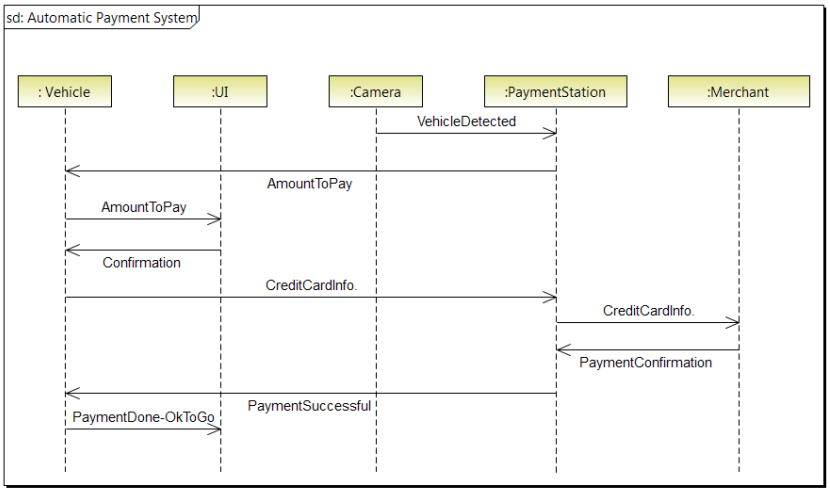

Figure 7.2: Automatic Payment System.

1. Sensitive data such as credit card information should not be available to unauthorized parties.
2. The vehicle only accepts transactions initiated by the payment station.

To achieve these requirements, the station needs to authenticate itself to the vehicle so that the vehicle can trust and send the credit card information. Moreover, sensitive information that is transferred between different parts should also be encrypted.

Another scenario that can happen in this system is that several vehicles may approach one station with a short time distance between each which can result in bursty processing loads on the system (analogous to bursty arrivals of aperiodic tasks in real-time terms). In such situations, timing requirements may be violated as even by using static analysis of the system, only certain levels of such bursty loads may be covered and not all the possible cases. One solution to mitigate timing violations in this scenario is to introduce runtime adaptation and adapt the security level of the system at runtime; meaning that security mechanisms that are less time-consuming (and presumably less strong) can be used when

such situations are detected. As the last resort, when the system realizes that many number of timing violations are occurring, to maintain a smooth traffic flow and prevent any possible accidents and safety issues due to the increasing queuing of the cars at the tolling station, instead of the on-site payment and charging of the vehicles, the system can just store their information to send a bill later to the owner of the vehicle, or even add the amount to the payment done at the next tolling stations on the road (if there are any and they are connected).

To model and build the system (software parts), particularly considering the timing constraints of the system, the following challenges are identified:

1. Modeling security mechanisms with enough details to enable both timing analysis on the model and synthesis of the security implementations,

2. Obtaining timing costs of security mechanisms,

3. Managing possible timing violations of the system at run-time.

The first challenge is discussed in the following two sections. To get the timing costs of security mechanisms, we rely on studies such as [18] that have done such measurements. To solve the third challenge, a runtime adaptation mechanism is introduced and we show how it helps to mitigate the runtime violations of timing constraints.

7.4 Approach

Based on the identified challenges in the previous example, we introduce an approach that aims to bring the security concerns in the design of embedded systems. Our suggested approach helps systems designers in expressing the security concerns in a system without the need to have much security expertise on the actual implementation of security mechanisms. It does so by just requiring the system designers to identify sensitive data entities that need to be protected. In the scope of our work, it can be for example the data that need to be confidential and/or whose sender must be authenticated. Moreover, to mitigate potential timing violations of security mechanisms at runtime, the approach provides the option to include an adaptation feature for the security mechanisms.

To implement the approach, ProCom [19] component model has been used; although the approach is not dependent on this specific component model and can be implemented using other component models as well. Security needs are specified as annotations on the component model. A benefit of the ProCom component model is its power in defining new attribute types using its attribute framework to annotate and specify new types of data. The term component model hereafter is used to basically refer to the component architecture model than the meta-model of ProCom. From the specification of the security needs at the data level and physical platform level, a model transformation is applied on the component model to derive a new component model including security implementations. The derivation of the new component model (which now has appropriate security components implementing the security needs) is done based on a selected strategy. The strategy basically specifies the preferences in terms of security implementations and which of them to choose among a set of different possible ones. Having the necessary information in the model, the steps that have been described so far can be summarized as follows:

1. The component model which specifies the functional and non-functional (extra-functional) part of the system is transformed into a functionally equivalent model with added security implementations;

2. Analysis can be performed on the derived component model that includes security components to identify any possible violations of timing constrains; and

3. Finally, the system is synthesized.

The considered process is iterative and allows to refine security specification after evaluating the resulted system properties such as timing properties. In other words, timing analysis, for example, can be performed on the derived component model and if timing properties of the derived model do not satisfy the timing requirements, the derivation process can be repeated with different preferences to finally gain a model which is satisfactory in terms of timing requirements. The process is depicted in Figure 7.3 showing different models and annotations that are used as input in each step (i.e., the analysis of the system model as well as synthesis of the implementation).

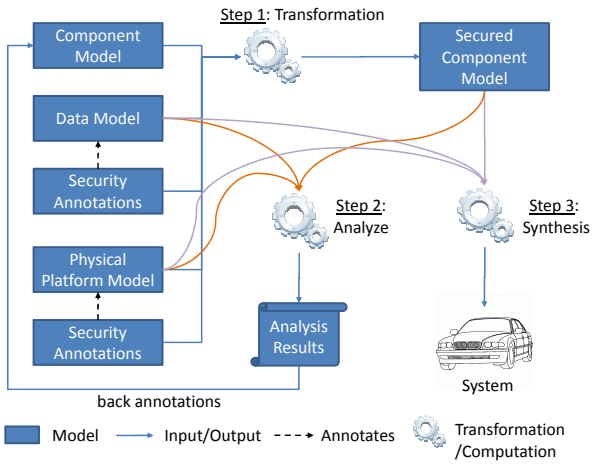

Figure 7.3: General description of the approach process.

7.5 Implementation

7.5.1 ProCom Component Model

While the approach principles are component model generic, we implemented it using ProCom. The ProCom component model targets distributed embedded real-time system domain. In particular, it enables to deal with resource limitations and requirements on safety and timeliness concerns. ProCom is organized in two distinct layers that differ in terms of architectural style and communication paradigm. For this paper, however, we consider only the upper layer which aims to provide a high-level view of loosely coupled subsystems. This layer defines a system as a set of active, concurrent subsystems that communicate by asynchronous message passing, and are typically distributed. Figure 7.4 shows ProCom design of the Automatic Payment System example.

A subsystem can internally be realized as a hierarchical composition of other subsystems or built out of entities from the lower layer of ProCom. Figure 7.5 shows the implementation of the subsystem E as an assembly of two component C1 and C2. Data input and output ports are

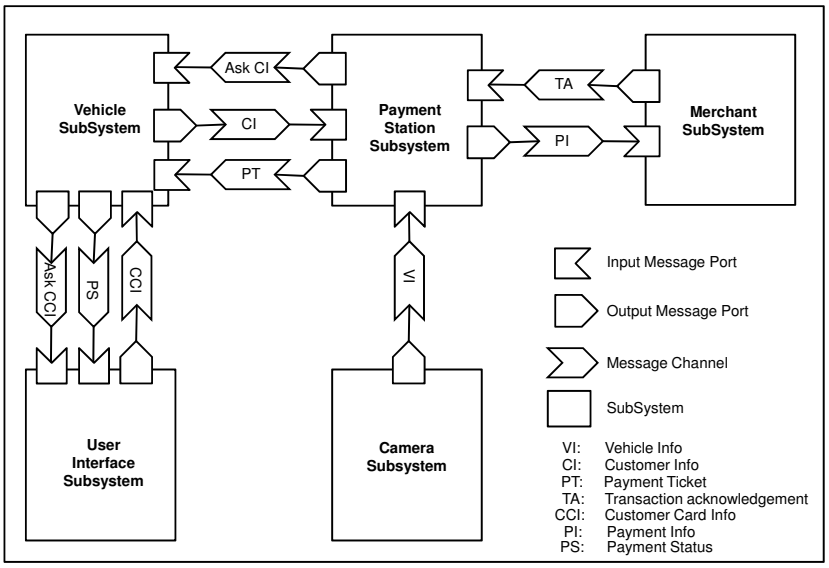

Figure 7.4: Component Model of the System using ProCom.

denoted by small rectangles, and triangles denote trigger ports. Connections between data and trigger ports define transfer of data and control, respectively. Fork and Or connectors, depicted as small circles, specify control over the synchronization between the subcomponents.

7.5.2 Data Model

As components are usually intended to be reused, their related data may also be reused. To this end, we propose to extend the data-entity approach described in [20] for design-time management of data in component-based real-time embedded systems. In this approach every data entity is stored in a shared repository and designers are provided with an additional architectural view for data management, namely the data architectural view. The description of a data entity contains its type (string, int...), its maximum size and its unit. A data entity can also be a composite entity defined as a list of data entities. We use the concept of data entity to identify data that are transfered through different message channels in the system (shown in Figure 7.4) and map them to

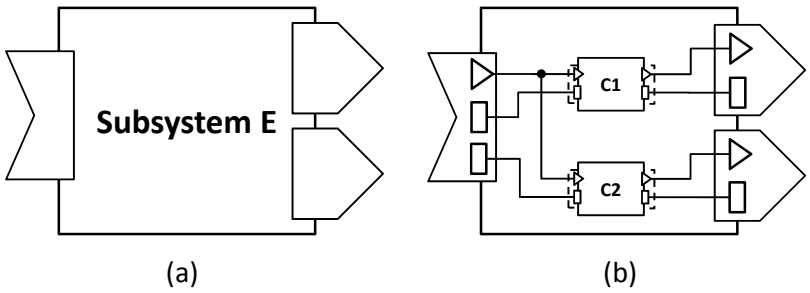

Figure 7.5: ProCom SubSystem Implementation.

their respective security concerns (e.g., if they need to be encrypted and protected or not). Table 7.1 and Table 7.2 show the data entities in our example. As described in the last section, subsystems communicate

Table 7.1: Primitive Data Entities.

Data Entity	Type	Max Size	Unit
CCNumber	String	16	byte
ExpirationDate	String	4	byte
AskCI	Empty	0	byte
AskCCI	Empty	0	byte
PaymentStatus	boolean	1	byte
VehicleNumber	String	20	byte
VehicleType	Enum	8	byte
AmountToPay	float	4	euro

through asynchronous message passing represented by message channels. A message channel is associated with a list of data entities which defines the message content. Table 7.3 presents the mapping between message channels and data entities for our example. We can observe that the same data entity can be used several times in different message channels. The mapping between data ports of message ports and data entities is based on naming convention which enables to distinguish between the data ports that require to encrypt/decrypt their data and those that do not. We call data model the set of data entities which are

Table 7.2: Composite Data Entities.

Data Entity	Contains
CreditCard	CCNumber, ExpirationDate
CustomerInfo	VehicleNumber, CreditCard
PaymentTicket	AmountToPay, PaymentStatus
PaymentRequest	AmountToPay, CreditCard

Table 7.3: Mapping between Data Entities and Message Channels.

Message Channel	Data Entities
AskCI	AskCI
CI	CustomerInfo
PT	PaymentTicket
AskCCI	AskCCI
PS	PaymentStatus
CCI	CreditCard
VI	VehicleNumber, VehicleType
TA	CCNumber, AmountToPay, PaymentStatus
PI	PaymentRequest

used in the related design.

7.5.3 Physical Platform And Deployment Modeling

The physical entities and their connections are described in a separate model called Physical Platform Model (see Figure 7.6). This model defines the different Electronic Computation Units (ECUs), called Physical Nodes, including their configurations such as processor type and frequency, the connections between the physical nodes, and the physical platforms which represent a set of ECUs fixed together.

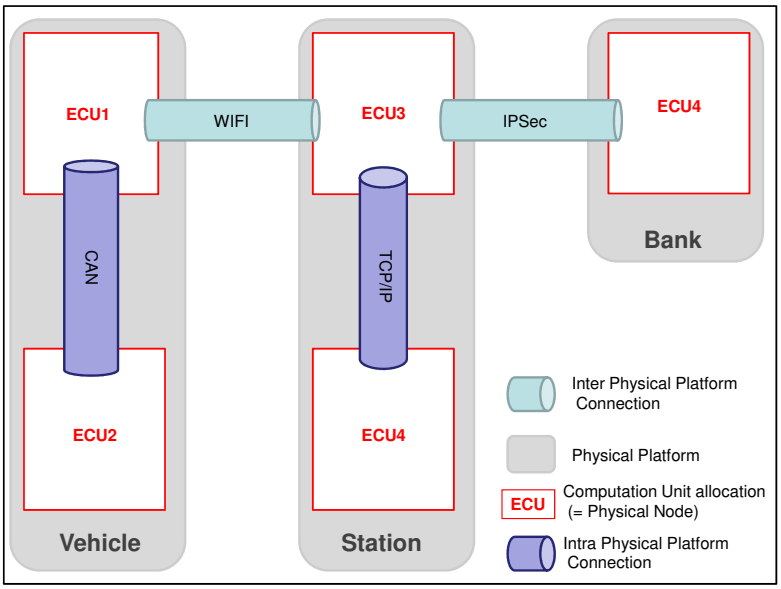

Figure 7.6: Physical Platform Model of the System.

ProCom system deployment is modeled in two steps, introducing an intermediate level where subsystems are allocated to virtual nodes that, in turn, are allocated to physical nodes. In a similar way, message connections are allocated to virtual message connections which, in turn, are allocated to physical connections. Figure 7.7 defines the physical platform and related mapping of Automatic Payment System model. To

simplify the example, we assume a one to one mapping between virtual node and physical node.

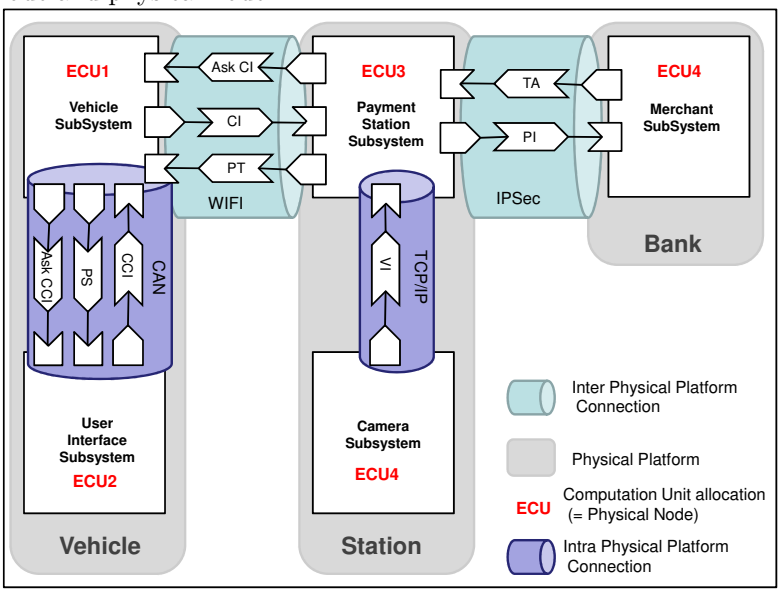

Figure 7.7: Deployment Model of the System depicting allocation to Physical Platforms.

7.5.4 Security Properties

Instead of defining the security properties on the architecture, i.e. the component model, we propose to annotate the data model and compute the required security properties on the architecture, based on these security requirements. It is an original part of our approach where a designer can think about sensitive data without considering the architecture models. The designer applies security properties to identify and annotate sensitive data in the system, which require to be protected using some security mechanisms (e.g., confidentiality and encryption, authentication, integrity, etc.). We consider two types of security properties:

- **Confidentiality** ensures that the considered information cannot

be read by any external person of the system; and

- **Authentication** which ensures that the considered information comes from the expected sender.

Table 7.4 shows security annotations associated to data entities for our example. In addition to security properties on the data model, we

Table 7.4: Data Entity Security Properties.

Data Entity	**Security properties**
CCNumber	Confidentiality
VehicleNumber	Authentication
AskCI	Authentication
AskCCI	Authentication
PaymentRequest	Authentication
PaymentStatus	Authentication

define the security properties related to the physical platform which are independent of any application:

- **Exposed** defines that the physical platform is potentially accessible to external persons and that they may be able to open it and modify physical parts.
- **NotAccessible** defines that the physical platform is not considered as accessible to unauthorized persons.

In a similar way, physical connections are annotated:

- **Secured** defines that the physical connection is considered as secured due to its intrinsic security implementation.
- **NotSecured** defines that the physical connection protocol does not implement a reliable security (opposite of the above).

Using these properties, the person responsible for the physical platform annotates physical entities and the physical connections between them in the platform model. Thanks to these annotations, we can deduce which parts do not need additional security implementations if it is already provided (by construction). For example, if a link is established

using mere TCP/IP, it is annotated as NotSecured, while in case that IPSec protocol suite is used for a link, that link is annotated as Secured. This means that the link is considered trusted and already secured, and no security component is necessary to be added for the link. Table 7.5 shows the security properties of Automatic Payment System physical platforms.

Table 7.5: Security Properties of Physical Entities.

Physical Platform or Connection	**Security properties**
Vehicle	Exposed
Station	NotAccessible
Bank	NotAccessible
WIFI	NotSecured
IPSec	Secured
TCP/IP	NotSecured
CAN	NotSecured

7.5.5 Cost of Security Implementations

Different encryption/decryption algorithms as security mechanisms can be selected to satisfy the identified security properties in the system. Considering the fact that each security mechanism in the system has its own costs in terms of timing and performance, power consumption and so on, choosing an appropriate security mechanism is critical in order to ensure the satisfaction of timing requirements of the system. For this purpose, and to take into account the timing costs of different security mechanisms, we rely on the results of studies such as [18] that have performed these cost measurements. Based on such methods, we assume the existence of such timing measurements for the platforms used in our system in the form of the Table 7.6. We assume that execution times can be computed knowing the target platform, algorithm, key size and data size. A timing estimation toolkit may also be provided which provides execution time estimates based on these measurements. As can be observed from the table, we also take into account and add this flexibility that some algorithms may not be supported on some platforms (marked as NS).

Table 7.6: Execution times and strength ranking of different security algorithms for a specific platform

Strength Rank	Algorithm	Key Size	ET-P1	ET-P2	ET-Pn
1	AES	128	NS	480	...
2	3DES	56	292	198	...
3	DES	56	835	820	...
...					

(ET-Px: Executime Time on Platform x in bytes per second, NS: Not Supported on corresponding platform)

7.5.6 Security Implementation Strategy

As mentioned previously, based on the selected strategy, a security mechanism is chosen from the table and the components implementing it are added to the component model. The user can then perform timing analysis on the derived component model to ensure that the overall timing constraints hold and are not violated. We propose several strategies to help choosing among all possible security implementations:

- The **StrongestSecurity** strategy selects the strongest security implementation available on the platforms (taking into account that some security mechanisms, namely encryption algorithms here, may not be available and possible on a certain platform, hence selecting the strongest available one);
- The **StrongestSecurityAndLimitImplemNb** strategy selects the strongest security implementation available on the platforms while ensuring that we use as few as possible different security implementations, since each message channel can use a different encryption algorithm (finding the most common security implementation which achieves the strongest level in terms of the strength rankings);
- The **LowestExecTime** strategy selects the security implementation available on the platforms which has the lowest execution time;
- The **LowestExecTimeAndLimitImplemNb** strategy selects the lowest execution time implementation available on the platforms while ensuring that we use as few as possible different security implementations; and

- The **StrongestSecuritySchedulable** strategy selects the strongest security implementation available on the platforms where the system remains schedulable.

The selection is driven by the fact that the same algorithm must be used for the sender and receiver components which may be deployed on different platforms which in turn may not support the same algorithms.

7.5.7 Transformation

The transformation is performed in four steps:

1. First, we identify the part of a message which needs to be confidential or authenticated while considering on which communication channels they are transferred;

2. Next, we add components in charge of the encryption and decryption of the identified communication channels;

3. Then, the strategies are used to choose which encryption algorithm to use and generate the code of the added components; and

4. Finally, the Worst Case Execution Time (WCET) of added components is estimated.

The transformation aims to ensure that data decryption is performed once and only once before that data will be consumed and that data encryption is performed once and only once when a message should be sent. To illustrate the algorithm, let's consider the example in Figure 7.8. We assume that only data D1 needs to be confidential. The pseudo algorithm of the transformation is described in Listing 7.1.

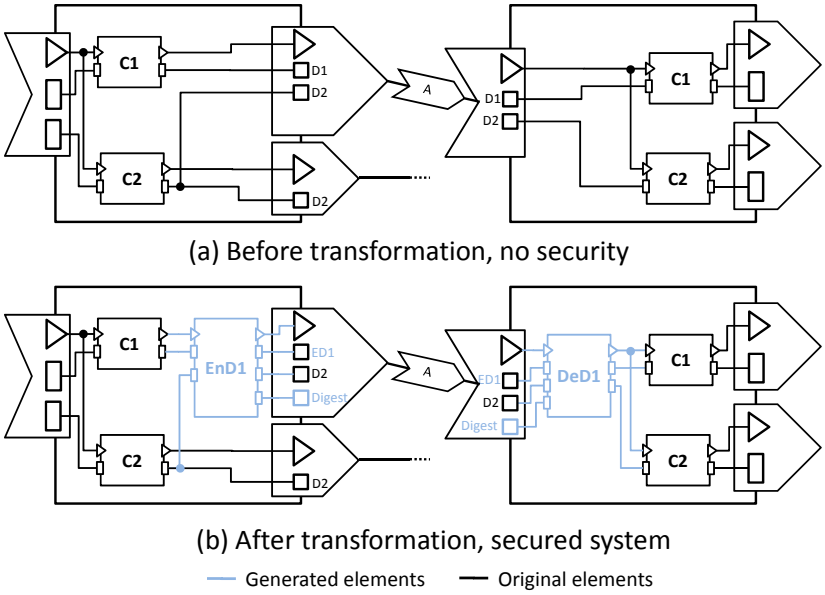

Figure 7.8: Transformation.

Listing 7.1: Transformation Pseudo Algorithm

```
msgToSecure = {}
for all channels M in component model {
 P = M.allocatedPhysicalChannel;
 if ((M.getConfidentialData() <> {}) or
      (M.getAutheticatedData() <> {}) and
    (P.isNotSecured()) and
    ((P.isIntraPlatform() and
      P.sourcePort.platform.isExposed()) or
     (P.isInterPlatform()))
        add M in msgToSecure;
}

for all M in msgToSecure {
 P = M.allocatedPhysicalChannel;

 Source = M.sourcePort;
 EnD = create component
        with same ports as Source;
 if (M.getAutheticatedData() <> {})
```

```
    add one output port Digest to EnD
    add one input port Digest to Source
 EnD.inConnections = Source.inConnections;
 create connections where EnD.outPorts
    are connected to corresponding
    Source.inPorts;
 generate EnD implementation code

 Dest = M.destPort;
 DeD = create component
       with same ports as Dest;
 if (M.getAutheticatedData() <> {})
    add one output port Digest to Dest
    add one input port Digest to DeD
 DeD.outConnections = Dest.outConnections;
 create connections where Dest.outPorts
    are connected to corresponding
    DeD.inPorts;
 generate DeD implementation code
}
```

Encryption/Decryption (in EnD1 and DeD1) is done only for confidential data while other data are just copied. An additional port is used to send the digest used for authentication. The decryption component (DeD1) ensures that all message data will be available at the same time through the output data ports. This implementation ensures the original operational semantic of the component model. Then, the security strategy is used to choose which encryption/decryption algorithm must be used and what its configuration will be.

7.6 Runtime Adaptation

The suggested approach results in a *static* and fixed set of security mechanisms to be implemented and used in each invocation and use of the system. The system model including the added security components can then be analyzed in terms of timing properties before reaching the implementation phase and therefore it can be evaluated whether the timing requirements are met or not.

There are, however, cases where such static analysis may not be possible or even economical. For example, when there is not much timing

information available about each task in the system to perform timing analysis, particularly in complex real-time systems with a big number of different tasks. In such systems even if enough timing information is available for each task, due to the complexity and big number of tasks, performing timing analysis may actually be not economical. Moreover, in performing static analysis some assumptions are taken into account and if those assumptions are violated at runtime then the static analysis results will not hold anymore. In such situations, a runtime adaptation mechanism can help to cope with the above challenges and mitigate timing violations by establishing balance between timing and security in a *dynamic* fashion.

To bring such adaptation mechanism into our approach, we introduce another strategy called **StrongestSecurityAdaptive**. By selecting this strategy, the implementation of added security components will be synthesized as depicted in Figure 7.9.

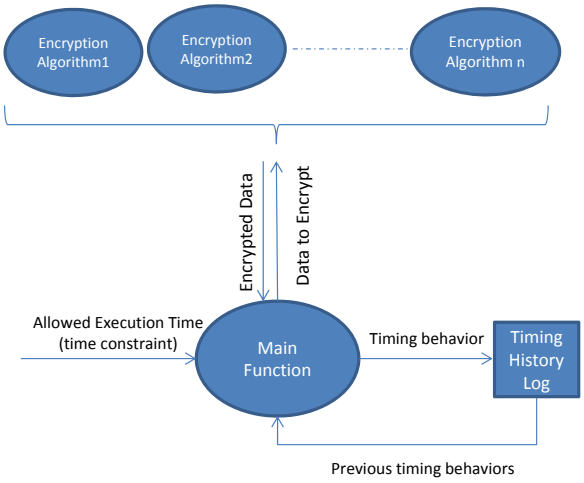

Figure 7.9: Adaptation mechanism.

As shown in Figure 7.9, by using this strategy, in the body of the added security components (here encryption ones), the implementation of all different possible encryption algorithms will also be included. When

a request for an encryption arrives, the component firstly tries to use the strongest possible encryption algorithm (based on the rank of algorithms in Table 7.6) to encrypt the data. The time it takes to perform the encryption is stored in the `Timing History Log`. If this time is more than the specified timing constraint for performing the job, then for the next encryption, another encryption algorithm with a lower rank but with less execution time will be selected (based on the information in Table 7.6).

In case the encryption job completes sooner than the specified time limit, the unused portion of its time budget is then used to determine whether it is feasible to adopt a higher ranked algorithm for the next encryption job or not. With this approach, the feedback that is produced regarding the timing behavior of encryption algorithm is used by the system to try to adapt itself. Therefore, when the system receives a burst of processing loads which it cannot fulfill under specified time constraints, it adapts itself to this higher load and similarly when the processing load decreases, it can gradually go back to using more time-consuming (and presumably more secure) encryption algorithms. This design is based on the implicit assumption that when it is detected that an executing encryption algorithm is exceeding its allowed time budget, it is basically more costly to terminate it in the middle of the encryption procedure, and restart the encryption of the data with another encryption algorithm, than just letting it finish its job, and instead use one with a lower execution time in the next invocation of encryption components.

The information that is logged in the `Timing History Log` has the following format: *Timestamp, Encryption algorithm, Time constraint, Actual execution time* (timestamp, time constraint and actual execution time are in system ticks unit in the following experiment). An example of the generated log information is shown in Table 7.7.

Considering the last row from the log as:

$$ts,\ alg,\ t,\ e$$

(ts: timestamp, alg: encryption algorithm, t: time constraint, e: actual execution time)

the decision that the system should adopt a lower ranked algorithm is made using the following formula:

(i) $e > t \Rightarrow$ *move down in the encryption algorithms table and select the next algorithm with a lower rank.*

Also, considering the two log records described as follows:

Table 7.7: Sample log information.

```
10360, AES, 50, 90
11800, 3DES, 80, 70
14500, 3DES, 60, 70
21353, DES, 60, 10
22464, 3DES, 90, 40
23112, AES, 50, 50
28374, AES, 60, 58
```

ts(l), alg(l), t(l), e(l) : representing the last log record
ts(h), alg(h), t(h), e(h) : representing the log record for the first encryption algorithm with a higher rank that was used before the last log record;

the decision to adopt a higher ranked algorithm is made using the following formula:

(ii) $e(l) < t(l) \wedge t(l) - e(l) > abs(e(h) - t(h)) \Rightarrow$ *move up in the encryption algorithms table and select the previous higher ranked algorithm.*

7.6.1 Evaluation of the Adaptation Mechanism

In [3], we have tested the introduced adaptation mechanism; here we include the evaluation results produced during that work to demonstrate the benefits of using the adaptation approach. A simulation environment was setup as described in [3] with the use of a tool called CPU Killer [21] to enforce arbitrary CPU loads at desired times.

Figure 7.10 shows the evaluation results comparing performing encryption with and without using the adaptation mechanism. In each case, CPU loads of 10%, 50%, 70%, and then back to 50%, and 10% were applied.

The columns for each log record in Figure 7.10 identify: system time (ticks), encryption algorithm (AES=1, 3DES=2, DES=3) , time constraint (for each invocation; in ticks), and actual execution time (ticks). The records in which the violation of the time constraint has occurred are marked with a '*'. Comparing the two cases (without adaptation and with it) shows that the number of time constraint violations are reduced in the second case compared to the first case where only one encryption

(I) No Adaptation		(II) With Adaptation	
10%:	580,1,300,258	10:%	584,1,300,276
	859,1,300,269		868,1,300,273
	1145,1,300,276		1155,1,300,277
	1429,1,300,274		1441,1,300,276
	1711,1,300,272		1719,1,300,268
50%:	2027,1,300,305*		2111,1,300,382A
	2474,1,300,429*		2331,2,300,203
	2918,1,300,430*	50%:	2770,1,300,428*
	3364,1,300,432*		2996,2,300,211
	3813,1,300,435*		3229,2,300,214
70%:	4482,1,300,658*		3452,2,300,203
	5399,1,300,900*		3706,2,300,243
	6301,1,300,881*	70%:	4105,2,300,381*
	7199,1,300,880*		4500,3,300,380*
	8115,1,300,898*		4880,3,300,361*
50%:	8640,1,300,505*		5257,3,300,360*
	9086,1,300,429*		5639,3,300,362*
	9529,1,300,428*		5950,3,300,294A
	9974,1,300,430*	50%:	6162,3,300,194
10%:	10285,1,300,296		6380,2,300,197
	10556,1,300,261		6594,2,300,199
	10819,1,300,251		6810,2,300,200
	11083,1,300,255		7023,2,300,200
	11349,1,300,256		7236,2,300,199
			7450,2,300,199
			7641,2,300,177
		10%:	7754,2,300,103
			8026,1,300,262
			8307,1,300,270
			8572,1,300,255
			8846,1,300,264

Figure 7.10: Performing encryption with and without adaptation.

algorithm (with the highest execution time) is used. Moreover, in the second case more number of encryption jobs have been performed under a shorter period of time.

Since the goal with this adaptive strategy is to use the strongest security algorithm possible, the adaptation mechanism assumes that the encryption algorithms in Table 7.6 are sorted according to their execution times resulting in the strongest but most time consuming one to be at the top and the weakest but less timing consuming algorithm at the bottom. Also, as a note for the decryption side, there are different ways to match and synchronize the decryption algorithm with the selected encryption algorithm. Our suggested way to do this is to add some additional bits identifying the used encryption algorithm (e.g., through the use of 2-bit or 3-bit ID numbers, according to the number of different al-

gorithms) to the encrypted message for the decryption side to correctly pick and use the appropriate algorithm. Moreover, in the introduced adaptation mechanism and its evaluation, an encryption algorithm and the respective decryption algorithm for it have been assumed to take the same amount of time which is generally valid as mentioned in [18]. However, to extend the adaptation approach for distributed systems where encryption and decryption can be performed on different nodes, more parameters for making adaptation decisions can be added. Such an extension can be to consider the sum of encryption time and decryption time for each algorithm to make adaptation decisions instead of just considering the encryption time only.

7.7 Discussion

This approach has been experimented partially in PRIDE, the ProCom development environment. The feasability at model level of the approach has been validated while the code generation part remains as future works. The security annotations have been added using the Attribute framework[22] which allows to introduce additional attribute to any model element in ProCom. The model transformation has been implemented using a QVTo[23] transformation plugged at the end of the process described in [24]. These experiments aim to show the benefits at the design level of the approach where timing properties of the overall system can be analysed. The current implementation only supports the LowestExecTime and StrongestSecurity strategies. The StrongestSecuritySchedulable strategy is hard to implement, however, it is the most interesting one. One of the reasons that we do not claim that we also provide this strategy, in spite of having the execution times of security components, is that the actual execution times in the synthesized system will not necessarily be the sum and individual addition of the execution times of the added security components to the rest of the system. More complex security implementation strategies can be considered but are not covered in this paper.

As for the synthesis of the code of the security components, in order to keep the approach generic, we intend to let certificate specification and other specific parameters of encryption algorithm to be filled in the generated code. One generator is associated for each algorithm. The suitability for timing analysis of the generated component code needs to

be planned but at least will allow for measurement based timing analysis as any other ProCom component. While the system functionality remains the same, the system needs also to react to authentication errors. This problem could be partially solved by allowing developers to add code to manage authentication errors in the generated code to define what must be the output data in each specific case.

Regarding the runtime adaptation mechanism, while on one hand, it may make the job of the attackers harder as not a fixed algorithm is used in each invocation and thus it will not be known and predictable to the attackers (hence some sort of "security through obscurity"), on the other hand, if attackers know the internal mechanism of the runtime adaptation, they can force some processing load on the system to make the system adopt the weakest algorithm possible, and that way, make it easier for themselves to break into the system. Moreover, the adaptation mechanism which was used as part of our general approach in this paper, can also be designed to act as an option; in the sense that it can be turned on and used when a processing load beyond a certain level is detected and turned off otherwise. This can help to mitigate the overhead of the adaptation mechanism itself (although another mechanism to monitor the processing load would need to be added in that case) and only use it when there are many requests for encryption.

7.8 Related Work

Designing security features for real-time embedded systems is a challenging task and requires appropriate methods and considerations. [16] and [25] particularly discuss the specific challenges of security in embedded systems and define it as a new dimension to be considered throughout the development process. Considering the unique challenges of security in embedded systems, [25] also emphasizes that new approaches to security are required to cover all aspects of embedded systems design from architecture to implementation. The methods that we introduced in this paper contribute towards this goal by applying different disciplines in the field of software engineering, such as model-driven development methods, to cope with the specific challenges of designing security for embedded systems.

Also as a non-functional requirement [7, 26], satisfying security requirements in a system has costs and implications in terms of impact on

other requirements such as performance, power consumption and so on. In [17], measurement and comparison of memory usage and energy consumption of several encryption algorithms on two specific wireless sensor network platforms have been done. Performance and timing comparisons of several encryption algorithms are offered in [18] where Pentium machines are used as the platform. The approaches we proposed in this paper, work by relying on the timing and performance comparison results of encryption algorithms in such studies.

While model-driven and component-based approaches serve as promising approaches to cope with the design complexity of real-time embedded systems, management of runtime data in these systems has also become an important issues than ever before due to the growing complexity of them. This fact becomes more clear when we realize that keeping track of all data that are passing through different parts of the system is an extremely hard task for a person. In addition, most design methods based on component models focus mainly on functional structuring of the system without considering semantics and meanings for data flows [20]. A data-centric approach for modeling data as well as using real-time databases for runtime data management in real-time embedded systems is proposed in [20]. In this work, however, non-functional (extra-functional) properties such as security are not addressed, and our approach presented in this paper basically follows a similar method for modeling data entities as a basis to define security specification.

As for modeling security aspects, there are several solutions such as UMLsec [12]. UMLsec is a UML profile [27] for the specification of security relevant information in UML diagrams. It is one of the major works in this area and comes with a tool suite which provides the possibility to evaluate security requirements and their violations. SecureUML [28] is also another UML profile for modeling of role-based access controls. UML profile for Modeling and Analysis of Real-time Embedded Systems (MARTE) [29] provides semantics for modeling non-functional properties and their analysis (e.g., schedulability). In [30], we have discussed MARTE and the benefits of extending MARTE with security annotations to better cover the modeling needs of embedded systems. Besides UMLsec and its tool suite which enables analysis of security requirements, in [31], a method for specifying security requirements on UML models and verifying their satisfaction by relating model-level requirements to code-level implementation is offered. In [32], we have provided a small example how it is possible to model security requirements along

with some other requirements of telecommunication systems and then perform model-based analysis using the analysis tool suite of UMLsec to identify possible violations of security requirements.

The need to identify sensitive data is also discussed in [33] where an extension to include security concerns as a separate model view for web-services based on Web-Services Business Process Execution Language (WS-BPEL) is offered. However, it does not take into account the consequences of security design decisions on timing aspects, while by identifying sensitive parts of messages which need to be secured, our objective is to ease the computation of the timing impacts of security implementations protecting those sensitive data. Considering the challenges of securing distributed systems [34] has done a survey on the application of security patterns, as a form of software design patterns, to secure distributed systems. Moreover, it discusses different methodologies that make use of these ad-hoc security patterns in a structured way. It also reports that the majority of the studied methodologies lack explicit support for distributed systems and special concerns that these systems have and mentions the development of tailored methodologies for different types of distributed systems as an important future work in this area. The approach that we suggested here could serve as an example for developing such methodologies in particular for distributed real-time and embedded systems in which timing requirements play a key role in the correctness of the whole system.

Regarding the adaptation method that we used as part of our suggested approach, there are also several related studies and approaches that we discuss them here. The study done in [35] is one of the interesting works in the area of security for real-time embedded systems which uses an adaptive method. In this work, the main focus is on a set of periodic tasks with known real-time parameters, whereas, our main target is complex systems that can consist of any type of real-time tasks. Also, while in our work, the security level of the system is considered implicitly through the selection of algorithms from the encryption algorithms table, in [35], a QoS value has been considered which explicitly represents the security level of the system. Moreover, in our work, it is the encryption algorithms which are adaptively replaced, while the main adaptation component in that work is the key length. Our approach can easily be extended to cover not only different encryption algorithms but also variations of each, including different combinations of key length, number of rounds and so on, as items (rows) in the encryption algo-

rithms table (e.g., AES256, AES128, etc.). Another interesting study with is close to our work is [36], which basically introduces a similar type of adaptation mechanism as ours. The main focus in that work is, however, on client-server scenarios using a database, and to manage the performance of transactions. The security manager component used int his work periodically adjusts the the security level of the system. In our approach, however, the adaptation mechanism is executed per request and is not active when there is no request for encryption. Moreover, it is possible in the approach introduced in this work that an inappropriate encryption method is used by a client, while security level change is occurring. To solve this situation, several acknowledgment messages are sent and the process is repeated to correct this issue. Therefore, it is possible that the security manager faces problems regarding synchronization and message loss due to out of order arrival of messages. As another approach for managing security in real-time systems, in [37], a secure-aware scheduler is introduced which basically incorporates and takes into account timing management of security mechanisms as part of its scheduling policy.

7.9 Conclusion and Future Work

In this paper, we introduced an approach to define security specifications in real-time embedded systems at a high level of abstraction based on the benefits of model-driven and component-based methods. Using the suggested approach we bring semantics to the data that are transferred in embedded systems to identify sensitive data. The approach enables also to derive automatically the security implementations and facilitates performing timing analysis including security features at early phases of development. It was also demonstrated how incorporating a runtime adaptation mechanism as part of the approach helps to mitigate the violations of timing constraints at runtime. As mentioned, such runtime adaptation mechanisms are especially useful for complex systems where performing static analysis may not be practical, as well as in cases where the assumptions that have been used for performing static analysis are prone to deviation and violation at runtime which can then lead to the invalidation of analysis results. Moreover, the introduced approach helps system designers to mainly focus on the system architecture and addressing timing properties, and at the same, including security concerns in

the design models without needing much expertise on how to implement security mechanisms. This again contributes to bringing security considerations in higher levels of abstraction.

One of the extensions of this work is to define and add more strategies for the designers to choose. Among the currently defined strategies, the StrongestSecuritySchedulable is the most interesting one but is hard to implement and will be part of our future works. One of the reasons that we do not claim that we also provide this strategy, in spite of having the execution times of security components, is that the actual execution times in the synthesized system will not necessarily be the sum and individual addition of the execution times of the added security components to the rest of the system. Also as another idea for the extension of this work, it would be interesting to define and assign required security strength to data and message channels as another factor that also affects the selection of security components. It should also be noted that in this work we mainly addressed encryption as a security mechanism. Considering other mechanisms such as authorization methods and their impacts on timing characteristics of systems is another interesting direction of this work to investigate. Also including other aspects than timing, such as power consumption of security mechanisms, performing trade-off, and establishing balance among them, similar to what we did here for timing properties, can be another extension of this paper and future work.

7.10 Acknowledgements

This work has been supported by Xdin Stockholm AB [38] and Swedish Knowledge Foundation (KKS) [39] through the ITS-EASY industrial research school program [40].

Bibliography

[1] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. On Generating Security Implementations from Models of Embedded Systems. In *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, Barcelona, Spain, October 2011.

[2] Mehrdad Saadatmand and Thomas Leveque. Modeling Security Aspects in Distributed Real-Time Component-Based Embedded Systems. In *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*, pages 437 –444, Las Vegas, USA, april 2012.

[3] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. Design of adaptive security mechanisms for real-time embedded systems. In *Proceedings of the 4th international conference on Engineering Secure Software and Systems*, ESSoS'12, pages 121–134, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] Premkumar T. Devanbu and Stuart Stubblebine. Software engineering for security: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 227–239, New York, NY, USA, 2000. ACM.

[5] Fu-Hau Hsu, Fanglu Guo, and Tzi-cker Chiueh. Scalable network-based buffer overflow attack detection. In *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, ANCS '06, pages 163–172, New York, NY, USA, 2006. ACM.

[6] Alec Main. Application Security: Building in Security during the Development Stage. *Journal of Information Systems Security*, 13(2):31–37, 2004.

[7] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Non-functional requirements: From elicitation to conceptual models. In *IEEE Transactions on Software Engineering*, volume 30, pages 328–350, 2004.

[8] Markus Voelter, Christian Salzmann, and Michael Kircher. Model Driven Software Development in the Context of Embedded Component Infrastructures. In Colin Atkinson, Christian Bunse, Hans-Gerhard Gross, and Christian Peper, editors, *Component-Based Software Development for Embedded Systems*, volume 3778 of *Lecture Notes in Computer Science*, pages 143–163. Springer Berlin / Heidelberg, 2005.

[9] Bran Selic. The Pragmatics of Model-Driven Development. *IEEE Software Journal*, 20:19–25, September 2003.

[10] M. T örngren, DeJiu Chen, and I. Crnkovic. Component-based vs. model-based development: a comparison in the context of vehicular embedded systems. In *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*, pages 432 – 440, aug.-3 sept. 2005.

[11] Ivica Crnkovic. Component-based Software Engineering - New Challenges in Software Development. In *Software Focus*, volume 2, pages 27–33, 2001.

[12] Bastian Best, Jan Jurjens, and Bashar Nuseibeh. Model-Based Security Engineering of Distributed Information Systems Using UMLsec. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 581–590, Washington, DC, USA, 2007. IEEE Computer Society.

[13] Anders Wall, Johan Andersson, Jonas Neander, Christer Norstr öm, and Martin Lembke. Introducing Temporal Analyzability Late in the Lifecycle of Complex Real-Time Systems. In *Real-Time and Embedded Computing Systems and Applications*, volume 2968 of *Lecture Notes in Computer Science*, pages 513–528. Springer Berlin Heidelberg, 2004.

[14] Sigrid Gürgens, Carsten Rudolph, Antonio Maña, and Simin Nadjm-Tehrani. Security engineering for embedded systems: the SecFutur vision. In *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems*, S&D4RCES '10, New York, NY, USA, 2010. ACM.

[15] Eirik Albrechtsen. Security vs Safety. NTNU - Norwegian University of Science and Technology `http://www.iot.ntnu.no/users/albrecht/rapporter/notat%20safety%20v%20security.pdf`, Accessed: December 2012.

[16] Paul Kocher, Ruby Lee, Gary McGraw, and Anand Raghunathan. Security as a new dimension in embedded system design. In *Proceedings of the 41st annual Design Automation Conference*, DAC '04, pages 753–760, 2004. Moderator-Ravi, Srivaths.

[17] Jongdeog Lee, Krasimira Kapitanova, and Sang H. Son. The price of security in wireless sensor networks. *Journal of Computer Networks*, 54:2967–2978, December 2010.

[18] A. Nadeem and M.Y. Javed. A Performance Comparison of Data Encryption Algorithms. In *First International Conference on Information and Communication Technologies, ICICT 2005.*, pages 84 – 89, 2005.

[19] Séverine Sentilles, Aneta Vulgarakis, Tomas Bures, Jan Carlson, and Ivica Crnkovic. A Component Model for Control-Intensive Distributed Embedded Systems. In Michel R.V. Chaudron and Clemens Szyperski, editors, *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*, pages 310–317. Springer Berlin, October 2008.

[20] Andreas Hjertström, Dag Nyström, and Mikael Sjödin. A data-entity approach for component-based real-time embedded systems development. In *Proceedings of the 14th IEEE international conference on Emerging technologies & factory automation*, ETFA'09, pages 170–177, Piscataway, NJ, USA, 2009. IEEE Press.

[21] CPU Killer. `http://www.cpukiller.com/`, Accessed: December 2012.

[22] Séverine Sentilles, Petr Štěpán, Jan Carlson, and Ivica Crnković. Integration of Extra-Functional Properties in Component Models. In *12th International Symposium on Component Based Software Engineering*. Springer, 2009.

[23] Ivan Kurtev. State of the Art of QVT: A Model Transformation Language Standard. In *Applications of Graph Transformations with Industrial Relevance*, volume 5088 of *Lecture Notes in Computer Science*, pages 377–393. Springer Berlin, 2008.

[24] Thomas Leveque, Jan Carlson, Séverine Sentilles, and Etienne Borde. Flexible Semantic-Preserving Flattening of Hierarchical Component Models. In *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE Computer Society, August 2011.

[25] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, 3:461–491, August 2004.

[26] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. Toward Model-Based Trade-off Analysis of Non-Functional Requirements. In *38th Euromicro Conference on Software Engineering and Advanced Applications(SEAA)*, September 2012.

[27] B. Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07. 10th IEEE International Symposium on*, pages 2 –9, may 2007.

[28] Torsten Lodderstedt, David A. Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 426–441, London, UK, 2002. Springer-Verlag.

[29] MARTE specification version 1.1. `http://www.omgmarte.org`, Accessed: December 2012.

[30] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. On the Need for Extending MARTE with Security Concepts. In *International Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011)*, March 2011.

[31] John Lloyd and Jan Jürjens. Security Analysis of a Biometric Authentication System Using UMLsec and JML. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, MODELS '09, pages 77–91, Berlin, Heidelberg, 2009. Springer-Verlag.

[32] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems. In *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, October 2011.

[33] Meiko Jensen and Sven Feja. A Security Modeling Approach for Web-Service-Based Business Processes. In *Proceedings of the 2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, ECBS '09, pages 340–347, Washington, DC, USA, 2009. IEEE Computer Society.

[34] Securing distributed systems using patterns: A survey. *Computers & Security Journal*, 31(5):681 – 703, 2012.

[35] Kyoung-Don Kang and Sang H. Son. Towards security and QoS optimization in real-time embedded systems. In *SIGBED Rev.*, volume 3, pages 29–34, New York, NY, USA, January 2006. ACM.

[36] Sang H. Son, Robert Zimmerman, and Jörgen Hansson. An adaptable security manager for real-time transactions. In *Proceedings of the 12th Euromicro conference on Real-time systems*, Euromicro-RTS'00, pages 63–70, Washington, DC, USA, 2000. IEEE Computer Society.

[37] T. Xie and X. Qin. Scheduling security-critical real-time applications on clusters. In *IEEE Transactions on Computers*, volume 55, pages 864 – 879, july 2006.

[38] Xdin AB. `http://xdin.com/`, Accessed: December 2012.

[39] KK-stiftelsen: Swedish Knowledge Foundation. http://www.kk-stiftelsen.org/SitePages/Startsida.aspx, Accessed: December 2012.

[40] ITS-EASY post graduate industrial research school for embedded software and systems. http://www.mrtc.mdh.se/projects/itseasy/, Accessed: December 2012.

Chapter 8

Paper C: Monitoring Capabilities of Schedulers in Model-Driven Development of Real-Time Systems

Mehrdad Saadatmand, Mikael Sjödin, Naveed Ul Mustafa
17th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA), Krakow, Poland, September, 2012.

Abstract

Model-driven development has the potential to reduce the design complexity of real-time embedded systems by increasing the abstraction level, enabling analysis at earlier phases of development, and automatic generation of code from the models. In this context, capabilities of schedulers as part of the underlying platform play an important role. They can affect the complexity of code generators and how the model is implemented on the platform. Also, the way a scheduler monitors the timing behaviors of tasks and schedules them can facilitate the extraction of runtime information. This information can then be used as feedback to the original model in order to identify parts of the model that may need to be re-designed and modified. This is especially important in order to achieve round-trip support for model-driven development of real-time systems. In this paper, we describe our work in providing such monitoring features by introducing a second layer scheduler on top of the OSE real-time operating system's scheduler. The goal is to extend the monitoring capabilities of the scheduler without modifying the kernel. The approach can also contribute to the predictability of applications by bringing more awareness to the scheduler about the type of real-time tasks (i.e., periodic, sporadic, and aperiodic) that are to be scheduled and the information that should be monitored and logged for each type.

8.1 Introduction

Model-Driven Development (MDD) is a promising approach to cope with the design complexity of real-time and embedded systems. It helps to raise the abstraction level and also perform analysis at earlier phases of development. Therefore, problems in the design of a system can be identified before the implementation phase [1].

Automatic code generation is also one of the end goals in model-driven development. In the context of real-time systems, this includes generating the implementation of periodic, sporadic and aperiodic tasks. However, most (industrial) Real-Time Operating Systems (RTOS) such as VxWorks, RTEMS, RT-Linux, Windows CE, OSE, and also others such as FreeRTOS only allow specification of priority for a real-time task. The definition of different types of real-time tasks (i.e., periodic, sporadic, and aperiodic) and specification of timing properties for them including period, deadline, Worst-Case Execution Time (WCET), etc. are not explicitly supported in these RTOSes. While in theory, a real-time task is simply specified by its timing parameters, in practice and when it comes to implementation, these parameters are introduced in the system in different ways. For example, a periodic task may be implemented in the form of an interrupt while its period is actually set by having a timer to trigger the interrupt periodically. For code generation, this means that for every model element defined as periodic, what is actually generated is an interrupt handler that *behaves* in a periodic way. The issue is that when we look at these systems at runtime, no tangible and single runnable entity as a real-time periodic task is actually observable and identifiable. In other words, the semantic mapping of a periodic task at model level and such an entity at runtime becomes weak.

Moreover, other parameters of a real-time task such as deadline are lost and not present at the implementation level or are defined in arbitrary and different ways in each implementation. This is because among all these parameters, what is usually supported explicitly by most real-time operating systems, is to specify only priority for a task. Other parameters are left to be defined and implemented by system designers in arbitrary ways, such as using timer interrupts and delays to enforce periodicity or Minimum Inter-Arrival Time (MIAT). The problem becomes even more evident when it comes to runtime monitoring of real-time systems and where a system needs to detect events such as deadline misses,

execution time overruns, etc.

To cope with these problems, we propose a second layer scheduler which takes as input the specification and implementation of real-time tasks including all of their temporal parameters, and schedules and executes them using the underlying scheduler of the operating system. With this design, the code generators can then generate tangible real-time tasks according to a well-defined specification (e.g., definition of a task as: task(task type, period, deadline, execution time)), regardless of whether, for instance, they are going to be actually implemented as a timer interrupt or some other mechanism. This way, an actual real-time task along with its parameters will be present and identifiable at the code level. The system can then be easily queried, for example, for the number of periodic or sporadic tasks and their specified timing parameters such as deadline. The top-level scheduler schedules a task and uses its parameters to manage and report events such as deadline misses or execution time overruns. In this approach, even if the underlying platform changes and the implementation of real-time tasks (e.g., as timer interrupt) are modified, it will not require any changes in the code generators and also the specification of real-time tasks. This may also improve the portability of the generated code and especially transformation engines, making them as *platform-independent* [2] as possible.

The issue is that for most commercial or closed-source RTOSes, it may not be possible or economical to modify the kernel and scheduler to add the above mentioned features. In such scenarios these features can be provided through an added layer on top of the core scheduler. In this work, by highlighting the role of schedulers in model-driven development of real-time systems and to provide round-trip support, we describe the suggested second layer scheduler built for OSE real-time operating system [3] and demonstrate how it improves the monitoring of the timing behaviors of tasks at runtime and detecting events such as deadline misses which are critical in real-time systems.

The remainder of the paper is as follows. In Section 8.2, we discuss the background context and motivation of the work. Section 8.3 describes the proposed approach along with its design and implementation details. In Section 8.4, an example is demonstrated and the implementation and behavior of the suggested approach is evaluated. In Section 8.5 we have a look at the related work, and finally in Section 8.6, we summarize the work and describe its future extensions and directions.

8.2 Background and Motivation

8.2.1 CHESS Project

This work has been done in the context of CHESS European project [4]. This project is about model-driven and component based development of real-time embedded systems for telecommunication, space, railway, and automotive domains which focuses on preservation and guarantee of extra-functional properties [5]. This is done by performing static analysis on design models and monitoring the behavior of the generated system at runtime. The idea is to back-annotate monitored results back to the model to inform the designer which modeled features have led to the violation of specified requirements and may need to be modified. The general structure of the approach is shown in Figure 8.1.

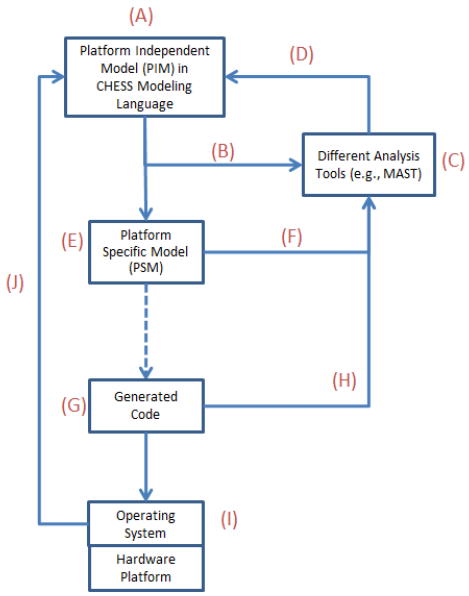

Figure 8.1: CHESS methodology

As can be seen from the above figure, different types of analysis are done at different abstraction levels (marked as B, F, and H), and the

results are propagated back to the model (D). MAST [6] is one of the analysis tools that is used in CHESS to perform schedulability analysis on the model. The system model which is defined in the modeling language called CHESS ML, is transformed into an appropriate model as input for the MAST analysis tool.

To ensure that the assumptions based on which the analyses were done hold true, the system's execution is monitored and runtime information are collected. For example, the difference between the characteristics of the (ideal) execution environment of the system taken into account for analysis and the actual one when the system is implemented may lead to the violation of the assumptions that are used to perform analysis [7]. Therefore, monitoring the behavior of the system at runtime is important in preservation of system properties.

Timing properties are of utmost importance in real-time embedded systems. In order to extract and collect information about runtime timing behaviors for back-annotation to the model, the platform should be able to provide this information. Among the most important timing data for back-annotation that are of interest are deadline misses, actual response time, and execution time overruns. However, such a feature is not present explicitly in many commercial real-time operating systems today and needs to be implemented in different ways by system developers. In the scope of this work, we narrow our focus on Part I of Figure 8.1.

8.2.2 OSE Real-Time Operating System

OSE is a real-time operating system developed by Enea [8]. It has been designed from the ground up for use in fault-tolerant distributed systems that are commonly found in telecommunication domain, ranging from mobile phones to radio base stations and is embedded in millions of devices around the world [3]. It provides preemptive priority-based scheduling of tasks. OSE offers the concept of direct and asynchronous message passing for communication and synchronization between tasks, and OSE's natural programming model is based on this concept. Linx, which is the Interprocess Communication Protocol (IPC) in OSE, allows tasks to run on different processors or cores, utilizing the same message-based communication model as on a single processor. This programing model provides the advantage of not needing to use shared memory among tasks.

The runnable real-time entity equivalent to a task is called *process* in OSE, and the messages that are passed between processes are referred to as *signals* (thus, the terms process and task in this paper can be considered interchangeable). Processes can be created statically at system start-up, or dynamically at runtime by other processes. Static processes last for the whole life time of the system and cannot be terminated. Types of processes that can be created in OSE are: interrupt process, prioritized process, background process, and phantom process. One interesting feature of OSE is that the same programming model is used regardless of the type of process. Of the timing properties that we have been discussing so far, only priority can be assigned for prioritized processes. Periodic behavior can be implemented by using timer interrupt processes. Information such as task completion time, deadline misses and such, is not reported by default and it needs to be implemented using system level APIs for events such as process swap_in and swap_out which are triggered when a process starts and stops running (i.e., context switches).

8.2.3 Goal

Figure 8.2 shows the target system that is built in the scope of this work. As discussed before, most real-time operating systems, such as OSE, only allow specification of priority for real-time tasks. In other words, semantically, there is no parameter or kernel level value that represents deadline, execution time, or period of a task. Similarly, the monitoring information and logs that are generated by the system do not contain information about deadline misses, or execution time overruns, because these concepts are not actually understood by the kernel and have no meaning for it. Therefore, it is the job of a programmer to implement such features and collect information by also implementing event handlers that monitor when a task gets CPU time and when it is preempted, and then calculate deadline misses or execution time overruns through this information.

The proposed solution that is shown in Figure 8.2 solves this situation by introducing a second layer scheduler. The interface to this scheduler layer representing the specification of a real-time task is in the form of: *Task(release time, period/MIAT, execution time, relative deadline, task type)*. Considering these parameters, the second layer scheduler then schedules the tasks using the priority-based scheduling mechanism of

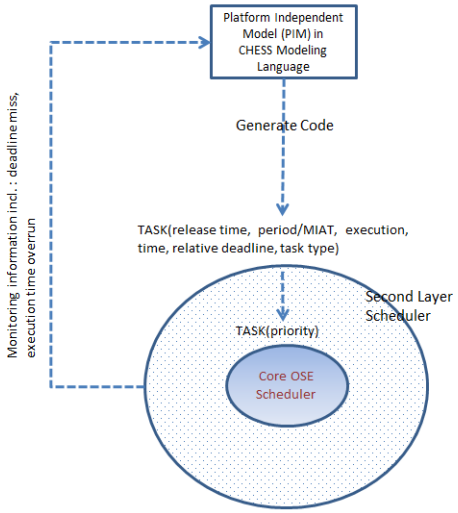

Figure 8.2: Interfaces between different parts in the suggested approach

the core scheduler.

The code generator on the other hand, generates real-time tasks (from the model) according to this specification, and need not care how such a task is actually implemented on the core scheduler, hence more portability and re-usability of the analyzed model and generated code are obtained.

From the monitoring perspective, since the second layer scheduler is responsible for scheduling tasks according to the described specification, it is aware of concepts such as deadline, and can therefore, produce log information representing events such as deadline misses. This generated log information from the second layer scheduler can then be used to propagate necessary information back to the model.

We believe that the suggested added layer in our approach can also help with decreasing the gap between theoretical aspects of real-time systems and their actual implementations by providing more semantics to parameters and specifications of real-time tasks at implementation level and thus increasing the applicability of theoretical knowledge such

as schedulability analysis techniques.

8.3 Scheduler Design and Implementation

In this section, the internal mechanism and design details of the second layer scheduler are described.

The second layer scheduler developed on top of OSE, schedules a given set of tasks (S) by releasing tasks to OSE core scheduler according to a selected scheduling policy. S can contain three kinds of tasks: Periodic, Sporadic, and Aperiodic tasks. Task parameters such as period and execution time are generated for the second layer scheduler from the model as input parameter files with .prm extension.

As shown in Figure 8.3, the system consists of a few other components besides the second layer scheduler process. At system startup, first process creator is started. Process creator creates an OSE process for each of the tasks that are specified as a set of input files. Initially, all these processes will be in the waiting mode (to receive a signal from the second layer scheduler process). From this point on, the second layer scheduler process, which has the highest priority in the system, controls the system. Based on a specified scheduling policy (e.g., EDF), the second layer scheduler selects an appropriate task from the queue of waiting tasks, and sends a *start* signal to it. It then enters a waiting state itself using OSE *receive_w_tmo* (receive with timeout) system call. This system call makes the caller process wait until it either receives a signal or the specified timeout expires. We make a specific use of this system call in our design by setting its timeout value equal to the time interval available before arrival of a new instance of a higher priority periodic task. Also, whenever a task finishes execution, it sends a completion signal back to the second layer scheduler process. Therefore, if the running task finishes its job before arrival of the next instance of a higher priority periodic task, the second layer scheduler will receive a completion signal (at the *receive_w_tmo* system call), and continues its job (scheduling next tasks). Otherwise, if the running task takes too much time, the timeout which is set in the *receive_w_tmo* command in the second layer scheduler will expire, and since the second layer scheduler process has the highest priority in the system, it preempts the running task, takes the CPU, checks the list of waiting tasks again, and selects the next appropriate task to run.

Right after receiving the completion signal by the second layer scheduler process, it generates log information about the behavior of the task which has just completed. Since the second layer scheduler has access to (and thus is aware of) all the real-time parameters of each task (e.g., periodic/MIAT, deadline, execution time), it can gracefully detect deadline misses, execution time overruns, and events of this kind, mark them in the log information and report them. This way, all this critical log information about the behavior of the system are also centralized, which can then be easily queried. This is an important feature which is absent in many real-time operating systems today.

Creation of monitoring log files and persistence of the collected information are done by the monitor process using the information that is sent to it by the second layer scheduler process in the form of signals. In this design, two separate log files are actually created: scheduling log file, and monitoring log file. Scheduling log file contains listing of schedules generated by the second layer scheduler by stating the time points at which a task in the task set is scheduled, completed or preempted. This log file is generated by the second layer scheduler. Events related to task deadlines can be investigated by examining the monitoring log file generated by the scheduler. Monitoring log file is updated with new information only when an instance of a task is completed, and scheduling log file is updated whenever a task is scheduled, preempted, resumed or completed.

The scheduling policy that the second layer scheduler uses for periodic tasks is selectable and not fixed. Same is the case with the scheduling mechanism for aperiodic and sporadic tasks. The selected policy is read as a configuration value at system startup. This makes the suggested approach flexible. Currently Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF) policies are supported for periodic tasks while aperiodic and sporadic tasks can be scheduled using background [9] or polling server [9, 10] schemes. Other policies can also be added to design.

In the following sections, the role of different components in our design are described in detail.

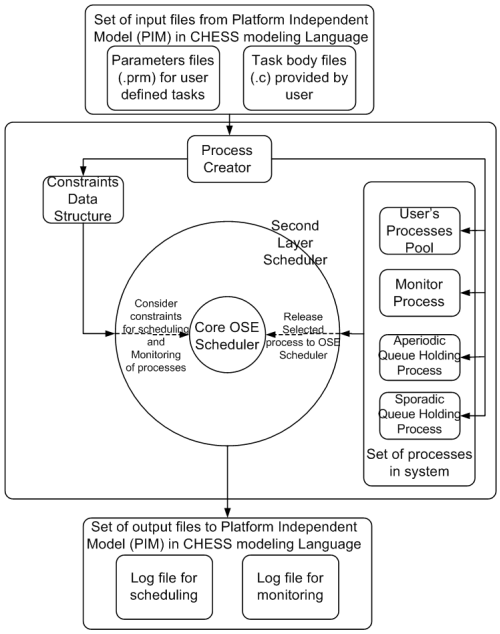

Figure 8.3: Components of Design

8.3.1 System Components

Process Creator

Each task in a task set is specified by two files.

- **Parameter File:** A file with .prm extension provides task parameters including release time, period/MIAT, execution time, relative deadline and type of the task.

  Type of task can have four valid values: 0 for Periodic, 1 for Sporadic, 2 for Aperiodic, and 3 for Polling Server.

- **Body File:** A file with .c extension contains the body of the task. In other words, .prm file of a task contains its timing non-functional specification while .c file contains its functional implementation.

Process creator reads the parameters for each task from its .prm file into a data structure, called ”constraints", and creates a prioritized OSE process against each user defined task. Moreover, It also creates following four OSE prioritized processes:

- Second layer scheduler process
- Sporadic queue holding process
- Aperiodic queue holding process
- Monitor process

None of the created OSE processes is started by process creator except the second layer scheduler process. Task parameters and Process Identifiers (PIDs) for all processes are then passed to the second layer scheduler in the form of “constraints" data structure.

Second Layer Scheduler

After receiving “constraints" structure and PIDs of all OSE processes created by process creator, the second layer scheduler schedules the tasks by releasing them to core OSE scheduler according to selected scheduling algorithm. The design provides options to select between RMS or EDF algorithm.

To schedule sporadic and aperiodic tasks, the second layer scheduler supports background scheduling and polling server for scheduling sporadic and aperiodic tasks.

Sporadic Queue Holder

Task set can contain periodic, aperiodic and sporadic tasks. Sporadic queue holder is a prioritized OSE process which maintains a list of sporadic tasks waiting for scheduling, by using a queue. Each element of queue contains two parameters for a sporadic task: PID corresponding to the given task, and release time of sporadic task.

To release a sporadic task, its PID and release time is to be placed in queue. An interrupt process (in case of hardware driven sporadic tasks) or a prioritized process (in case of software driven sporadic tasks) may initiate this placement by sending a signal to sporadic queue holder.

Upon receiving this signal, sporadic queue holder extracts the PID

and release time of sporadic task from signal and updates the queue with extracted information.

The second layer scheduler checks if there is a sporadic task to be scheduled by making a query to sporadic queue holder process.

Aperiodic Queue Holder

Aperiodic queue holder has the same structure and mechanism as the sporadic queue holder, except that it maintains a list of aperiodic tasks. Also separate signals are defined for use with aperiodic and sporadic queue holders.

Monitor

Monitor process generates a log file to state whether specified timing constraints for each task in task set S are met or not. For example, if the specified MIAT parameter, in case of a sporadic task, is violated then monitor records this violation in a monitoring log file.
When a task is released to the core OSE scheduler for execution, the second layer scheduler observes its timing parameters. As soon as a task completes its execution, the second layer scheduler sends a signal to the monitor process. This signal contains start time, completion time, desired deadline, desired execution time, desired MIAT, actual execution time, and actual MIAT of completed task. These timing values are extracted from .prm file of the task under monitoring (i.e., desired deadline and execution time) and measured by the second layer scheduler (i.e., actual deadline and execution time). Monitor extracts these timing values from the received signal and saves the relevant monitoring statements in a monitoring log file.

8.3.2 Signals and Communications

To achieve the scheduling of tasks in a reliable way, several signals are defined and used by system. These signals play two important roles: carry required data from one component to another, and ensure synchronous execution of all components.

These signals are described below.

- **start_exe_sig:** Start execution signal. This signal is sent by the second layer scheduler to a process to be scheduled on core OSE scheduler. Target process can start execution only if it has received start_exe_sig signal.

- **comp_sig:** Completion signal. This signal is sent as an acknowledgment to the second layer scheduler upon completion by a process which is created against a user defined task.

- **aper_update_sig:** Update signal for aperiodic queue holder. To release an aperiodic task, an interrupt process or prioritized process sends the aper_update_sig signal to aperiodic queue holder process. This signal contains the PID and release time of an aperiodic task to be scheduled.
  This signal is also used as a response to the second layer scheduler by aperiodic queue holder on receiving start_exe_sig from scheduler.

- **spor_update_sig:** Update signal for sporadic queue holder. To release a sporadic task, an interrupt process or prioritized process sends the spor_update_sig signal to sporadic queue holder process. This signal contains the PID and release time of sporadic task to be scheduled.
  This signal is also used as a response to the second layer scheduler by sporadic queue holder on receiving start_exe_sig from scheduler.

- **qupdate_confirm_sig:** Queue update confirmation signal. This confirmation signal is sent back to the sender of an update signal, after receiving aper_update_sig (in case of aperiodic queue holder) or spor_update_sig (in case of sporadic queue holder). Confirmation signal informs sender if queue is updated successfully. In case of failure, sender can send aper_update_sig again with same content after waiting for a finite amount of time.

- **monitor_info_sig:** Monitoring information signal. This signal is sent by the second layer scheduler to the monitor, every time a task completes its execution. Monitor uses the information contained in this signal to determine if a completed task has met its constraints, such as deadline and WCET.

8.3.3 Priority Assignment

In OSE, there are 32 priority levels. Priority 0 is considered the highest while 31 is considered as the lowest priority level. In our system, process creator creates one OSE process for each task in the input task set. All such processes are assigned priority level of 1. Similarly, sporadic queue holder process and aperiodic queue holder process have priority level of 1. However, the second layer scheduler process has priority level of 0 which is the highest possible priority level. The reason for assigning priority level 0 to the scheduler is to make it non preemptable by any other prioritized OSE process.

Monitor behaves as a background OSE process and hence has the lowest priority level. This ensures that monitoring is performed only when no task is ready and the scheduler is idle. This reduces the effect of monitoring on the scheduling of tasks.

8.3.4 Scheduling of Tasks

As described in the previous section, all OSE processes are created by Process creator but not started by it. Process creator starts only the second layer scheduler process and passes “constraints" structure along with PIDs of all OSE processes.

Scheduling of Periodic Tasks

The second layer scheduler examines the “type" parameter of all tasks to identify periodic tasks among the task set. Tasks are scheduled by releasing them to core OSE scheduler according to specified scheduling algorithm, for example RMS.

The second layer scheduler sends start_exe_sig to the the process representing the user defined task which has highest priority according to selected scheduling algorithm. Then scheduler waits for receiving comp_sig back from target OSE process but with a finite waiting time called “timeout".

If comp_sig is received before the timeout is expired, it implies that the target process has completed. Hence the second layer scheduler releases to the core OSE scheduler the next ready OSE process representing the user defined periodic task. If comp_sig is not received within the timeout duration and a process representing a user defined task with higher priority is ready, then the former process is preempted and second

layer scheduler releases to core OSE scheduler the process with higher priority.

Scheduling of Sporadic and Aperiodic Tasks

To schedule a sporadic task, it is necessary that its corresponding PID and release time are placed in the sporadic processes queue maintained by sporadic queue holder. This can be achieved by sending a spor_update_sig signal to the sporadic queue holder containing release time and PID of the OSE process corresponding to the task. Signal, spor_update_sig, can be sent to the sporadic queue holder either by an interrupt OSE process or a prioritized OSE process. In the first case, target sporadic task becomes interrupt driven while in second case it behaves as a program driven sporadic task. The above discussion is valid also for achieving interrupt and program driven behavior for aperiodic tasks.

Sporadic and aperiodic tasks can be scheduled by using one of following two approaches:

- Background Scheme: One approach to schedule sporadic and aperiodic tasks is to use time slots in which no periodic task is ready to run. In such case, the second layer scheduler first makes query to sporadic queue holder by sending start_exe_sig to find if there is any ready sporadic task. The spor_update_sig signal is sent back by sporadic queue holder to the second layer scheduler, indicating availability status of sporadic task.
  If there is a ready sporadic task, the second layer scheduler releases sporadic task to OSE core scheduler and waits until either it completes its execution or a periodic task becomes ready. If sporadic task is completed and no periodic task is ready to run, second layer scheduler again makes query to sporadic queue holder to find if there are any more sporadic tasks waiting in the queue.
  If sporadic task queue is empty and no periodic task is ready to run, the second layer scheduler makes query to aperiodic queue holder by sending start_exe_sig. Availability status of aperiodic task is communicated back to the second layer scheduler by sending aper_update_sig from aperiodic queue holder. If aperiodic queue is not empty and aperiodic task at the head of the queue is ready to run, the second layer scheduler releases that aperiodic task to core OSE scheduler.

If there is no periodic, sporadic and aperiodic task to execute, Monitor process is released to core OSE scheduler by the second layer scheduler.

- Polling Server Scheme: An alternative approach to schedule sporadic and aperiodic tasks is to use polling server. Polling server is a periodic task like any other periodic task. It has a period P_s and execution time E_s. Execution time of polling server is known as its budget.
  Polling server is scheduled along with all other periodic tasks according to selected scheduling algorithm. However, when polling server gets the chance to execute, the second layer scheduler makes query to sporadic and aperiodic queue holding processes to find if there is any ready sporadic or aperiodic task. If sporadic or aperiodic task is ready to run, the second layer scheduler releases that task to OSE core scheduler and budget of polling server keeps declining per unit time.
  If sporadic or aperiodic task completes its execution before budget is expired, the second layer scheduler picks next ready sporadic or aperiodic task to release to OSE core scheduler. This sequence continues until either there is no sporadic or aperiodic task or budget of server is expired or a higher priority periodic task becomes ready to execute.
  At the start of each period of the polling server, its budget is set equal to its execution time. If at that time point, no sporadic or aperiodic task is ready to run then budget immediately declines to zero. Otherwise the budget decreases one level per time unit.

8.3.5 Monitoring of Tasks

On completion of a task, independent of its type, the second layer scheduler sends monitor_info_sig signal to Monitor. Monitor is implemented as a background OSE process. Hence, it can execute only when there is no periodic, sporadic or aperiodic task ready to run. Monitor continuously checks its input message queue for monitor_info_sig signal. This message carries following information to the monitor process regarding completed task:

- start time of the task

- completion time of the task
- specified deadline parameter for the task
- specified MIAT parameter for the task
- specified execution time for the task
- actual execution time for task
- actual deadline for task
- actual MIAT for the task

Monitor uses this information to make decision if a completed task has met its parameters or violated them. In any case, monitor records the information in a monitoring log file.

Operation of the scheduler is summarized by the sequence diagram of Figure 8.4. In this diagram, the task set consists of two periodic tasks T_1 and T_2, one sporadic task T_3 and an aperiodic task T_4. Timing parameters of the tasks defined in this task set are listed below using the specification convention: Task (Release Time, Period, WCET, Relative deadline, Task type).

Periodic task: $T_1(0, 12, 3, 8, 0)$
Periodic task: $T_2(0, 4, 1, 3, 0)$
Sporadic task: $T_3(0, 15, 2, 6, 1)$
Aperiodic task: $T_4(0, 0, 1, 6, 2)$

This figure shows a valid sequence of execution when RMS is used as the scheduling policy for periodic tasks, while sporadic and aperiodic tasks are scheduled using background scheme. As is evident from the sequence diagram, monitor gets the chance to execute only when no other process is in ready state.

8.4 Experiment and Monitoring Results

The described approach has been implemented and tested on OSE Soft-Kernel (SFK) version 5.5.1 [8]. In this section, we show an example of a task set which is implemented based on the specification of the proposed second layer scheduler. The way the task set is scheduled and the log

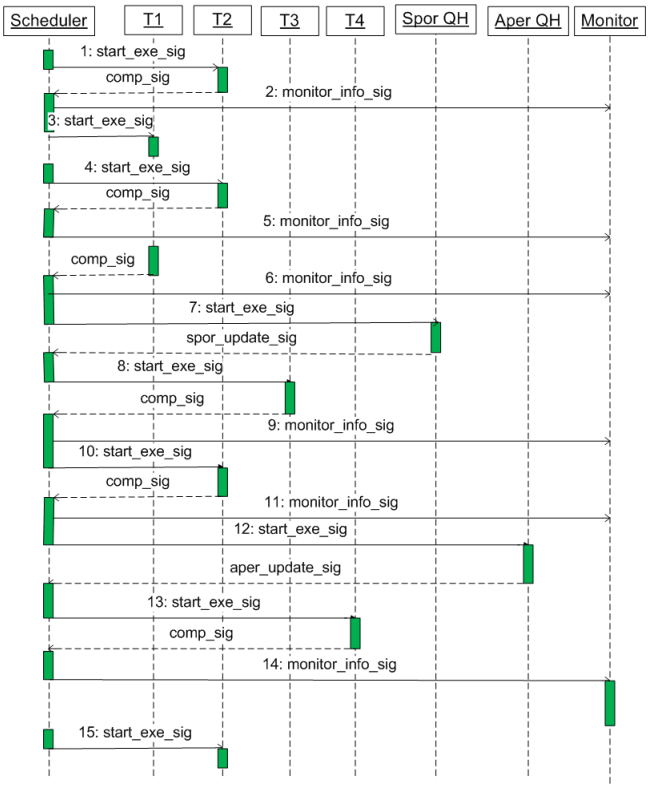

Figure 8.4: Sequence diagram to demonstrate operation of the scheduler

information that is generated for the behaviour of tasks are illustrated.

A task set consisting of four tasks is created. Periodic tasks in the task set are configured to be scheduled using RMS while aperiodic and sporadic tasks are to be scheduled using polling server scheme (this can be changed by changing configuration parameters).

Timing constraints for tasks are given below (last parameter identifies the type of task):

- $T_1(0, 10, 2, 5, 3)$: Polling server with release time=0, period=10, WCET/Budget=2, Relative Deadline=5, Task type=3.
- $T_2(0, 5, 2, 4, 0)$: Periodic Task with release time=0, period=5, WCET=2,

Relative Deadline=4, Task type=0.

- $T_3(0, 5, 2, 4, 1)$: Sporadic Task with release time=0, MIAT=5, WCET=2, Relative Deadline=4, Task type=1. Two instances of Sporadic task T_3 are released at time 0;
- $T_4(0, 0, 2, 7, 2)$: Aperiodic Task with release time=0, period= 0 (Not Applicable), WCET=2, Relative Deadline=7, Task type=2.

As mentioned before, these timing parameters are actually specified in the .prm file of each task (i.e., t1.prm, ..., t4.prm). Process creator opens these files and populates "constraints" data structure with these data.

Using the implemented second layer scheduler to schedule this task set, the following log files are automatically generated:

- **Scheduling log file:** Scheduling log file provides the time points for each task at which it is scheduled, preempted/not completed, resumed or completed. Parts of the scheduling log information generated for the task set are shown in Listing 8.1. To make it easier to follow and understand the information in the log file, the PIDs that are assigned to each task by the system are also mentioned below:
  - PID of process representing T_1 = 65595.
  - PID of process representing T_2 = 65596.
  - PID of process representing T_3 = 65597.
  - PID of process representing T_4 = 65598.

Listing 8.1: Scheduling log file

```
task PID=65596
Scheduled for 5 ticks at ticks=1115
task PID=65596
Completed at ticks=1117
task PID=65597
Scheduled with budget= 2 ticks at ticks=1117
task PID=65597
Not completed at ticks=1119
Remaining Execution Time in ticks=1
task PID=65596
Scheduled for 5 ticks at ticks=1120
task PID=65596
Completed at ticks=1122
task PID=65596
```

```
Scheduled for 5 ticks at ticks=1125
task PID=65596
Completed at ticks=1127
task PID=65597
Resumed with budget= 2 ticks at ticks=1127
task PID=65597
Completed at ticks=1128
task PID=65597
Scheduled with budget= 1 ticks at ticks=1128
task PID=65597
Not completed at ticks=1129
Remaining Execution Time in ticks=1
task PID=65596
Scheduled for 5 ticks at ticks=1130
task PID=65596
Completed at ticks=1132
task PID=65596
Scheduled for 5 ticks at ticks=1135
task PID=65596
Completed at ticks=1137
task PID=65597
Resumed with budget= 2 ticks at ticks=1137
task PID=65597
Completed at ticks=1138
task PID=65598
Scheduled with budget= 1 ticks at ticks=1138
task PID=65598
Not completed at ticks=1139
Remaining Execution Time in ticks=1
task PID=65596
Scheduled for 5 ticks at ticks=1140
task PID=65596
Completed at ticks=1142
```

- **Monitoring log file:** On completion of each instance of a task, monitoring log file lists type of task, PID of process representing that task in system, start time of the task, specified deadline, completion time of the task, specified WCET for the task, actual execution time consumed by the task, response time of the task, specified MIAT/period and actual interval between two consecutive invocations of the task. Listing 8.2 shows parts of the monitoring log information generated for the task set.

Listing 8.2: Monitoring log file

```
PID =65596
Type of task =0
start time in ticks=1115
specified deadline in ticks=1119
completion time in ticks =1117
specified WCET in ticks=2
actual execution time in ticks=2
Response time in ticks =2
```

```
specified Period/MIAT in ticks =5
Interval between two consecutive invocations in ticks=5
PID =65596
Type of task =0
start time in ticks=1120
specified deadline in ticks=1124
completion time in ticks =1122
specified WCET in ticks=2
actual execution time in ticks=2
Response time in ticks =2
specified Period/MIAT in ticks =5
Interval between two consecutive invocations in ticks=5
PID =65596
Type of task =0
start time in ticks=1125
specified deadline in ticks=1129
completion time in ticks =1127
specified WCET in ticks=2
actual execution time in ticks=2
Response time in ticks =2
specified Period/MIAT in ticks =5
Interval between two consecutive invocations in ticks=5
PID =65597
Type of task =1
start time in ticks=1117
specified deadline in ticks=1121
completion time in ticks =1128
specified WCET in ticks=2
actual execution time in ticks=3
Response time in ticks =11
specified Period/MIAT in ticks =5
Interval between two consecutive invocations in ticks=10
PID =65596
Type of task =0
start time in ticks=1130
specified deadline in ticks=1134
completion time in ticks =1132
specified WCET in ticks=2
actual execution time in ticks=2
Response time in ticks =2
specified Period/MIAT in ticks =5
Interval between two consecutive invocations in ticks=5
PID =65596
Type of task =0
start time in ticks=1135
specified deadline in ticks=1139
completion time in ticks =1137
specified WCET in ticks=2
actual execution time in ticks=2
Response time in ticks =2
specified Period/MIAT in ticks =5
Interval between two consecutive invocations in ticks=5
PID =65597
Type of task =1
start time in ticks=1128
specified deadline in ticks=1132
completion time in ticks =1138
specified WCET in ticks=2
```

```
actual execution time in ticks=2
Response time in ticks =10
specified Period/MIAT in ticks =5
Interval between two consecutive invocations in ticks=9
```

The first four lines in the generated scheduling log information shown in Listing 8.1 indicates that the periodic task T_2 with PID of 65596, which is started at tick time 1115, has completed at 1117. The next task which is scheduled is T_3 with PID of 65597. The polling server has the capacity of two time unit at this time instance, therefore, the sporadic task T_3 can run until tick time 1119, and at 1120 another instance of T_2 arrives which causes the second layer scheduler to preempt T_3. However, at 1119, T_3 has not managed to complete its job, and therefore, it is marked as 'Not completed'.

On the other hand, the monitoring information in Listing 8.2, among other things, can be used to check whether any deadline miss has occurred or not. For example, it shows that the first two instances of T_2 (PID=65596) have met their deadlines. The first instance has finished its job at 1117 and finished before its deadline which is 1119. The deadline for the second instance is at 1124, and it has managed to complete its job at 1122, and therefore, meet its deadline. However, the deadline of the sporadic task T_3, with PID of 65597, has been 1121 while it has managed to finish its job at 1128. Its actual execution time has also been three time units which is one time unit more than its specified WCET. This shows that there has been execution time overrun for this task and there is something wrong with the specified WCET value of it, and it needs to be re-considered. Such information are hardly provided by default in any real-time operating system.

Figure 8.5 visualizes the schedule generated by the scheduler. This figure is created (manually) using the information available in the scheduling log file generated by the scheduler. The scheduling log file shows that the first task is released at 1115 system ticks. To make this schedule easier to understand in the figure, subtraction of 1115 ticks is performed at every time point.

As is indicated by a cloud symbol in Figure 8.5, actual execution time consumed by first instance of sporadic task is 3 ticks instead of 2 ticks as specified in timing constraint of WCET=2. Therefore, it misses its deadline of 5 ticks and is completed at 13 ticks. Second instance of sporadic task is scheduled immediately after completion of the first instance. This is because MIAT of 10 ticks is already elapsed (10+2=12).

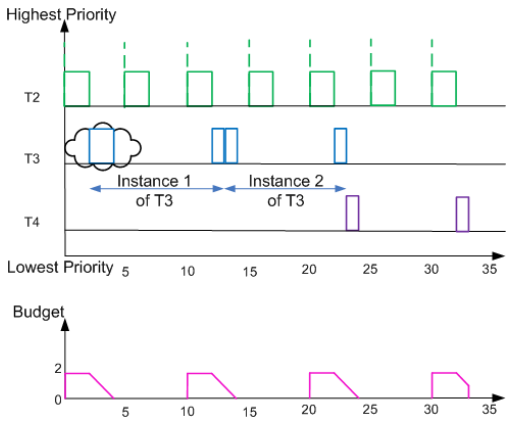

Figure 8.5: Schedule generated by the design using second layer scheduler

The diagram in the lower part of Figure 8.5 indicates the replenishment and decrease of budget with passage of time as is defined for the behavior of polling servers.

Now that the necessary information about the runtime behavior of tasks is provided in the log files generated by the system, a user can easily query them, extract desired parts, and draw conclusions. For example, it is very easy to find out the number of deadline misses, execution time overruns, the task with maximum number of deadline misses, etc. by using the log files as the data source. Similarly, at any time point, the number of periodic, sporadic, and aperiodic tasks in the system can easily be requested from the second layer scheduler; a simple but important feature which is not provided by default in many RTOSes today. Also, it is now possible to identify and report the time period during which maximum number of deadline misses have occurred, and examine as well how the system has been behaving in terms of context switches and preemptions during that period. These are features whose implementations can be very hard and complex without having the necessary monitoring information and using the suggested approach.

8.5 Related Work

Many of the operating systems and also programming languages today provide support for measuring the CPU time a runnable entity (i.e., thread, etc.) consumes to perform its function. However, the monitoring facilities and event handling mechanisms provided by these platforms are not usually integrated with their scheduling facilities [11]. As a result, the platform cannot enforce and monitor real-time properties of threads such as their allowed execution times and deadlines. Real-Time Specification for Java (RTSJ) [11] is introduced to integrate scheduling of threads with the execution time monitoring facilities and enforce execution budgets on them in Java. For Ada applications and particularly the Ada Ravenscar profile [12] different kernels such as ORK [13] have been introduced to enforce and manage task budgets and handle critical real-time events. Moreover, the Ada compiler, GNAT, has defined GNARL as the tasking runtime system of the compiler which is divided into two layers. The lower layer of GNARL abstracts the execution platform and provides OS services via POSIX interfaces.

The implementation of different scheduling algorithms on top of a fixed priority scheduler has also been used in the FIRST project [14]. The main objective in this project has been to develop a scheduling framework for real-time applications that have various types of tasks (hard, soft, firm, etc.) and scheduling paradigms within the same system to achieve a flexible integrated real-time system. The FRESCOR project [15] which also relies on the results of the FIRST project, targets the gap between the real-time theory and the industrial and practical aspects of real-time systems. In this project a platform independent scheduler called FRSH has been developed which provides a set of APIs for the applications. FRSH implementation is made portable to different operating systems using the FRSH Operating System Adaptation layer (FOSA) which encapsulates all native operating system calls and types used by FRSH into neutral names acceptable by both POSIX and non-POSIX compliant systems [15]. The main objective of this project has been to provide a contract-based model and framework for real-time embedded systems.

The implementation of a new scheduling class called *SCHED_DEADLINE* for the Linux kernel that adds EDF scheduling policy support to Linux is done in[16]. It is also motivated by acknowledging the fact that due to limited support for specifying timing constraints for real-time

tasks (e.g., deadline) and lack of control over them, feasibility study of the system under development and guaranteeing the timing requirements of tasks are not possible. There are however several differences between this work and ours. It focuses only on mechanisms for adding EDF scheduling policy to the Linux kernel, while we target the problem in a more general manner and allowing the scheduling policy to be configurable. Our focus is mainly on improving the monitoring of real-time events by providing more control over real-time tasks and providing more knowledge about their timing constraints to the scheduler regardless of the scheduling policy and without modifying the core scheduler. Moreover, we try to provide an abstraction layer around the core scheduler to hide platform-dependent implementation details from the user while SCHED_DEADLINE tries to solve a different problem and is basically added as a separate module to the system.

One concept which also introduces different levels of abstraction around a core scheduler is Hierarchical Scheduling Framework (HSF) [17, 10]. There are fundamental differences between what we introduced here and HSF. HSF is a modular approach in which a system is divided into several subsystems. The subsystems are scheduled by a global (core) scheduler, while the tasks in each subsystem are scheduled by local (subsystem) level schedulers. The structure that we introduced here does not try to divide a system into different subsystems where each of these subsystems may be scheduled differently by a different scheduler.

There are also studies that focus on execution monitoring of real-time systems. Many of these studies, such as [18], try to predict timing violations in the system in different ways, for example, using statistical models. Our suggested approach does not to try to predict violations and produces precise monitoring information for behavior of real-time tasks and violation of timing constraints. The monitoring part in our approach is coupled with the scheduler and by bringing awareness to the scheduler about the type of tasks it is scheduling, monitoring such information becomes a natural and straightforward part of the scheduler.

8.6 Discussion and Conclusion

In this paper, we introduced the concept of the second layer scheduler as an approach to bring semantics and awareness for different types of real-time tasks and their parameters to the scheduler without modifying it.

It was shown how this awareness improves the monitoring capabilities of the system to help with the detection of critical events such as deadline misses, and execution time overruns. While the approach was motived and described in the context of model-driven development of real-time systems to enable back-annotation of data and provide round-trip engineering support, it does not necessarily need to be used in this context and the concept of the second layer scheduler is applicable and practical per se.

Considering a larger set of timing parameters for scheduling of tasks and generating detailed log information in the second layer scheduler can bring along their own overheads. Measurement of these overheads and evaluation of the price of these added features are left to be done as a future work. Especially we plan to perform two overhead measurements: startup overhead (reading configurations and initializing tasks), and context switch and scheduling decisions overheads. It should however be noted that the actual logging is done by the monitor process in our design which behaves as a background process. The second layer scheduler only sends out a signal (including needed information) to the monitor process and continues its job (asynchronously) without using any critical section for data sharing by using message passing mechanisms of OSE. This way, the overhead of creating log information in the second layer scheduler process is tried to be mitigated.

Generation of such detailed monitoring information can also help with the predictability of real-time systems at runtime. For instance, even in cases where no deadline misses occur in the system, it becomes possible to observe how close tasks are to missing their deadlines and whether this gap is decreasing or increasing. Based on such analysis of monitoring information, the system can also adapt itself in order to prevent deadline misses.

One issue that we did not discuss in this paper is the priority inversion problem. This problem is handled automatically by OSE for communication among periodic tasks, but for sporadic and aperiodic tasks, the priority inversion issue should be more investigated. Also the way the system is designed for the background scheme, periodic tasks will have higher priority over sporadic tasks, and the priority of sporadic tasks will be higher than aperiodic ones. When the polling server scheme is used, the priority of sporadic tasks will be dependent on the priority of their periodic server, but still higher than aperiodic ones. This can also be extended to be configurable by the user. Moreover, in this work,

since the tasks were assumed to be generated from a model, the task set was considered to be known and static. We leave the extension of the implementation to accept new tasks dynamically as a future work. Also, the possibility to have different numbers and types of servers for sporadic and aperiodic tasks could be another future work.

Since the suggested approach is designed to be flexible in terms of the used scheduling algorithms, it would be interesting as a future direction of this work to investigate the possibility to let the system intelligently select an appropriate/optimal scheduling algorithm based on the requirements at the model level and generate code accordingly especially that the back-annotation mechanism can also be used as a feedback loop.

8.7 Acknowledgements

This work has been partially supported by the CHESS European Project (ARTEMIS-JU100022) [4] and XDIN AB [19].

Bibliography

[1] Bran Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20:19–25, September 2003.

[2] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture - Practice and Promise*. 2003.

[3] Enea. The Architectural Advantages of Enea OSE in Telecom Applications. `http://www.enea.com/software/products/rtos/ose/`, Last Accessed: February 2012.

[4] CHESS Project: Composition with Guarantees for High-integrity Embedded Software Components Assembly. `http://chess-project.ning.com/`, Last Accessed: April 2012.

[5] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Non-functional requirements: From elicitation to conceptual models. In *IEEE Transactions on Software Engineering*, volume 30, pages 328–350, 2004.

[6] Modeling and Analysis Suite for Real-Time Applications (MAST). `http://mast.unican.es/`, Last Accessed: February 2012.

[7] S.E. Chodrow, F. Jahanian, and M. Donner. Run-time monitoring of real-time systems. In *Real-Time Systems Symposium (RTSS). Proceedings., Twelfth*, pages 74 –83, dec 1991.

[8] Enea. `http://www.enea.com`, Last Accessed: April 2012.

[9] Brinkley Sprunt. Aperiodic Task Scheduling for Real-Time Systems. Technical report, Ph.D. thesis, Carnegie Mellon Univ, 1990.

[10] Robert Davis and Alan Burns. Hierarchical fixed priority preemptive scheduling. In *In Proceedings of the 26 th IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, 2005.

[11] Andy J. Wellings, Gregory Bollella, Peter C. Dibble, and David Holmes. Cost Enforcement and Deadline Monitoring in the Real-Time Specification for Java. In *7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, pages 78–85. IEEE Computer Society, 12-14 May 2004.

[12] Enrico Mezzetti, Marco Panunzio, and Tullio Vardanega. Preservation of Timing Properties with the Ada Ravenscar Profile. In Jorge Real and Tullio Vardanega, editors, *Reliable Software Technologies - Ada-Europe 2010*, volume 6106 of *Lecture Notes in Computer Science*, pages 153–166. Springer Berlin / Heidelberg, 2010.

[13] Juan Zamorano and Jose F. Ruiz. GNAT/ORK: An open cross-development environment for embedded Ravenscar-ADA software. In *Proceedings of the 15th IFAC World Congress*. Elsevier, 2003.

[14] FIRST Project: Flexible Integrated Real-Time Systems Technology. `http://www.frescor.org/index.php?page=related-projects`, Last Accessed: April 2012.

[15] FRESCOR Project: Framework for Real-time Embedded Systems based on COntRacts. `http://www.frescor.org/index.php`, Last Accessed: April 2012.

[16] Dario Faggioli, Fabio Checconi, Michael Trimarchi, and Claudio Scordino. An EDF scheduling class for the Linux kernel. In *Proceedings of the Eleventh Real-Time Linux Workshop*, Dresden, Germany, September 2009.

[17] Thomas Nolte, Moris Behnam, Mikael Åsberg, Reinder Bril, and Insik Shin. Hierarchical Scheduling of Complex Embedded Real-Time Systems. In *École d'Éte Temps-Réel (ETR'09)*, pages 129–142, August 2009.

[18] Yue Yu, Shangping Ren, and Ophir Frieder. Prediction of Timing Constraint Violation for Real-Time Embedded Systems with Known

Transient Hardware Failure Distribution Model. In *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, pages 454 –466, dec. 2006.

[19] XDIN AB. `http://ny.xdin.com/om-xdin/enea-experts/`, Accessed: June 2012.

Chapter 9

Paper D: An Automated Round-trip Support Towards Deployment Assessment in Component-based Embedded Systems

Federico Ciccozzi, Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin
The 16th International Symposium on Component-Based Software Engineering (CBSE), Vancouver, Canada, June, 2013.

Abstract

Synergies between model-driven and component-based software engineering have been indicated as promising to mitigate complexity in development of embedded systems. In this work we evaluate the usefulness of a model-driven round-trip approach to aid deployment optimization in the development of embedded component-based systems. The round-trip approach is composed of the following steps: modelling the system, generation of full code from the models, execution and monitoring the code execution, and finally back-propagation of monitored values to the models. We illustrate the usefulness of the round-trip approach exploiting an industrial case-study from the telecom-domain. We use a code-generator that can realise different deployment strategies, as well as special monitoring code injected into the generated code, and monitoring primitives defined at operating system level. Given this infrastructure we can evaluate extra-functional properties of the system and thus compare different deployment strategies.

9.1 Introduction

Complexity of embedded systems is continuously increasing and therefore solicits the introduction of more powerful and automated development mechanisms able to mitigate it. In this direction, Model-Driven Engineering (MDE) [1] and Component-Based Software Engineering (CBSE) [2] can be considered as two orthogonal ways of reducing development complexity through different means. The former shifts the focus of the development from hand-written code to models from which the implementation is meant to be automatically generated through the exploitation of model transformations. The latter breaks down the set of desired features and their intricacy into smaller replaceable sub-modules, namely components, starting from which the application can be built-up and incrementally enhanced. Moreover, their combination has been recognised as an enabler for them to definitely break through for industrial development of embedded systems [3].

In this research work we exploit the synergy between MDE and CBSE to demonstrate the benefits of an automated round-trip support in aiding deployment optimization when developing embedded systems. The round trip support consists of the following steps:

- Modelling: the first step is represented by modelling the system through a structural design, in terms of components, a behavioural description by means of state-machines and action code, as well as a deployment model describing the allocation of software components to operating system's processes;
- Code generation: from the information contained in the design model, we automatically generate full functional code. Note that in the paper we refer to generated code as *full* or *full-fledged* if it is entirely generated in an automated manner and does not require manual tuning in order to be executed on the selected platform;
- Monitoring: after the code has been generated we monitor its execution on top of the target platform and measure selected Extra-Functional Properties (EFPs). In fact, in modern complex embedded systems, certain EFPs cannot be accurately predicted at modelling level, hence requiring measurements at runtime. This would be the case, e.g., of performance-related EFPs, that often only emerge in a running product. As example, let us consider two sorting algorithms that speed up a program because they use a big portion of the main memory. Although both increase the perfor-

mance in isolation and they have no direct functional interaction, in combination they may degrade the overall performance because both share the same (too small) main memory [4];
- Back-propagation: at this point, gathered values are back-propagated to the design model and, after their evaluation, the deployment configuration can be manually tuned to generate more resource-efficient code.

The round-trip support and its usefulness in aiding the preservation of EFPs[1] from models to generated code have been already proven and discussed in [5] and [6]. Nevertheless, their limited support for platform configurations (i.e., only single-process) prevented their employment for deployment issues. The approach has been therefore enhanced in order to enable synthesis of design models to either a single-process or to a set of communicating processes. This gives flexibility to generate either highly resource efficient (in terms of inter-system communications) single-process systems or exploit multi-process configurations that could, e.g., run in parallel on multicores and/or maintain error encapsulation within one process.

In this work we describe the enhancements made to the round-trip support and how it can employ measurements gathered at system implementation (or runtime) level towards deployment optimization when developing embedded systems. More specifically, in order to enhance the generation of full code from models addressing multiprocess applications, additional modelling artefacts (e.g., deployment model) had to be considered, higher variability of the transformation process had to be addressed by enhancing the code generation transformations as well as expanding the intermediate metamodels to entail deployment information and diverse communication patterns. In order to employ the approach for deployment assessment the execution platform had to be modified for enabling monitoring facilities and back-propagating transformations were implemented too. Moreover, the approach has been validated against industrial case-studies from the telecom-domain.

The remainder of the paper is structured as follows. The scope of the paper is defined in terms of context delimitation and contribution formalisation in Section 9.2. The state-of-the-art related to similar approaches with focus on back-propagation features, monitoring activities and deployment optimization based on measurements at system imple-

[1] By preservation of EFPs we intend keeping them within their validity ranges and protect them from violation at runtime.

mentation level is described in Section 9.3. A running example in terms of an industrial case-study is introduced in Section 9.4 while the proposed solution is described in Section 9.5. The application of the proposed approach to the example is described in all its details in Section 9.6 with focus on showing how the round-trip support can aid the developer in taking deployment decisions. Section 9.7 proposes a discussion on the proposed solution and possible enhancements; the paper is then concluded by a summary in Section 9.8.

9.2 Context

Following the MDE paradigm, a system is developed by designing models and refining them starting from higher and moving to lower levels of abstraction until code is generated; refinements are performed through transformations between models. A *model transformation* translates a source model to a target model while preserving their well-formedness [7]. Since a model is an abstraction of the system under development, rules and constraints for building it have to be properly described through a corresponding language definition. In this respect, a *metamodel* describes the set of available concepts and well-formedness rules a correct model must conform to [8].

Since different nuances of the CBSE-related terminology can be found in the literature, in this work we refer to *component-based development* as prescribed by the UML Superstructure [9]. That is to say, a system is modelled as an assembly of components communicating via required and provided interfaces exposed by ports, where a port represents an interaction between a classifier instance and its internal or external environment. Additionally, features owned by required interfaces are meant to be offered by one or more instances of the owning classifier to one or more instances of the classifiers in its internal or external environment.

A fairly wide variegation of different approaches to the measurement of EFPs at system implementation level exists. In this work we focus on *runtime monitoring*, that represents a method to observe the execution of a system in order to determine whether its actual behaviour is in compliance with the intended one. In comparison to other verification techniques such as static analysis, model checking, testing and theorem proving which are used mainly to determine "universal correctness" of software systems, runtime monitoring focuses on each instance

and current execution of a system [10].

In the following sections we describe the scope of the proposed solution in terms of modelling language, target platform and intended contribution.

9.2.1 CHESS Modelling Language

In our work we employ the *CHESS* modelling language (CHESS-ML) [11], defined within the CHESS project (cf. Acknowledgements) as a UML profile, including tailored subsets of SysML and MARTE profiles. CHESS-ML allows the specification of a system together with relevant EFPs such as predictability, dependability and security. Moreover, it supports a development methodology expressly based on separation of concerns; distinct design views address distinct concerns. In addition, CHESS actively supports component-based development as prescribed by the UML Superstructure [9].

According to the CHESS methodology, functional and extra-functional characteristics of the system are defined in specific separated views as follows:

- Functional: UML component and composite component diagrams are employed to model the structural aspects of the system while state-machines and activity diagrams are used to express functional behaviour. Action Language for Foundational UML (ALF) [12] is used to enrich the behavioural description. In this way, we reach the necessary expressive power to be able to generate full implementation code directly from the functional models with no need for manual fine-tuning of the code after its generation;
- Extra-functional: in compliance with the principle of separation of concerns adopted in CHESS, the functional models are decorated with extra-functional information thereby ensuring that the definition of the functional entities is not altered. In respect to the back-propagation proposed in this work, it is worth clarifying that there is no cloning nor versioning of the design model but rather a decoration in terms of values updates on the extra-functional stereotypes. This means that, at each iteration, the EFPs are updated and, if any modification to the model is manually performed, no history of previous versions is kept.

9.2.2 OSE Real-Time Operating System

OSE is a commercial and industrial real-time operating system developed by Enea [13] which has been designed from the ground specifically for fault-tolerant and distributed systems. It is widely adopted mainly in telecom-domain and systems ranging from mobile phones to radio base stations [14]. OSE provides the concept of direct and asynchronous message passing for communication and synchronisation between tasks, and its programming model is based on this concept. This allows tasks to run on different processors or cores, utilising the same message-based communication model as on a single processor. This programming model provides the advantage of avoiding the use of shared memory among tasks. In OSE, the runnable real-time entity equivalent to a task is called *process*, and the messages that are passed between processes are referred to as *signals* (thus, the terms *process* and *task* in this paper can be considered synonyms).

A system modelled through the CHESS-ML is manipulated through model transformations to generate C++ code which constitutes definition and functionality of the processes that will run on the platform. To retrieve information about desired EFPs, the platform has been extended with a set of monitoring and logging processes which collect information about the behaviour of the generated code in terms of memory and CPU usage of a process, execution time, response time, number of signals generated by a process (to mention a few), as well as several system-level properties such as overall CPU usage and total number of signals in the system.

9.2.3 Contribution

Goal The aim of this work is to lessen the developer's effort devoted to optimise deployment configuration in the development of component-based embedded systems exploiting measurements gathered at system implementation level and avoiding manual editing of code.

Solution The solution we propose is represented by a fully automated model-driven round-trip approach that enables:

- Generation of full-fledged code from source models entailing customisable single- and multi-process deployment configurations;

- Monitoring of code execution for measuring selected EFPs at system implementation level, such as CPU load, number of generated signals, system throughput, execution time, response time of a process, instance execution time of a process, heap and stack usage;
- Back-propagation facilities to enrich the source models with values gathered by monitoring.

Through the proposed approach, the developer exploits model-driven techniques, thus operating exclusively at modelling level, and at the same time takes advantages of measurements gathered at runtime and automatically brought back at modelling level. This is of critical importance in order to assist the developer in understanding at a glance the relationships between expected and actual behaviour, without having to inspect the related generated code. Thanks to the automated support, the process can be iterated at will until the developer is satisfied with the results.

Specific Contribution Considering that the round-trip support and its usefulness in aiding the preservation of EFPs from models to generated code have been already presented in [5], the specific contributions of this paper are the following:

1. Enhancements to the code generation for entailing the synthesis of both single-process and multi-process configurations (Section 9.6.1);
2. Specific extensions to the execution platform in order to enable per process monitoring features for EFPs (Section 9.6.2);
3. Adaptation of the back-propagation facilities (i.e., in-place model transformations) to address newly introduced monitoring features (Section 9.6.3);
4. Exploitation of the synergy between round-trip support and measurements gathered at system implementation (or runtime) level towards deployment optimization when developing embedded systems (Section 9.6.4).

9.3 Related Work

Attempts that propose to solve the problem by back-propagation, i.e., by reporting the measured values back to the models in order to possibly fix and/or refine estimated numbers can be found in the literature. Nav-

abi et al. in [15] in the early 90's, and later Mahadevan and Armstrong in [16], proposed different approaches for back-annotating behavioural descriptions with timing information; however, both operate horizontally[2] in terms of abstraction levels and no automation is provided.

Moreover, Varró et al. propose in [17] back-propagation for enabling execution traces retrieved by model checkers or simulation tools to be integrated and re-executed in modelling frameworks; some similarities to our approach might be found when dealing with traceability issues, although the two approaches aim at solving two different problems. An approach similar to ours is described by Guerra et al. in [18] where back-propagation of analysis results to the original model by means of triple graph patterns is described. Nevertheless, the approach is meant to horizontally operate at modelling level with propagation of data among models. Our approach focuses on vertically propagating analysis results observed at code level back to design models for better understanding of those EFPs that cannot be accurately predicted without executing the code on a specific platform.

In the literature there exist approaches dealing with deployment optimization based on measurements at system implementation level as described in [19]. A dated approach by Yacoub in [20] introduces systematic measurements of a component-based system, but no tool support is provided.

The COMPAS framework by Mos et al. [21] is a performance monitoring approach for J2EE systems. Components are EJBs and the approach consists of monitoring, modelling, and prediction. An EJB application is augmented with proxy components for each EJB, which send timestamps for EJB life-cycle events to a central dispatcher. Performance measurements are then visualised with a proof-of-concept graphical tool and a modelling technique is used for generating UML models with SPT annotations from the measured performance indices. Then, for performance prediction of the modelled scenarios, the approach suggests using existing simulation techniques, which are not part of the approach. Based on the COMPAS framework, two further approaches have been proposed: AQUA, by Diaconescu et al. in [22], and PAD, by Parsons et al. in [23]. Both approaches expect working J2EE applications as input. AQUA focuses on adapting a component-based application at runtime

[2]Horizontal and vertical are used for specifying the direction of data transitions among artefacts; therefore with *horizontal* we intend transitions from model to model, while with *vertical* we mean those from code to model.

if performance problems occur. The main idea is that a software component (EJB) with performance problems is replaced with one which is functionally equivalent from a set of redundant components. Furthermore, the approach involves monitoring the workload of a running application. PAD focuses instead on automatic detection of performance anti-patterns in running component-based systems. The approach targets EJB systems and includes performance monitoring, reconstruction of a design model, and anti-pattern detection.

The described approaches assume that a complete component-based system has been implemented and can be run. The goal is therefore to identify performance problems in the running system and adapt the implementation to make it able to fulfil EFPs requirements. Instead, the uniqueness of our round-trip approach consists in introducing a new dimension to deployment optimization at model level with the help of measurements gathered at system implementation level. In fact, when measurements are completed, the code is not manually tuned, but changes to the system are rather performed at model level from which code is re-generated. Doing so, consistency between models and code is kept and thereby the validity of decisions made at model level is likely to be preserved at code level (and the other way around). Moreover, by exploiting the accuracy of system implementation level measurements at modelling level, the developer is relieved from complex code inspection and error-prone manual tuning of code.

Regarding measurements of EFPs at system implementation level, besides runtime monitoring, other verification techniques (e.g., static analysis) can be used for small and simple systems, but their application for large and complex systems might not always be practical and economical [24]. Even in cases where such techniques are feasible, conditions that cause invalidation of the analysis results at runtime may happen. An example of such is the difference between the ideal execution environment (considered for performing analysis) and the actual one which leads to the violation of the assumptions that were taken into account when performing static analysis [25]. Therefore, the information gathered through monitoring the execution of a system is not only interesting and useful for observing the actual behaviour and to detect violations at runtime, but also to be used for making adaptation decisions, as well as to induct enforcement and preservation of properties. Our work in [26] serves as an example of using monitoring information for balancing timing and security properties in embedded real-time systems. Also in

[6], we provided an approach for improved enforcement and preservation of timing properties in embedded real-time systems. Huselius and Andersson in [27] present a method for the synthesis of models of embedded real-time systems from the monitoring information collected from their execution. In this paper, however, we exploit monitoring results, from which observed values are extracted and used to refine design models with EFPs' values detected during the execution of the system.

9.4 The AAL2 Subsystem: a Running Example

The solution proposed in this work has been validated against industrial case-studies modelled in CHESS-ML as described in Section 9.7. In order to show a concrete application of the proposed solution, we employ the Asynchronous Transfer Mode (ATM) Adaptation Layer 2 (AAL2) subsystem, originally intended to adapt voice for transmission over ATM and currently used in telecommunications as part of connectivity platform systems. The AAL2 subsystem is composed by several thousands of component instances and multiple levels of hierarchical composition of components. Due to its verbosity and complexity, we will exploit a simplified version of the AAL2 to show how the approach operates. Nevertheless, the validation of the approach has been carried out exploiting the complete AAL2 subsystem. In Fig. 9.1 we propose the simplified version of the AAL2 subsystem (i.e., `SwSystem` composite component) which is composed by three main components: (i) `NCC_i`, (ii) `AAL2RI_Client_i`, (iii) `NCIClient_i`. Note that the multiplicity (in terms of number of instances) of components and ports is depicted within square brackets. Each of the components has a complex internal structure in terms of composition of other components; in this example we consider only part of the `NCC_ci` internal structure while considering `AAL2RI_Client_i` and `NCIClient_i` as stubbed. `NCC_i` is a connections handler providing connectivity services for the establishment/release of communication paths between pairs of connection endpoints handled by `AAL2RI_Client_i`. `NCIClient_i` represents an application asking for services provided by `NCC_i` and its underlying layers; the components communicate through functional interfaces (function calls or message passing depending on the deployment configuration) exposed by their provided ports.

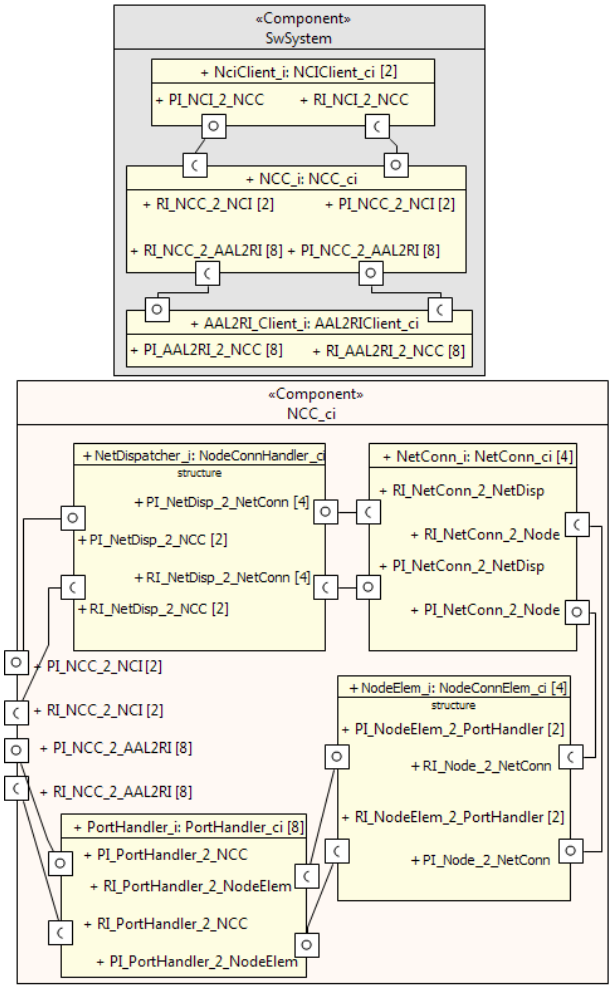

Figure 9.1: Structural design of AAL2 in CHESS

The `NCC_ci` is a composite component (Fig. 9.1), and in this study we focus on: `NodeConnHandler_i`, which dispatches the incoming connection requests to available NetConn_i instances, `NetConn_i`, that controls establishment and release of network connections between nodes (Node-

ConnElem_i instances), `NodeConnElem_i`, that handles management of connections to the network within the single node, and `PortHandler_i`, which manages connection resources. Each of these subcomponents has in turn a complex internal structure in term of components composition; in this case-study we consider only the first two levels of decomposition (down to the `NCC_ci`'s internal structure).

The behaviour of the system (`NetConn_i` state-machine is defined through state-machines and are enriched with action code definitions for the involved operations specified by means of ALF in order to reach the required level of expressive power needed to automatically generate full code. Components are connected by means of ports and links between them. The communication is thereby performed by calling operations on the component's required ports that propagate the invocation to the component owning the provided ports connected to them (note that connected provided and required ports share the same *Interface*). A typical connection scenario in the AAL2 subsystem is the establishment of a connection between two end-points residing on the same node. This is a constrained case of a more general network-wide connection where the two end-points reside on different nodes and the communication transits through a number of other intermediate nodes in the network.

9.5 The Round-trip Support

This work proposes the exploitation of a round-trip technique to provide an automated support to aid deployment decisions at modelling level in the development of component-based embedded systems. Such a support is achieved by exploiting the multiple benefits of the round-trip technique, namely the generation of full-fledged functional code from source models, a set of monitoring features to gather values of EFPs of interest from the execution of generated code on the target platform, and the possibility to automatically propagate the computed values back to the source models. Doing so, source models can be evaluated and possibly tuned in terms of deployment of software components on processes to enable a better resource utilisation based on actual values.

In order to maintain consistency between different artefacts in the automated process and therefore ensure correctness of monitoring and back-propagation activities, the generated code is not meant to be edited by hand. Possible optimizations are indeed not performed directly through

code editing, but rather by re-iterating the code generation process once the deployment model has been refined according to the evaluation of the back-propagated EFPs' values. This is especially important in the context of complex systems and in large organisations, where usually code tweaks are done at sub-system and sub-department levels to achieve, for instance, better performance. This can in turn lead to inconsistencies among development artefacts and modifications often remain unknown to developers at upper levels of the organisational hierarchy [28]. The fact that EFPs (e.g., memory usage and performance) are not independent and have inter-dependencies further emphasises the importance of having such a consistency mechanism for development artefacts since, most often, EFPs cannot be considered in isolation and their mutual impacts and trade-offs need to be taken into account too [29]. The general goal of this work is to demonstrate that the round-trip support helps the developer in determining the most suitable deployment configuration based on actual values gathered by executing and monitoring code which is automatically generated from source models (Fig. 9.2). Once design modelling tasks have been successfully completed, the objective is to enable automatic generation of implementation code from source models. Taking design models as source artefacts [3], we generate target code through appropriate model transformations (Fig. 9.2.a).

Information regarding tracing of source (e.g., model elements) and target (e.g., code segment(s)) artefacts has to be defined and maintained for further back-propagation activities. Therefore, code generating transformations have to be properly defined by encoding apposite rules for the generation of traceability links (explicit traceability [30]) between models and code (Fig. 9.2.b); such rules populate the back-propagation model with traceability information according to the metamodel described in [5]. Once the code has been generated as well as the traceability links, EFPs can be measured by code execution monitoring features (Fig. 9.2.c).

The monitoring features that have been added to the platform enable collecting information about the desired EFPs of the generated system in order to assess its behaviour at runtime and thus decide whether it is satisfactory with respect to the resource constraints. This capability is achieved by back-propagating to the models the necessary information about the behaviour of the system and its components gathered at

[3]Throughout the paper, source and design models refer to the artefacts at the highest level of abstraction as shown in Fig. 9.2

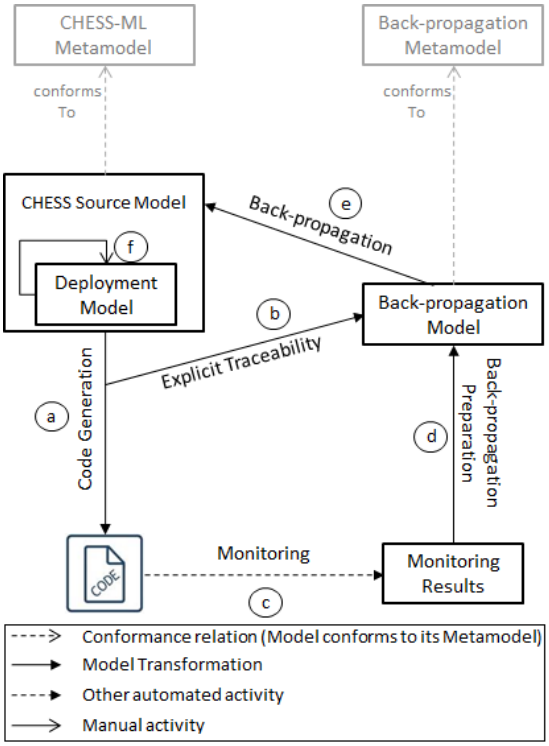

Figure 9.2: Proposed Round-trip Solution

runtime. Since the monitoring is performed per process (as the runtime entity), a one-to-one mapping between component and process at modelling level (i.e., a process is created for each component at the model level) would enable the monitoring of EFPs per component.

Depending on the monitoring's output format, different actions, varying from text-to-model to model-to-model transformations (Fig. 9.2.d), are required to extract and formalise execution results in order to complete the traceability chain from models to monitoring results. The last step of the round-trip approach aims at finally annotating the source models with the code execution results (Fig. 9.2.e) through dedicated in-place model-to-model transformations.

At this point the source models are evaluated by means of actual

EFPs' values and, if needed, the allocation of components to processes can be tuned (Fig. 9.2.f) by the developer to generate more resource-efficient code. Thanks to the automated support, the process can be re-iterated until the developer is satisfied with the generated code.

9.6 From Models to Code and Back

In this section we show the application of the round-trip support to the AAL2 subsystem in terms of generation of C++ code from the source model described in Section 9.4, monitoring of the code execution on OSE and back-propagation of gathered values to the CHESS model. Moreover, after performing deployment tuning based on such values, we re-iterate the process to show how these modifications at modelling level can affect EFPs of re-generated code.

9.6.1 Generation of C++ from CHESS Model

A set of model-to-model transformations operates on the CHESS model to generate three different intermediate artefacts: i) *Instance model*, that stores component instances and links among them via ports [31], ii) *Intermediate model*, that represents the main intermediate artefact in the process and contains all the information needed to generate code [5], and iii) *Back-propagation model*, that contains explicit traceability links between model elements and code as well as place-holders for EFPs values coming from the following monitoring activities [5]. The intermediate model is enriched with behavioural information defined in ALF by means of parsing operations and in-place model-to-model transformations. At this point, the C++ implementation code is generated through model-to-text transformations taking as input the intermediate model [5].

Each of these intermediate artefacts conforms to its corresponding metamodel that we expressly defined for the code generation process. With regards to the technologies employed in the generation process, model-to-model transformations are defined by means of Operational QVT, while model-to-text transformations are defined in Xpand [32].

As aforementioned, the synthesis described in [5] was limited to single-process applications, which prevented its employment for deployment issues. The approach has been therefore enhanced in order to be able to consider deployment configurations specified at modelling level

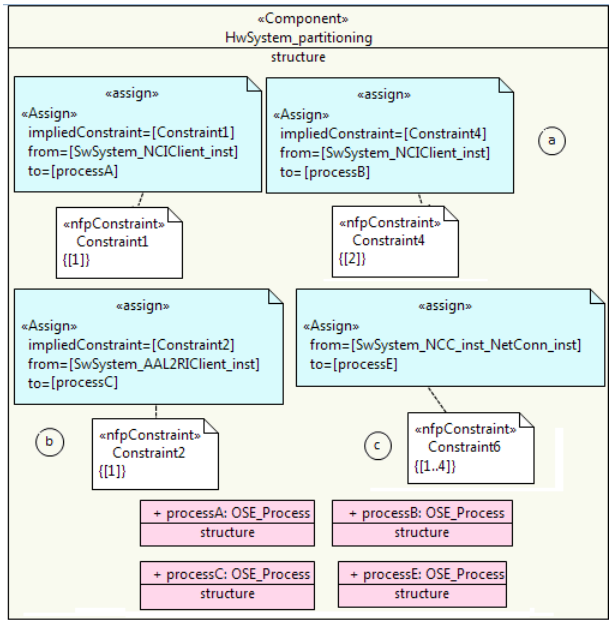

Figure 9.3: Multiprocess Deployment Configuration of AAL2 Subsystem

and therefore generate single- or multi-process implementation accordingly. Deployment is described in CHESS-ML by means of an ad-hoc model in which the modeller defines allocation of components instances to processes with the help of specific concepts provided by MARTE. In Figure 9.3 a portion of the deployment model concerning the AAL2 subsystem is depicted.

More specifically the following possible allocations are shown:

- Different instances of a same component on different processes: represented by the two instances of the component `NCIClient_i` (`SwSystem_NCIClient_inst`) respectively allocated to `processA` and `processB` (Figure 9.3.a);
- Single component instance on a process: represented by the instance of `AAL2RIClient_i` (`SwSystem_AAL2RIClient_inst`) allocated to `processC` (Figure 9.3.b);
- Multiple set of component instances on a single process: represented by the entire set of `NetConn_i` (`SwSystem_NetConn_inst`)

instances allocated to `processE` (Figure 9.3.c).

In order to achieve this, the transformation process proposed has been enhanced to take into account the information carried by the deployment model that, together with the functional model, drives the code generation. More specifically, in the case of code tailored for OSE, that information is exploited to create processes and deploy component instances on them, as well as generate the communication code in order to distinguish between:

- **Intraprocess communication**: communication between components deployed on a same process is achieved by function calls;
- **Interprocess communication**: components allocated to different processes communicate via signals across processes.

More specifically, the communication between components defined in the CHESS-ML model in terms of ALF as function calls on ports had to be properly translated into appropriate intermediate concepts. Depending on the deployment configuration of the communicating components, a function call in the model is translated into (i) a function call, if the components are deployed on the same process, or (ii) into a message `send` (i.e, OSE signal) in the case of components deployed on different processes. This means that the entailment of different deployment configurations has introduced a higher degree of variability in the semantic interpretation of the design models that depends on the modelled deployment. The ability to correctly interpret such information and thereby generate code that reflects the modelled deployment configuration has been embedded in the model transformations responsible for the code generation.

Due to the complexity of the multi-process code generation, the details related to the involved artefacts ranging from intermediate metamodels to model transformations will be dealt with separately. For the interested reader, preliminary information about the single-process generation chain can be found in [5].

9.6.2 Monitoring Code Execution

In order to enable the needed monitoring features, specific extensions to the execution platform have been made. These extensions have been implemented mainly in the form of two additional system processes: one for monitoring and another for logging. These two processes are assigned lower priorities than the generated application ones. The monitoring pro-

cess is responsible for calculating and determining the values for EFPs of interest for both the whole system (e.g., overall CPU usage) as well as per process. The actual task of logging such information is separated from the monitoring process and performed by the logging one. This separation allows to mitigate the side effects of resource-demanding I/O activities. When a request for monitoring is issued by one of the application processes, the monitoring process starts executing and determining EFPs' values. The information to be logged is sent to the logging process by the monitoring one through apposite signals. Therefore, if the logging process does not get the needed CPU time to perform its job, the signals sent to it are pushed in its signal queue, maintained automatically by OSE, and processed as soon as it gets to execute.

The implemented monitoring process is capable of determining values for the following properties:

- System-level properties: total CPU load, total number of generated signals in the system, system throughput (sent and received packets), total number of processes in the system;
- Process-level properties: total execution time of a process (from the startup of the system including all invocations of it), instance execution time of a process (one invocation and instance), response time of a process, heap and stack usage for a process, number of signals generated by a process, and CPU load of a process.

To calculate execution and response times, `swap_in` and `swap_out` handlers of OSE have been used. The former event handler is invoked each time a process gets CPU to execute, and the latter is invoked when CPU is taken from a process and it is preempted. The algorithms and mechanisms for the calculation of execution and response times have been implemented into these two event handlers. However, since they are invoked for every process in the system, additional tweaks were made in order to filter their executions for only the generated application processes which are of interest.

The monitoring activities give a textual description of the gathered values as output. The results of monitoring the execution of the C++ code generated from the AAL2 model and with the deployment configuration partially depicted in Fig. 9.3 are shown partially in Listing 9.1.

Listing 9.1: Monitored Properties

```
466,PROCID, mprocA,1003c,65596
476,S_CPU_LOAD, 29.9780
476,S_NUMBER_OF_PROCESSES, 73476
476,S_NUMBER_OF_SIGNALS, 365
476,S_MTBF, 558.6177479
479,S_THROUGHPUT, 942,1676
479,P_HEAP_USAGE, 384,512
479,P_STACK_USAGE, 1536,0,2048
479,P_NUMBER_OF_SIGNALS, 2
579,P_EXECUTION_TIME, 14
579,P_RESPONSE_TIME, 1609984
-----------------------------------
579,PROCID, mprocE,10040,65600
...
-----------------------------------
612,PROCID, mprocC,1003e,65598
622,S_CPU_LOAD, 9.8427
622,S_NUMBER_OF_PROCESSES, 73
622,S_NUMBER_OF_SIGNALS, 675
622,S_MTBF, 558.6177
622,S_THROUGHPUT, 820,1676
623,P_HEAP_USAGE, 384,512
623,P_STACK_USAGE, 1536,0,2048
623,P_NUMBER_OF_SIGNALS, 2
623,P_EXECUTION_TIME, 11
623,P_RESPONSE_TIME, 1619979
-----------------------------------
623,PROCID, mprocB,1003d,65597
...
```

The first column in Listing 9.1 indicates the time instance at which monitoring has been performed (in system ticks unit). The second column identifies the type of the monitored information; the properties beginning with 'S_' indicate a system-level value while the ones starting with 'P_' identify a process-level value (e.g., S_NUMBER_OF_SIGNALS: total number of signals in the system at the moment of monitoring, P_NUMBER_OF_SIGNALS: total number of signals owned by a process). The values after the name of the process (i.e., mprocA) indicates process' ID in hexadecimal and decimal format respectively. As it can be seen, some of the properties have multiple values, in which case they mean different aspects related to the same property. For example, the first value related to P_HEAP_USAGE represents the heap size requested by the process and the second one shows the actual heap size allocated for the process by the operating system (the difference between the two is due to factors such as memory paging and memory management mechanisms of OSE).

9.6.3 Back-propagation to CHESS Model

While a similar approach had been proposed in [5], since the monitoring features, and thereby the output format, had entirely changed, the back-propagation facilities had to be consequently adapted. More specifically, a specific in-place text-to-model transformation has been implemented from scratch in order for the monitoring results of interest to be manipulated and stored in the back-propagation model. Observed values as source for the back-propagating transformations are not enough. In fact, the traceability chain defined along the path from design models to observed values is also part of the source artefacts to be fed to the transformations in order to correctly propagate values back to the design models. The actual enrichment of the CHESS model with such EFPs values is performed through a model-to-model transformation. Taking as input the CHESS model and the back-propagation model, it performs a set of in-place transformations (adapted version of the ones proposed in [5]) on the CHESS model to enrich it with the observed values stored in the back-propagation model. The results of the back-propagation are partially shown by means of extra-functional decorations of the AAL2 model in Fig. 9.4. More specifically, we can notice that values concerning *mprocA* are back-propagated to instance 1 of `NCI_Client_i` (marked by [1] – e.g., `EXECUTION_TIME[1]`) while the ones carried by *mprocC* apply to the single instance of `AAL2RI_Client_i`. Such correspondence is stored in the back-propagation model and originates from the deployment model depicted in Fig. 9.3. Regarding the technologies used in the back-propagation process, model-to-model transformations are defined by means of Operational QVT as well as text-to-model transformations (which are aided by structured text parsers defined in Java). Once the monitored results have been back-propagated, the developer has at her disposal the modelled system enriched with actual values gathered at runtime. The values depicted in Listing 9.1 and propagated back to the AAL2 model (in Fig. 9.4) are related to the deployment configuration in which the component instance `NCI_Client_i[1]` is deployed on `ProcessA` (*mprocA* in Listing 9.1) and the component instance `AAL2RI_Client_i[1]` is allocated to `ProcessC` (*mprocC*).

9.6.4 Evaluation of Deployment Configurations

At this point, let us try out a different deployment configuration in which we allocate both `NCI_Client_i[1]` and `AAL2RI_Client_i[1]` to

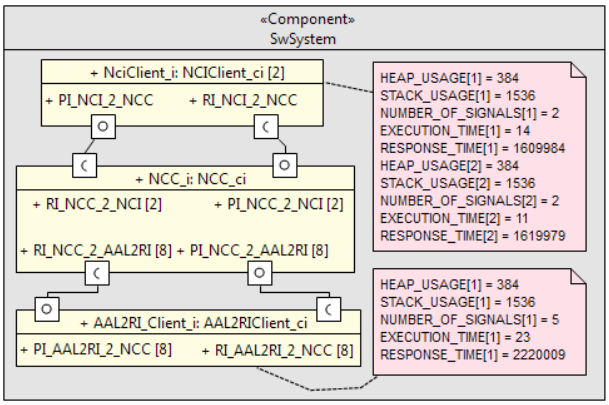

Figure 9.4: Decorated AAL2 model in CHESS

`ProcessA`, since the communication between them is quite dense. Once the model is modified, the code can be regenerated and its execution monitored. In Table 9.1 the monitoring results concerning both the first as well as the tuned deployment configurations are depicted. Configuration `1` represents the deployment of the two component instances on separate processes, while `2` concerns the deployment of both instances on a single process.

As we can see, by changing the deployment configuration, in the specific case by allocating the two instances on the same process, we actually experience a decrease of the execution time of `AAL2RI_Client_i` from 11 to 3. Besides this reduction which may not be relevant in the actual employment of the system, what we aimed at pointing out was the usefulness of having an automatic mechanism for gathering and back-propagating runtime values to model level for allowing thorough evaluation of the system's deployment configuration. While providing meaningful values at model level, the approach is not yet able to provide any hint on how to interpret them. Limitations of the current solution in this sense as well a future enhancements towards further automation in the tuning phase are discussed in the next section.

Config.	Comp. Instance	Process	Exec. Time
1	NCI_Client_i[1]	A	14
1	AAL2RI_Client_i	C	**11**
2	NCI_Client_i[1]	A	14
2	AAL2RI_Client_i	A	**3**

Table 9.1: Different deployments that induce different monitoring results

9.7 Discussion and Future Work

Validating the approach against an industrial case-study gave us the possibility to evaluate the scalability of the proposed solution by testing several design model sizes for the same system. The hardware configuration was composed by a Windows machine running on a 2.6GHz CPU and 8GB RAM. Moreover, we employed the OSE Soft-Kernel (SFK) 5.5.1 as target platform for code execution. The SFK edition provides a simulated environment of the actual target embedded board on a host machine (a Windows or Linux host) and accurately reproduces the behaviour of the real environment. Focusing on the most time-consuming task in the approach (i.e., automatic code generation) and considering n as the greatest number of instances per component, m as the greatest number of instances per port, and k as the number of hierarchical composition level, the general limit behaviour of the computation is represented by $O((n * m)^k)$. Overall the proposed solution, on a model with $k = 2$, resulted very scalable up to $n + m = 10^3$ (i.e., within 5 minutes) while gradually slowing down for greater number of instances (e.g., over 30 minutes for $n + m = 10^4$). More specifically, in terms of number of component and port instances (i), the generation time (in seconds) was: (i) 52s for $i = 90$, 113s for $i = 1853$, and 831s for $i = 16003$ (complete AAL2 subsystem). However, the presented validation has been performed on a preliminary implementation of the code generator which focuses mostly on the correctness of the generation process with no major emphasis on performance issues. That is why we expect to achieve better results, in terms of scalability, with a more mature version of the code generator.

Regarding the time needed for monitoring activities, it heavily depends on the duration of the code execution since the measurements themselves are mostly performed by parallel processes during the execution. Nevertheless, the calculation of more complex EFPs could introduce additional complexity hence requiring additional computation

time.

Concerning back-propagation tasks, they resulted to be more scalable thanks to the detailed information, concerning the path to the specific model element to be annotated, carried by the back-propagation model. This means that most of the needed computation is performed when generating the back-propagation model, while the actual values injection, first from monitoring results to back-propagation model and therefore from the latter to the design model only involves an update of specific values with no need of complex searches nor navigations.

Generally, the number of iterations for reaching the desired EFPs depends on the accuracy of measurements as well as the modeller's ability in both modelling the system and also effectively employ the back-propagated values to tune the models accordingly. That is to say that the developer is supposed to be able to understand the back-propagated values in relation to the expected behaviour and thereby tune the models accordingly to generate a better-performing implementation. In aid to the modeller, model-based analysis and deployment optimization techniques could be exploited to minimise the number of iterations.

As described in the previous sections, measurements of EFPs are generally performed at process level, which means that in order to obtain component-level values, each component should be deployed on a separate process. This, although representing an important step forward in comparison to the previous version of our approach, only opens up to further enhancements. In fact, while in principle the back propagation can handle the case of several components mapped to one process, the monitoring facilities cannot due to the execution platform. Therefore, future improvements will entail the possibility to always monitor properties at runtime per component even when several components would be deployed on a same process. In order to achieve this, both improved dedicated marking in the generated code as well as specific tweaks at platform level should be defined and implemented.

We plan to further enhance the code generation in order to address deployment on multiple processing units, which would amplify the benefits of multiprocessing addressed in this work (i.e., multi-process on single processing unit).

As aforementioned, starting from an initial model, the approach is iterated at will until the EFPs requirements are fulfilled. Therefore, once providing support for multicore solutions, there would be the possibility to minimise the number of such iterations by exploiting model-based

deployment optimization techniques (e.g., [33] by Feljan et al.) to enhance the deployment-tuning phase with a semi-automated approach that combines both runtime measurements as well as design-time performance predictions.

Additionally, as for the monitoring part, by adding the support for further EFPs, it becomes possible to take into consideration more parameters for tuning the deployment of the system. It would be possible, as an extension of the monitoring part, to enable the calculation of MTBF for each generated process, and therefore for the components that are mapped to it. Having MTBF values as one of the parameters indicating the dependability of the system, would add a further dimension (i.e., dependability) at model level that may help in making deployment decisions. Similarly, if there would be a mechanism to determine an indicator value for the energy consumption of each process (e.g., something similar to the ACPI standard [34]) it would become possible to even tune the system in terms of energy consumption. The feasibility of this possible enhancement needs further investigations.

9.8 Conclusion

In this paper we described a possible solution to relieve the developer's effort in optimising deployment configuration when developing component-based embedded systems with the help of accurate measurements gathered at system implementation level and avoiding manual editing of code. The solution consists of a fully automated model-driven round-trip approach that enables: (i) generation of full-fledged code from source models entailing customisable single- and multi-process deployment configurations, (ii) monitoring of code execution for measuring a selected set of EFPs at system implementation level, and (iii) back-propagation facilities to enrich the source models with such measurements.

By operating exclusively at modelling level, the developer is discharged from inspecting as well as editing generated code. At the same time, through automatic back-propagation of measurements gathered at system implementation level, the approach assists the developer in understanding at first sight the relationships between expected and actual behaviour hence enabling a well-aware tuning of deployment configuration at model level.

While a previous version of the round-trip support (i.e., only single-

process deployment was entailed) had been already described in [5], in this work we describe how we enhanced it to enable synthesis of design models to generate either highly resource efficient (in terms of inter-system communications) single-process applications or exploit multi-process configurations that could run in parallel on multicores. Moreover, having at our disposal this variety of deployment options, we were able to focus on how the round-trip support can employ measurements at system implementation level towards deployment optimization and we showed it by applying the solution to an industrial example in the telecom-domain. Additionally, we described complexity and scalability of the different steps constituting the solution, as well as interesting directions for possible future enhancements.

9.9 Acknowledgments

This research is supported by the RALF3 (Swedish Foundation for Strategic Research (SSF), `http://www.mrtc.mdh.se/projects/ralf3/`) project and the Swedish Knowledge Foundation (KKS) through the ITS-EASY research school.

Bibliography

[1] J. Bezivin. On the unification power of models. *Software and Systems Modeling*, pages 171–188, 2005.

[2] Ivica Crnkovic. Component-based software engineering for embedded systems. In *Procs of ICSE'05*, pages 712–713. ACM.

[3] R. Land, J. Carlson, S. Larsson, and I. Crnkovic. Project Monitoring and Control in Model-driven and Component-based Development of Embedded Systems - The CARMA Principle and Preliminary Results. In *Procs of ENASE'10*, pages 253–258.

[4] N. Siegmund, M. Rosenmuller, M. Kuhlemann, C. Kastner, and G. Saake. Measuring Non-Functional Properties in Software Product Line for Product Derivation. In *Procs of APSEC'08*, pages 187 –194, 2008.

[5] F. Ciccozzi, A. Cicchetti, and M. Sjödin. Round-trip support for extra-functional property management in model-driven engineering of embedded systems. *Information and Software Technology*, 2012.

[6] M. Saadatmand, M. Sjödin, and N. U. Mustafa. Monitoring Capabilities of Schedulers in Model-Driven Development of Real-Time Systems. In *Procs of ETFA 2012*, 2012.

[7] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, pages 621–645, 2006.

[8] S. Kent. Model Driven Engineering. In *IFM*, 2002.

[9] Object Management Group (OMG). UML Superstructure Specification V2.3. `http://www.omg.org/spec/UML/2.3/Superstructure/PDF/`, 2011. [Online. Last access: 11/04/2012].

[10] N. Delgado, A.Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *Software Engineering, IEEE Transactions on*, 30(12):859 – 872, 2004.

[11] A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, A. Zovi, and T. Vardanega. CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems. In *Procs of ASE'12*, pages 362–365. ACM.

[12] OMG. Action Language For FoundationalUML - ALF. `http://www.omg.org/spec/ALF/`, Oct 2010.

[13] Enea. `http://www.enea.com`, Last Accessed: January 2013.

[14] Enea. The Architectural Advantages of Enea OSE in Telecom Applications. `http://www.enea.com/software/solutions/rtos/`, Last Accessed: January 2013.

[15] Z. Navabi, S. Day, and M. Massoumi. Investigating Back Annotation of Timing Information into Dataflow descriptions. In *Procs of VHDL International User Forum*, pages 185–195, 1992.

[16] G. Mahadevan and J. R. Armstrong. Automatic Back Annotation of Timing into VHDL Behavioral Models. In *Procs of VHDL International User Forum*, pages 27–41, 1995.

[17] Á. Hegedüs, G. Bergmann, I. Ráth, and D. Varró. Back-annotation of Simulation Traces with Change-Driven Model Transformations. In *Procs of SEFM'10*, pages 145–155, 2010.

[18] E. Guerra, D. Sanz, P. Díaz, and I. Aedo. A transformation-driven approach to the verification of security policies in web designs. In *Procs of ICWE'07*, pages 269–284, Berlin, Heidelberg. Springer-Verlag.

[19] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658, 2010.

[20] S. Yacoub. Performance Analysis of Component-Based Applications. In *Software Product Lines*, LNCS, pages 299–315. Springer Berlin Heidelberg, 2002.

[21] A. Mos and J. Murphy. A framework for performance monitoring, modelling and prediction of component oriented distributed systems. In *Procs of WOSP'02*, pages 235–236. ACM.

[22] A. Diaconescu and J. Murphy. Automating the performance management of component-based enterprise systems through the use of redundancy. In *Procs of ASE'05*, pages 44–53. ACM.

[23] T. Parsons and J. Murphy. Detecting Performance Antipatterns in Component Based Enterprise Systems. *Journal of Object Technology*, pages 55–91, 2008.

[24] A. Wall, J. Kraft, J. Neander, C. Norström, and M. Lembke. Introducing Temporal Analyzability Late in the Lifecycle of Complex Real-Time Systems. In *Procs of RTCSA'03*. Springer Berlin Heidelberg.

[25] S.E. Chodrow, F. Jahanian, and M. Donner. Run-time monitoring of real-time systems. In *Procs of RTSS'91*, pages 74 –83.

[26] M. Saadatmand, A. Cicchetti, and M. Sjödin. Design of adaptive security mechanisms for real-time embedded systems. In *Procs of ESSoS'12*, pages 121–134. Springer-Verlag, 2012.

[27] J. Huselius and J. Andersson. Model Synthesis for Real-Time Systems. In *Procs of CSMR'05*, pages 52–60. IEEE Computer Society.

[28] M. Saadatmand, A. Cicchetti, and M. Sjödin. UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems. In *Procs of ICSEA'11*.

[29] M. Saadatmand, A. Cicchetti, and M. Sjödin. Toward Model-Based Trade-off Analysis of Non-Functional Requirements. In *Procs of SEAA'12*.

[30] B. Grammel and S. Kastenholz. A generic traceability framework for facet-based traceability data extraction in model-driven software development. In *Procs of ECMFA-TW'10*. ACM.

[31] F. Ciccozzi, A. Cicchetti, and M. Sjödin. Exploiting UML Semantic Variation Points to Generate Explicit Component Interconnections in Complex Systems. In *Procs of ITNG'13)*. IEEE CS.

[32] Eclipse Projects. Xpand. http://www.eclipse.org/modeling/m2t/?project=xpand, October 2011.

[33] J. Feljan, J. Carlson, and T. Seceleanu. Towards a model-based approach for allocating tasks to multicore processors. In *Procs of SEAA'12*.

[34] ACPI: Advanced Configuration & Power Interface. http://www.acpi.info/, Last Accessed: January 2013.

Chapter 10

Paper E: Towards Accurate Monitoring of Extra-Functional Properties in Real-Time Embedded Systems

Mehrdad Saadatmand, Mikael Sjödin
The 19th Asia-Pacific Software Engineering Conference (APSEC), Hong Kong, December, 2012.

Abstract

Management and preservation of Extra-Functional Properties (EFPs) is critical in real-time embedded systems to ensure their correct behavior. Deviation of these properties, such as timing and memory usage, from their acceptable and valid values can impair the functionality of the system. In this regard, monitoring is an important means to investigate the state of the system and identify such violations. The monitoring result can also be used to make adaptation and re-configuration decisions in the system as well. Most of the works related to monitoring EFPs are based on the assumption that monitoring results accurately represent the true state of the system at the monitoring request time point. In some systems this assumption can be safe and valid. However, if in a system the value of an EFP changes frequently, the result of monitoring may not accurately represent the state of the system at the time point when the monitoring request has been issued. The consequences of such inaccuracies can be critical in certain systems and applications. In this paper, we mainly introduce and discuss this practical problem and also provide a solution to improve the monitoring accuracy of EFPs.

10.1 Introduction

Successful design of real-time embedded systems depends heavily on how they behave with respect to their extra-functional properties. This is mainly due to the constraints and limitations of these systems in terms of available resources [1]. Therefore, a real-time embedded system needs to achieve its functionality under these limitations and constraints. This implies that the extra-functional properties of the system should remain within an acceptable range and violations in this aspect should be managed and prevented. For example, due to memory constraints, different components of a system should not consume more memory than expected. Similarly, in terms of timing properties, a task may not execute more than its allowed execution time budgets; otherwise, it can affect the overall behavior and functionality of the system.

Monitoring extra-functional properties is an important means in this regard not only to detect and identify violations but also to provide necessary information for runtime adaptation and reconfiguration in systems [2, 3]. However, *accuracy* of the monitored values plays a significant role in the correct identification of violations and also making appropriate decisions for performing runtime adaptation. The term accuracy in this paper is used in the sense that to what extent a monitored value represents the actual state of the system and the extra-functional property that is monitored at a certain time point.

One of the factors that contribute to the accuracy of monitored extra-functional properties is the time difference between the point when a request for monitoring an extra-functional property is issued and the time point when it is actually monitored and its value is obtained. This time difference (referred to as 'monitoring time difference' in this paper) is especially critical in systems where frequent changes of extra-functional properties can happen. An example could be when CPU load in a system changes constantly due to frequent termination and execution of different jobs or when memory usage varies rapidly because of frequent allocation and deallocation of memory. For this reason, it is also deemed necessary to consider timestamps in monitoring of extra-functional properties to be able to judge their accuracy and validity. A point to remember though is that how this time difference affects the accuracy of monitored values can change from one system and even execution scenario to another. For example, in one system, rapid changes in CPU load may happen while memory usage can remain at a certain level. In such a system, the

aforementioned monitoring time difference is critical for validity and accuracy of CPU load measurements while it may not be so for monitoring memory usage. In another system the situation could be the opposite of this case.

In this paper, we focus on this problem and how we can improve the accuracy of monitored values by having more control over the time difference between a request for monitoring and the actual act of monitoring. We introduce an approach which enables to reduce this time difference and thus helps with the accuracy of the monitored extra-functional properties. This is achieved by considering priorities for performing monitoring of different extra-functional properties. The approach works by performing the task of monitoring properties with higher frequency of changes (for which the monitoring time difference can greatly affect the accuracy of the obtained values) at a higher priority level than other properties. We have implemented the approach using OSE Real-Time Operating System (RTOS) [4] which is a commercial RTOS used heavily in telecommunication systems and is embedded in millions of devices around the world.

The remainder of the paper is structured as follows. In Section 10.2 a short introduction to OSE RTOS is provided. Section 10.3 describes the general approach along with its implementation and in Section 10.4, an evaluation of the implemented approach is done. Discussions on several important points regarding the suggested approach is done in 10.5. In Section 10.6 related works are mentioned and finally in Section 10.7 conclusions are made and future directions are explained.

10.2 OSE Real-Time Operating System

OSE is a real-time operating system developed by Enea [4] designed from the ground for use in fault-tolerant distributed systems that are commonly found in telecommunication domain, ranging from mobile phones to radio base stations [4]. It provides preemptive priority-based scheduling of tasks. OSE offers the concept of direct and asynchronous message passing for communication and synchronization between tasks, and OSE's natural programming model is based on this concept. The Interprocess Communication Protocol (IPC) in OSE, allows tasks to run on different processors or cores, utilizing the same message-based communication model as on a single processor. This programing model provides

the advantage of not needing to use shared memory among tasks. The runnable real-time entity equivalent to a task is called *process* in OSE, and the messages that are passed between processes are referred to as *signals* (thus, the terms process and task in this paper can be considered interchangeably). Processes can be created statically at system start-up, or dynamically at runtime. Static processes last for the whole life time of the system and cannot be terminated. Types of processes that can be created in OSE are: interrupt process, timer interrupt process, prioritized process, background process, and phantom process. A process can be in one of the following states: ready, running or waiting. One interesting feature of OSE is that the same programming model based on signal passing can be used regardless of the type of a process. OSE also has a tool for debugging, profiling and monitoring called Optima. Among many other features, Optima enables to monitor properties such as CPU load and number of signals by communicating with a target embedded system using OSE signals.

10.3 Priority-Based Monitoring Approach

10.3.1 Background Scenario

To monitor different properties in the system, a monitoring process has been implemented. In this process, by using the APIs provided by the operating system, values for different properties such as total CPU load, CPU load of a specific process, memory usage (both heap and stack usage), total number of generated signals in the system, number of signals belonging to a certain process, etc. can be monitored. When an application (e.g., a monitoring and profiling tool) needs to obtain the value for any of the properties, it just sends a signal containing a code identifying the desired property to the monitor process. Upon the receipt of this signal, the monitor process which is in waiting state, starts and obtains the property value and sends it back to the requesting process. In this scenario, the monitor process needs to compete with other processes in the system based on their assigned priority levels to obtain CPU and perform its job. Therefore, it is natural that there will be some time difference until the monitor process can successfully obtain the value of a property. This scenario is shown in Figure 10.1. We aim to reduce the time difference which is denoted in the figure by t for properties whose values can change during this time and also this change is considered

important and sensitive.

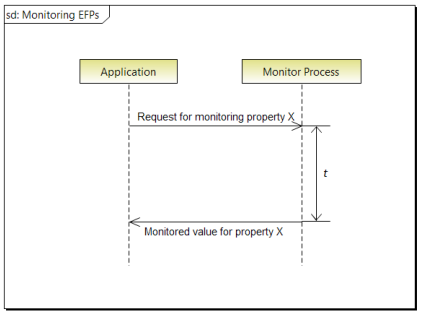

Figure 10.1: Monitoring a property and the time difference between the request for monitoring and the reply

As stated before, if a property in the system changes so frequently then this time difference affects the accuracy of its read value. In this case, if the application requesting this value needs to perform some calculations and make decisions based on the value of the property at the exact time point that it issued the request for monitoring, the accuracy of the monitored value can affect the correctness of such calculations and decisions.

10.3.2 Priorities for Extra-Functional Properties

As described so far, the frequency of change in the value of properties in the system can be different from one property to another. Considering this fact, the user may prefer to fetch the value of a specific property with a shorter monitoring time difference to have better accuracy for it. To capture this need, we let different priority levels be assigned for different properties for performing monitoring. This way, the monitoring of CPU load, for example, can be done at a higher priority level than the monitoring of memory usage. It is achieved by dynamically adjusting the priority of the monitoring process for each property, instead of always invoking the monitoring process at the same fixed priority level. Therefore, the system becomes more flexible and monitoring of different properties does not have to be done at the fixed and default priority level of the monitoring process.

10.3.3 Implementation

The preferred way of communication between processes in OSE is through signals. *Send* and *Receive* APIs are provided in OSE for this purpose. To send a signal, a pointer to the signal structure along with the PID of the receiver is specified as: *send(&sig, pid)*. In our implementation, the signal contains an identifier code to tell the monitor process which priority to monitor, which is defined in the system in the form of an *enum* structure in C/C++.

To allow the specification of priority levels for properties, we have defined a wrapper around the *send* API as: *send_w_prio(&sig, pid, prio)*. What *send_w_prio* does is basically that it first sets the priority of the monitor process (which is in the waiting mode to receive a signal) to the value of *prio*, and then sends to it the rest of the parameters using the normal *send* API. This way, it can be triggered to execute at a lower or higher priority level than its default value.

10.4 Evaluation

To test the suggested approach, a simulation environment has been setup. A monitor process is implemented which waits to receive a signal containing the identifier of a property based on which it then obtains the value for that property from the system and sends back the result to the requester. Another process (referred to as P) is also implemented which constantly changes the value of a dummy property (represented by a variable called N) to a random number and logs it in a file for later inspections and comparisons. The purpose of this process is to simulate a property whose value changes very frequently and constantly in the system. Two additional processes are defined in the system with priority levels below that of process P but above the default priority of monitor process (we call them I_1 and I_2). A shell command (in the OSE shell) called *App* has also been created which when executed sends a request for monitoring the property N to the monitor process.

When a request for monitoring the property N is initiated using the added shell command, the execution of the monitor process is delayed by higher priority processes. This situation is demonstrated in the Table 10.1 where log records corresponding to such cases (extracted from the log files generated by the P and monitor processes) are listed.

The first row in the table, for example, shows that the request for

MRTP* (μs)	**Monitoring Done at** (μs)	**Monitored Value**	**Value at MRTP***
5140013	5150001	838	3200
6630025	6630026	5600	5600
7350011	7360001	5883	9600
13460011	13470001	5032	1600

Table 10.1: Discrepancy between property values at monitoring request time and when actual monitoring is done.
(* MRTP:Monitoring Request Time Point - from the startup of the system)

monitoring property N is issued at 5140013μs (from the start of the system). Due to the interferences from the two other processes (I_1 and I_2) in the system, the actual monitoring is performed at time point 5150013μs which results in obtaining 838 as the value for the property (remember that when P executes, it repeatedly generates a random number and sets it as the value of N). However, by consulting the generated log file from process P, we can see that the actual value of the property at the time when monitoring request has been issued was 3200.

In the next step, we initiate again a request for monitoring but this time with a priority higher than those of I_1 and I_2 processes. Part of the result from the generated log files is shown in Table 10.2.

MRTP* (μs)	**Monitoring Done at** (μs)	**Monitored Value**	**Value at MRTP***
5970013	5970014	6804	6804
8630001	8630002	6698	6698
12060001	12060002	3629	3629
14500018	14520000	7673	9600
21590011	21590012	6400	6400

Table 10.2: Property values using flexible priority levels for the monitor process.
(* MRTP:Monitoring Request Time Point - from the startup of the system)

From the result of the second case, we can see that more instances of monitoring have managed to obtain accurate results for the property. The discrepancies which are still observed in this scenario, such as the second row from the bottom of Table 10.2, are due to the re-activations of process P which still has higher priority than the monitor process (the 1μs difference between the monitoring request time point and actual time of performing monitoring is the natural increment of internal clock of the system).

10.5 Discussions

In the previous section, it was shown how by allowing to assign priority levels for monitoring of different properties it becomes possible to achieve better accuracy. However, there are some important points that need careful attention. If the priority of the monitor process is increased, it also means that other processes in the system can suffer delays due to preemption by the now higher priority monitor process. The consequences of such delays are of course dependent on the nature and responsibilities of those processes in the system. Therefore, the user initiating a monitoring request at a high priority level should also consider its consequences.

Another factor that contributes to the monitoring time difference and thus accuracy of monitored properties is the time length of the monitoring process itself. Implementation efficiency of the monitor process can thus help with the reduction of monitoring time difference.

A point which can be specific to our implementation is that the shell daemon in OSE is a system process which also has an assigned priority level. This can be important for knowing how the shell command that we implemented is interpreted by the shell daemon and executed and how the daemon behaves in terms of getting CPU time to execute comparing its priority level to other processes in the system. In our settings, since the priority of the shell daemon was the same default value in both cases, its effect on monitoring was simply ignored.

Here we mainly discussed the frequency of change in the value of a property. Another aspect that can also be added to the picture is the magnitude of such changes and its importance for accurate monitoring. For example, the value of a property can change quite frequently in small increments from 1.0 to 5.0. But depending on the use of the monitored value, the difference between 1.45 and 1.80 may not be significant at all. Therefore, deciding for which properties we need more accurate monitoring is dependent on the use of the monitored values as well as the importance, tolerance and negligible fluctuations in the value of those properties. Moreover, in this paper we did not discuss how exactly the values of different extra-functional properties are calculated and determined. For some properties such as CPU load or the number of signals (in case of OSE and signal-based systems) it may be straightforward to obtain such values, while for some other properties such as reliability, it might be very cumbersome and complicated. This topic, however, is out

of the scope of the paper.

10.6 Related Work

The work done in [5] discusses the issue of accuracy in monitoring performance of a system using the hardware counters and registers that are available on modern microprocessors. One of the differences between this work and ours is that it compares the accuracy of monitoring using hardware counters versus hardware sampling while we basically discussed the issue of accuracy by targeting the monitoring time difference. It also touches on the effects of adding monitoring features by acknowledging the fact that "as in any physical system, the act of measuring perturbs the phenomenon being measured" [5]. [6] which introduces a very interesting approach to reduce energy consumption in real-time distributed embedded systems, defines the concept of monitoring intervals for cyclically monitoring the system to identify and exploit dynamic slacks. It discusses the overhead of monitoring and how to find an optimal interval value for it. However, it does not deal with the accuracy of monitoring results versus constant changes in the state of the system.

As for the application of monitoring, [7] serves an example for the use of monitoring results where models of real-time systems are synthesized based on the monitoring information that is collected. In [3], we have introduced and implemented an adaptive approach using monitoring information about timing behaviors of encryption algorithms to balance security and timing requirements in real-time systems. Regarding the implementation of monitoring mechanisms, in [2], we have discussed the challenges and practicalities of monitoring timing properties in real-time systems, especially in industrial RTOSes, and provided a solution to improve monitoring of real-time events. The work in [8] provides a framework for monitoring of timing properties and a model for the specification of timing constraints. It provides two general approaches for synchronous and asynchronous monitoring of real-time constraints. The disturbances of adding monitoring features and issues related to *probe-effect* are discussed in detail in [9].

10.7 Conclusion and Future Work

In this paper, we discussed the issue of accuracy in monitoring EFPs and its importance in real-time embedded systems. Mainly, the monitoring time difference has been discussed as an important factor that contributes to the accuracy of monitoring results. An approach was introduced to reduce this time difference. An implementation of the approach was also provided and it was evaluated and shown how the approach reduces the monitoring time difference.

By implementing the approach as part of OSE and Optima, more flexibility can be provided for monitoring the system in terms of accuracy levels, and also regarding the inclusion of timestamps for monitored values, the users can get a better idea about the monitoring time difference. One point to remember is that adding monitoring features bring along their own costs and effects which should also be taken into account in the design of systems. In our evaluation example, the monitoring request was always initiated with a priority less than the priority of the main process P. If not so, the monitoring could preempt process P which could represent a main task in the system such an engine control task, and thus impair the functionality of the system. Therefore, there is a type of trade-off between the monitoring accuracy and other functions of the systems. Considering this issue especially on multi-core systems could be an interesting direction of this work.

As a continuation of this work, we are working on building a monitoring framework to provide a flexible environment to the user to tune the monitoring of different properties based on different factors that contribute to its accuracy. Towards this goal, we are also investigating other possible factors that can affect the accuracy of monitoring results.

10.8 Acknowledgements

This work has been supported by the CHESS European Project (ARTEMIS-JU100022) [10] and XDIN AB [11] through the ITS-EASY program.

Bibliography

[1] Thomas Henzinger and Joseph Sifakis. The Embedded Systems Design Challenge. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg.

[2] Mehrdad Saadatmand, Mikael Sjödin, and Naveed Ul Mustafa. Monitoring Capabilities of Schedulers in Model-Driven Development of Real-Time Systems. In *17th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, September 2012.

[3] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. Design of adaptive security mechanisms for real-time embedded systems. In *Proceedings of the 4th international conference on Engineering Secure Software and Systems*, ESSoS'12, pages 121–134, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] Enea. The Architectural Advantages of Enea OSE in Telecom Applications. `http://www.enea.com/software/products/rtos/ose/`, Last Accessed: May 2012.

[5] Michael E. Maxwell, Patricia J. Teller, and Leonardo Salay. Accuracy of performance monitoring hardware. In *Proceedings of the LACSI Symposium, Sante Fe, New Mexico*, 2002.

[6] Subrata Acharya and Rabi Mahapatra. A Dynamic Slack Management Technique for Real-Time Distributed Embedded Systems. *IEEE Trans. Comput.*, 57(2):215–230, February 2008.

[7] Joel Huselius and Johan Andersson. Model Synthesis for Real-Time Systems. In *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, CSMR '05, pages 52–60, 2005.

[8] S.E. Chodrow, F. Jahanian, and M. Donner. Run-time monitoring of real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 74 –83, dec 1991.

[9] Henrik Thane. Design for Deterministic Monitoring of Distributed Real-Time Systems. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-23/2000-1-SE, Målardalen University, May 2000.

[10] CHESS Project: Composition with Guarantees for High-integrity Embedded Software Components Assembly. `http://chess-project.ning.com/`, Last Accessed: June 2012.

[11] XDIN AB. `http://ny.xdin.com/om-xdin/enea-experts/`, Accessed: June 2012.

Chapter 11

Paper F: 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 (SEAA), Verona, Italy, August, 2014.

Abstract

Architectural models, such as those described in the EAST-ADL language, represent convenient abstractions to reason about automotive embedded software systems. To enjoy the fully-fledged advantages of reasoning, EAST-ADL models could benefit from a component-aware analysis framework that provides, ideally, both verification and model-based test-case generation capabilities. While different verification techniques have been developed for architectural models, only a few target EAST-ADL. In this paper, we present a methodology for code validation, starting from EAST-ADL artifacts. The methodology relies on: (i) automated model-based test-case generation for functional requirements criteria based on the EAST-ADL model extended with timed automata semantics, and (ii) validation of system implementation by generating Python test scripts based on the abstract test-cases. The scripts represent concrete test-cases that are executable on the system implementation. We apply our methodology to analyze the ABS function implementation of the Brake-by-Wire system prototype.

11.1 Introduction

The complexity of embedded systems in the automotive domain is continuously increasing, in part due to the replacement of mechanical or hydraulic technologies with electrical/electronic systems that implement complex functions, such as cruise control, automatic braking, etc. Consequently, the system development needs to conform with stringent safety standards. To align with such standards, the development process must provide evidence that requirements are satisfied at each level of system abstraction, from architectural and behavioral models to implementation (e.g., as required by ISO 26262). In this context, there is a real need for advanced and formal methodologies for verification and testing of automotive systems, which can provide industrially relevant artifacts.

Although there is a solid research know-how of generating test-cases from behavioral specification models [1, 2, 3], in principle these methods are not directly applicable to architectural models where the system behavior is defined in terms of function blocks with no formal support to specify and analyze their internal behaviors. The latter is usually described in semi-formal languages such as UML, Simulink etc. The operation of each function block can, for instance, be formalized using notations such as timed automata (TA) [4], which could serve as the semantic representation of the block behavior [5]. Assuming this, the abstract test-case generation can then be carried out using component-aware model-checking algorithms [6]. The resulting abstract test-cases contain internal state information not corresponding to the actual code. Hence, the abstract test-cases need to be transformed into executable scripts that would then be used as concrete test-cases to analyze the system implementation. This need has kindled our motivation to introduce a methodology (see Section 11.4) for model-based testing against functional requirements of embedded systems, starting from the EAST-ADL architectural models, an emerging standard for automotive industry, already used by Volvo Group Trucks Technologies, Sweden. As part of the methodology, we show how to generate executable test-cases for the system implementation automatically, starting from abstract tests generated by model-checking EAST-ADL high-level artifacts extended with TA behavior. The main goal of this paper is to check the feasibility of the EAST-ADL+TA generated abstract test-cases by actually running the corresponding executable test-cases on the implemented code, in an attempt to obtain a *pass* or *fail* verdict. If the endeavor succeeds, the

testing effort of the code could then be reduced by the automatic provision of valid test-cases.

Our contribution assumes three actors in the system development process: the System Designer, the Developer, and the Tester. The methodology presented in this paper describes only two main phases: model-based system implementation and testing. Concretely, we adopt ViTAL [7] as our main modeling and analysis framework, as it integrates component-aware model-checking with EAST-ADL models. In Section 11.5 we define an executable semantics of the UPPAAL PORT TA that facilitates code implementation (see Section 11.5) in a semantics-preserving manner. Next, we we show how to generate abstract test-cases for functional requirements, from the TA model of the EAST-ADL system description (see Section 11.6.1). The functional requirement criterion is formalized as a reachability property in UPPAAL PORT [8], and the result is an abstract test-case defined by an execution trace of the TA models corresponding to the function blocks. In Section 11.6.2 we transform the states and transitions of the test-case into C/C++ code signals, by generating Python test-scripts in Farkle test execution environment [9]. These test-scripts are run against the system under test (SUT) to obtain a pass or fail verdict w.r.t. the testing goal. Our framework adapts already existing model-checking based testing techniques, to obtain a novel integrated approach for testing automotive embedded systems, starting from high-level system artifacts modeled in EAST-ADL, and their requirements specification. To check the applicability of our framework, we illustrate it on a simplified version of Volvo's Brake-by-Wire System prototype, which we describe in Section 11.3. We compare to related work in Section 11.8, and summarize our work, together with outlining ideas for future work in Section 11.9.

11.2 Preliminaries

In this section we give a brief overview of: (i) ViTAL, used for generating abstract test-cases, and (ii) Farkle, used for transforming the latter into test-scripts.

11.2.1 ViTAL

ViTAL, a Verification Tool for EAST-ADL Models using UPPAAL PORT, integrates architectural languages and verification techniques to provide

simulation and model-checking of timing and functional behavioral requirements. To achieve this, the tool provides functional and timing behavior for EAST-ADL functional blocks using timed automata semantics, and performs an automatic model transformation to the input language of UPPAAL PORT, which enables the model-checker to handle EAST-ADL models for formal verification.

The tool is an integrated environment based on Eclipse plug-ins and contains an editor for the EAST-ADL model (i.e., Papyrus), an editor for timed automata description of the behavior of EAST-ADL blocks, and a semantic mapping between each EAST-ADL block and a corresponding TA model (e.g., mapping internal TA variables to EAST-ADL external ports). The result of the transformation is compliant to the input language of the UPPAAL PORT model-checker, able to simulate the system model and verify various requirements (e.g., functional, timing), specified in Timed Computation Tree Logic (TCTL). ViTAL integrates the following artifacts:

EAST-ADL EAST-ADL is an architecture description language dedicated to the development of automotive embedded systems [10]. The definition of an EAST-ADL system model is given at five levels of abstraction representing different stages of the engineering process with complete traceability between them. At each level, the behavioral description relies on the definition of a set of function prototypes (refereed as blocks) f_ps, executed assuming the "read-execute-write" semantics. A block starts executing by reading data, which are constantly replaced by fresh data arriving on ports, performs some calculation and finally outputs data on the output ports.This enables analysis, behavioral composition, and makes the function execution independent of its internal behavior. The functionality of each f_p is defined using different notations and tools, e.g., Simulink or UPPAAL PORT TA in ViTAL.

UPPAAL PORT TA Component As depicted in Figure 11.1, an UPPAAL PORT [6] component is defined by its interface and its timed behavior. The interface consists of a set of input data ports, a set of output data ports, and a set of trigger ports that define the control flow. The timed behavior is modeled as a tuple:

$$B = (N, l_0, l_f, V_D, V_C, r_0, r_f, Ed, I) \tag{11.1}$$

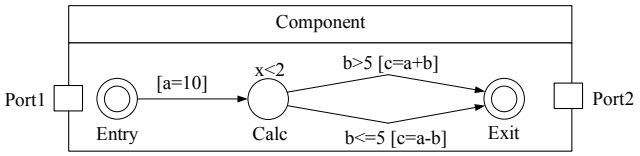

Figure 11.1: UPPAAL PORT TA Component.

where N is a finite set of locations, l_0 is the initial location, l_f is the final location, V_D and V_C are sets of data and clock variables, respectively, r_0 and r_f are sets of initial and final clock resets, and Ed is a set of edges. The function $I : N \cup \{l_0, l_f\} \rightarrow B(V_C)$, with $B(V_C)$ denoting the set of conjunctive formulas of clock constraints of the form $x_i \sim m$, or $x_i - x_j \sim n$, $x_i, x_j \in V_C$, $\sim \in \{\leq, <, =, \geq, >\}$, $m, n \in N$, assigns each location $l \in N \cup \{l_0, l_f\}$ to its invariant $I(l)$. To describe an edge from location l to l', with guard g, update action e, and clock resets r, we write $l \xrightarrow{g,e,r} l'$, for $(l, g, e, r, l') \in Ed$.

The timed behavior associated to the UPPAAL PORT TA component depicted in Figure 11.1 has three locations *Entry*, *Calc*, and *Exit*, three data variables a, b, and c, and a clock variable x. When the execution of the TA is triggered, it enters the *Entry* location, updates variable a to 10 along the first edge to location *Calc*, where it remains as long as the invariant $x < 2$ holds. The update of variable c is determined based on the evaluation to "TRUE" of guards $b > 5$ and $b <= 5$, placed on the edges to location *Exit*.

The semantics of an UPPAAL PORT TA component are defined as a state: (l, u, v), where l is a location, v is a data valuation, and u is a clock valuation. UPPAAL PORT [6] allows the following transitions from one state to another:

- internal transitions: $(l, u, v) \xrightarrow{\tau} (l', u', v')$, along an edge $l \xrightarrow{g,e,r} l'$,
- delay transitions: $(l, u, v) \xrightarrow{\delta} (l, u, v + \delta)$ where $\delta \in \mathbb{R}_{\geq 0}$,
- read transitions: $(idle, u, v) \xrightarrow{read} (l_o, input(u), [r_0 := 0]u)$ if $triggered(v)$,
- write transitions, $(l_f, u, v) \xrightarrow{write} (idle, output(u), [r_f := 0]u)$.

UPPAAL PORT Model-checker UPPAAL PORT is an extension of the UPPAAL tool, which supports simulation and model-checking of component-based systems, without the usual flattening of the TA network [8]. This is complemented by the **P**artial **O**rder **R**eduction **T**echnique (PORT) that improves the efficiency of analysis by exploring only a relevant subset of the state-space when model-checking. The tool also uses local time semantics [11] to increase independence, being suited for the analysis of "read-execute-write" component models.

11.2.2 Farkle

Farkle is a test execution environment that enables testing an embedded system in its target platform. It uses LINX as the Inter-Process Communication (IPC) protocol to provide direct and asynchronous message passing between tasks. This allows tasks to run on different processors or cores, while utilizing the same message-based communication model as on a single processor, but without using shared memory. The messages that are passed between processes (i.e., tasks) are referred to as *signals*. An example signal definition is shown in Figure 11.2.

```
1 #define WHEEL_SPEED_SIG 1026
2 typedef struct WheelSpeedSignal{
3         SIGSELECT sigNo;
4         float WheelSpeed;
5 } WheelSpeedSignal;
```

Figure 11.2: Signal example

Using the signal passing mechanisms of LINX, Farkle runs on a host machine and communicates with the target. Hence, Farkle enables testing an embedded system by providing certain inputs to the target in the form of signals and receiving the result as signals containing output values. The test-scripts that are used to send and receive signals, and also decide the verdict of a test-case are implemented in Python. Moreover, in order for the signal passing mechanism to work between the host and target, the host needs to also have information about signal structures. For this purpose, Farkle also generates signal definitions in Python from the signal definitions of the application source code, which is then imported and used in the Python test-script.

11.3 Brake-by-Wire Case Study: Functionality and Structure

Through the paper we use the Brake-by-Wire (BBW) system as a running example. Figure 11.3 shows the EAST-ADL model of the BBW system at the analysis level. To simplify, we have modeled only two out of the four wheels of the system. The **Brake Pedal Sensor** reads the position of the pedal and the **Brake Torque Calculator** computes the desired braking force based on this value. Similarly, the **Wheel Sensor** reads the rotation speed of the wheel. The **Global Brake Controller** calculates the actual braking force by updating the desired braking force based on the speed of the wheels, and provides it to the **ABS** block, which calculates the slip rate to decide if the braking force can be applied without locking the wheel. Finally, the braking force is applied by the **Wheel Actuator**.

The **ABS** f_p calculates the slip rate s based on the equation:

$$s = (v - w \times R)/v, \tag{11.2}$$

where w is the rotation speed of the wheel, v is the speed of the car, and R is the radius of the wheel. The friction coefficient of the wheel has a nonlinear relationship with the slip rate: when s starts increasing, the friction coefficient also increases, and its value reaches the peak when s is around 0.2. After that, further increase in s reduces the friction coefficient. For this reason, if s is greater than 0.2 the brake actuator is released and no brake is applied, or else the requested brake torque is used. Our goal is to test whether the actual system implementation meets this functional requirement.

Figure 11.3: The EAST-ADL model of the BBW system.

11.4 From EAST-ADL to Code Validation: Methodology Overview

This section overviews our model-based testing (MBT) framework, which allows test-case generation starting from EAST-ADL models, down to their execution on the system under test (SUT). This framework follows the MBT methodology [12], and it is implemented by a tool chain consisting of ViTAL and Farkle, as depicted in Figure 11.4.

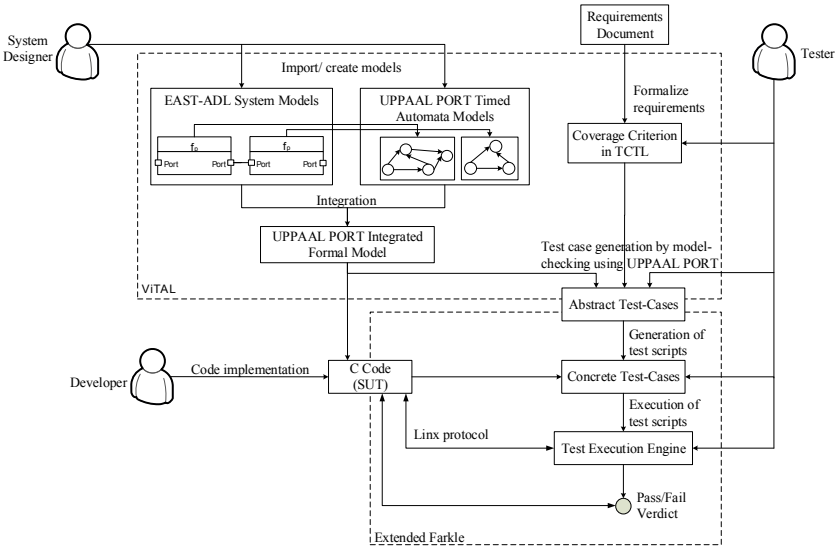

Figure 11.4: From ViTAL to Farkle: The Methodology

We assume three actors in the process: the System Designer, the Developer, and the Tester. Their roles are explained below.

The **System Designer** performs the following actions:

- Imports the EAST-ADL model and creates the associated TA behavior to each EAST-ADL f_p.
- Performs an automatic transformation from the EAST-ADL + TA models to the input language of UPPAAL PORT, in ViTAL. The

result is the integrated abstract formal model used for formal verification and test-case generation by means of model-checking.

The **Developer** implements the code (here the SUT) manually, based on the system's integrated abstract formal model (for which we define an executable semantics).

The **Tester** performs the following actions:

- Formalizes the system requirements manually into TCTL properties, the query language of UPPAAL PORT. Each requirement represents a testing goal, whereas their collection is our coverage criterion.
- Generates abstract test-cases with UPPAAL PORT for the integrated formal model, against the above formalized criterion.
- Converts the abstract test-cases into concrete test-cases automatically, by generating Python test-scripts executable by Farkle.
- Executes the concrete test-cases against the SUT, to obtain a *pass* or *fail* verdict, and also code-related information (e.g., variable values).

The activities performed by the System Designer are part of our previous work, and for further details we refer the reader to our previous publication [7]. The next sections provide details on semantics preserving code implementation (see Section 11.5), as well as our method for test-case generation from EAST-ADL models with TA semantics, up to test-case execution on the SUT (see Section 11.6).

11.5 Implementation Activities

The implementation is an important, labor intensive, and error prone phase in the development process of any software system. To ease the implementation process, we are interested in providing guidelines that could help the developers to implement C code, based on the EAST-ADL system models extended with TA semantics. For this, we introduce an executable semantics for UPPAAL PORT TA that could serve as the basis of future code synthesis.

11.5.1 Executable Semantics of UPPAAL PORT TA

In principle, the TA behavior of an EAST-ADL block can be non-deterministic. To obtain an implementation of the EAST-ADL component whose behavior is modeled as UPPAAL PORT TA, we need to define its deterministic semantics that needs to be obeyed by the code. For this, we adapt the approach proposed by Amnell et. al [13] for task automata code synthesis, to UPPAAL PORT TA.

Similar to ordinary timed automata, the semantics of an UPPAAL PORT timed automaton is given in terms of a labeled transition system. Assume that the set of V_D variable valuations is ranged by v, the set of V_C clock valuations by u, and l stores the automaton's current location, $l \in N \cup \{l_0, l_f\}$. In addition, we say that a transition $trs = (l \xrightarrow{g,e,r} l')$ is enabled in state $s = (l, u, v)$, denoted by Enabled(trs,s), when its guard holds, that is, $u, v \models g$.

The non-determinism of the semantic representation of the UPPAAL PORT TA stems from the internal, read, or write actions, as well as from time-delays. As in previous work on TA, we resolve non-determinism, as follows: (i) Let $Pr : E \mapsto \mathbf{N}$ be a function that assigns unique priorities to each edge in the UPPAAL PORT TA. If several transitions are enabled in a state, the function Pr establishes the order in which the transitions are fired. This resolves the action non-determinism; (ii) Time non-determinism is resolved by implementing the *maximal progress assumption* [14], in which delay transitions are forbidden if an action transition is enabled. The TA should fire all the enabled transitions until no enabled transition exists anymore.

In the following, we write $(l, u, v) \xrightarrow{e,r} (l', u', v')$ for a state-changing *discrete transition* (internal, read, or write) on which update actions in form of assignments e, or clock resets r, occur. In case of a *delay transition* that does not result in a state-change, we write $(l, u, v) \xrightarrow{t} (l, u', v')$, where $u' = u + t$, and $(u + t) \models I(l)$ holds.

Definition 1 (Deterministic Semantics)**.** *Let* $B = (N, l_0, l_f, V_D, V_C, r_0, r_f, Ed, I)$ *be a* UPPAAL PORT *TA behavior of an EAST-ADL component. Assuming a function Pr that assigns priorities to TA edges, the deterministic semantics of the component's behavior is a labeled transition system defined by the following rules:*

- $(l, u, v) \xrightarrow{e,r} (l', u', v')$ *if* Enabled$(l \xrightarrow{g,e,r} l', (l, u, v))$*, and there is no*

edge $\in Ed$ *such that* $Pr(edge) > Pr(l \xrightarrow{g,e,r} l')$ *and* $\mathsf{Enabled}(edge, (l, u, v))$*;*
• $(l, u, v) \xrightarrow{t} (l, u + t, v')$ *if* $(u + t) \models I(l)$*, and for all edge* $\in Ed$ *and* $d < t$*,* $\neg\mathsf{Enabled}(edge, (l, u + d, v))$*.*

The above definition ensures conformance of the implementation to the high-level behavioral model, since the behavior defined by the deterministic semantics is a subset of the UPPAAL PORT TA behavior. Hence, all the transition sequences possible in the (deterministic) implementation model are also possible in the original (possibly non-deterministic) one, thus guaranteeing preservation of the safety properties of the EAST-ADL behavioral TA model. However, the code generation is not automated yet. Even if the latter were achieved, the code might still need human intervention in implementing primitives or aggregations that would improve its performance. Hence, testing the code itself cannot be avoided.

11.5.2 Implementing the System Model

We approach the problem of code implementation as a mapping activity between the EAST-ADL system model extended with TA behavior and C code. We propose a simple 1-to-1 mapping to code elements starting with the elements of the EAST-ADL model focusing on: (i) components, (ii) ports, (iii) connectors and (iv) triggering information. The mapping is defined in Table 11.1.

Table 11.1: Mapping EAST-ADL f_p's interface to C code

EAST-ADL f_p	**C Code**
Component (block)	C function
Port (input and output)	Buffer of size 1
Connector	Connection between buffers
Triggering information	Function calls and time interrupts

This mapping provides guidelines for the structure of the C code, which should also conform to the semantics of Definition 1. For instance, for each f_p in the EAST-ADL model we will create a C function implementing its behavior. Since the input and output data-flow ports

can store only the latest value of the corresponding variable, the implemented C function will have a buffer of size 1 for each port. This implies that the connection between a block's output ports and another one's input ports is translated into a link between buffers. The triggering information for each f_p is implemented as a function call or a time interrupt. However, some of the features of the EAST-ADL model are lost at the implementation level. For instance, the C code does not respect the "read-execute-write" semantics of the model.

The C code inside each function is implemented based on the TA behavior. Each element in the TA tuple is mapped to code elements according to Table 11.2.

Table 11.2: Mapping TA behavior to C code

TA model	**C code**
data variables	variables
clock variables	variables of type long
locations	state variables
clock reset (initial and final)	assignment to zero
invariant	while loop
enabled action selection	if/case statement
action update	assignment

Based on these rules and Definition 1, a complete implementation of the system can be obtained. The implementation conforms with the model. This conformance is an important aspect in model-based testing, as the formal model and the SUT need to be in close relation for the abstract test-cases to really aid the generation of executable test-cases, in a meaningful way.

11.6 Testing Activities

In model-based testing, the formal model is a faithful yet abstract representation of the intended system, based on requirements and specification, and describes rigorously the intended behavior using formal modeling notations. In our framework, we employ ViTAL to create the formal model starting from the architectural representation of the system in EAST-ADL and describe the behavior of each f_p as a UPPAAL

PORT TA model using the specific TA semantics. In this section, we use such a model to generate abstract test-cases with UPPAAL PORT automatically, followed by their automatic conversion to Python scripts, and their execution on the SUT.

11.6.1 Generation of Abstract Test-Cases in ViTAL

ViTAL employs UPPAAL PORT to automate the abstract test-case generation and takes as input: (i) the abstract formal model represented by the "UPPAAL PORT compliant" EAST-ADL model with TA semantics, and (ii) a testing goal, i.e., a functional requirement of the system (or a collection of such requirements) formalized as a TCTL reachability property (properties).

UPPAAL PORT is a model-checker designed for the simulation and the formal verification of component-based embedded systems, and is not tailored for test-case generation. However, we exploit its ability to automatically generate witness traces for *reachability* properties specified in TCTL. Such properties are encoded as $E <> q$, where E represents the existential path quantifier, $<>$ is the temporal operator, and q is the goal state. The property can be read as follows: there exists an execution path such that, eventually, the state q is reached. The test goal guides the generation of a witness trace from an infinite number of possible executions of the system.

The witness trace is a sequence of states and transitions, and it represents the abstract test-case (ATC) for the test goal specified by the reachability property:

$$ATC \triangleq (l_0, u_0, v_0) \xrightarrow{a_0} (l_1, u_1, v_1) \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} (l_n, u_n, v_n)$$

In the above, the state of the system is defined by the current locations l_i, data valuations u_i and clock valuations v_i in the TA network that describes the system behavioral model. The transition actions a_j represent either delays, or internal TA transitions, or the special read/write transitions from/to ports, respectively.

For each test goal, the UPPAAL PORT model-checker generates only one trace representing the execution of the system from its initial state to the goal state encoded by the reachability property. Such a trace represents our abstract test-case with respect to a particular system requirement. A collection of such abstract test-cases is provided to Farkle, for further transformations and execution on the code.

11.6.2 Generation and Execution of Concrete Test-Cases in Farkle

The abstract test-cases generated by ViTAL are provided as input to the Extended Farkle environment (i.e., Farkle plus Parser, etc.). The abstract test-cases are parsed and the order of states and transitions, along with the values of variables in each state, are identified. Based on this information a test-script is created. The test-script basically creates signals with values of variables at each state extracted by parsing the abstract test-cases, which are then sent to the target system. The initial values of variables that are sent to the target system by the test-script, in form of signals, trigger the SUT to execute and evolve through a set of states and transitions. The information about the actual set of states and transitions taken by the SUT during execution is collected and sent back to the script (again in form of signals). An important contribution here is that we enable tracking of state changes at runtime by implementing the code based on the formal models. In other words, a switch-case structure is used and some variable keeps track of the current system state, and changes accordingly when moving to a different state.

The test-script receives the result, that is, the information on the set of states and transitions, as well as the values of variables at each state. This information, collected during execution and at runtime, is then used to compare the actual order of states and transitions against the expected one originated from the models and specified in the abstract test-case. If any discrepancy between the actual and expected orders of traversed states and transitions (with the option of also checking the expected values of variables) is found, the test result is evaluated to *fail*, otherwise a *pass* verdict is issued. In other words, it is checked that, based on the given inputs, the *exact* same order of states as in the trace and abstract test case appears at runtime, during the execution of the system.

In the above steps, the generation of executable test-scripts is based on the following principles:

• For each variable (in the UPPAAL PORT TA), we create an array containing all of its expected values, respectively, at each state and in the exact same order as it appears in the abstract test-case, as follows: <statemachine_name>_<variable> = [x,..,y].

• The initial values of variables are used in the structure of a signal, which is then sent to the target.

• We define an additional array to preserve the order of states, in the form of: <statemachine_name>_state = [x,..,y]; e.g., line 18 in Figure 11.8, where the numbers serve as IDs, and each represents a unique state in the automaton, respectively.

• Based on the number of states, we create a loop in the script.

• Inside the loop, we add assertion statements for each variable, e.g., line 28 in Figure 11.8, to check its expected value at the current state versus its received value at that state, which is retrieved from the log information sent back from the target (in the form of a signal), respectively.

• An additional assertion statement is used in a similar way, for checking the actual order of visited states in the code, versus the expected ones in the model.

Basically, this allows to verify the internal state of the system, and to determine whether it is behaving as expected (as specified in the models) or not. Moreover, it makes it possible to determine exactly between which states a deviation from the expected behavior has occurred. This mechanism provides a *defect localization* feature as well. In other words, testers can get some insight into the vicinity of a problem in the code, which can ease debugging and fixing that respective problem.

11.7 Brake-by-Wire Revisited: Applying the Methodology

We illustrate and exercise the applicability of our approach on the BBW system, introduced in Section 11.3.

11.7.1 Creating the formal model

In ViTAL, we have imported the EAST-ADL model described in Section 11.3 and created nine TA describing the behavior of each EAST-ADL f_p, respectively. Figure 11.5 shows the behavior of the **pABSFL** f_p as an UPPAAL PORT TA model.

The functionality of the timed automaton is described as follows. First, the speed of the car is evaluated; if the car has no speed then no brake force is applied which corresponds to transversing the edge annotated with $v == 0[torqueABS = 0]$, otherwise the slip rate is evaluated. If the slip rate exceeds 0.2, no braking force should be applied

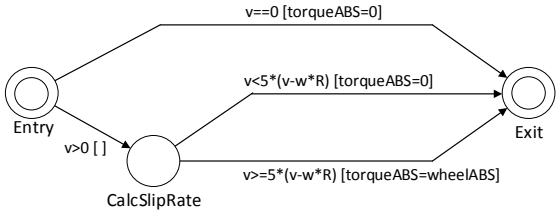

Figure 11.5: The TA description of the ABS function.

to not block the wheel. In our TA model, we are evaluating $s > 0.2$ as $v < 5 \times (v - w \times R)$.

Table 11.3: Mapping TA variables to EAST-ADL ports

TA variable	**EAST-ADL port**
w	WheelSpeed_rpm_FL
wheelABS	RequestedTorque_FL
torqueABS	ABSTorque_FL
v	VehicleSpeed_kmph_FL

In ViTAL, the TA local variables need to be mapped to the EAST-ADL ports shown in Figure 11.3. This mapping is presented in Table 11.3. Next, ViTAL performs an automatic transformation to the input language of UPPAAL PORT. At this point, we can use the UPPAAL PORT model-checker to simulate and formally verify the model against its requirements. Once the correctness of our model is ensured, we start generating test-cases.

11.7.2 Code implementation

Based on the guidelines of Section 11.5, we have implemented the code for the BBW system. A section of the code, depicting the functionality of the ABS component is shown in Figure 11.6.

Each location in the TA model depicted in Figure 11.5 represents a possible value of the variable *state*, which is initially set to *Entry*. While *state* is different from *Exit*, the code implements all the possible computations of the TA, e.g., if $v == 0$ then *torqueABS* is set to zero.

```
1  void mbatAbs_calc(MbatAbsInput* input, void* hdl)
2  { state = Idle;
3    /* Internal variables of automaton */
4    float s;
5    /* Output variables */
6    U32 TorqueABS=0;
7
8    state = Entry;
9    while(state != Exit) {
10     switch(state) {
11       case Entry: {
12         if(input->v > 0) {state=CalcSlipRate; }
13         else
14           if(input->v == 0) { TorqueABS=0; state=Exit; }
15           else { // Error }
16         break; }
17       case CalcSlipRate: {
18         s = (float)(input->v-input->w*input->R)/input->v;
19         if(s > 0.2) { TorqueABS=0; }
20         else { TorqueABS=input->WheelABS; }
21         state = Exit;
22         break; }
23       case Exit: { break; } }
24     printf("  Tracing ABS calculation state:%d\n",
25         state);
26     mbatAbs_transition(state, input->w, input->WheelABS,
27             input->v, TorqueABS, input->R,
28             (MBAT_TRC*)hdl);
29     state = Idle; } }
```

Figure 11.6: Implementation of the ABS component

Note that time is not considered in this implementation, so all transitions are taken instantly.

11.7.3 Testing goal

In this paper, we focus on one of the requirements of the ABS function, which states that: "*If the brake pedal is pressed and a wheel has a slip rate > 20%, then the brake torque for that wheel should be set to 0* N/m^2".
The requirement is expressed in TCTL as follows:
$E <> (BrakePedalSensor.pos > 0 \; and \; ABS.v < 5\times$
$(ABS.v - ABS.w \times ABS.R) \; and \; WheelActuator.NoBrake)$

11.7.4 Abstract test-case generation

As presented in Section 11.6.1, we employ UPPAAL PORT to generate abstract test-cases from the abstract model previously constructed. The model-checker takes as input the formal model together with the test goal specified as the TCTL reachability property above.

```
State:(ABSFL.idle)
    ABSFL.w=0 ABSFL.wheelABS=0 ABSFL.torqueABS=-1
    ABSFL.v=0 ABSFL.R=1/2
Transitions: ABSFL.idle->ABSFL.Entry { w:= 8, wheelABS:= 1,
    v:= 12}
State:(ABSFL.Entry)
    ABSFL.w=8 ABSFL.wheelABS=1 ABSFL.torqueABS=-1
    ABSFL.v=12 ABSFL.R=1/2
Transitions: ABSFL.Entry->ABSFL.CalcSlipRate { v> 0}
State:(ABSFL.CalcSlipRate)
    ABSFL.w=8 ABSFL.wheelABS=1 ABSFL.torqueABS=-1
    ABSFL.v=12 ABSFL.R=1/2
Transitions: ABSFL.CalcSlipRate->ABSFL.Exit
    { v< 5* (v- w* R), torqueABS:= 0 }
State: (ABSFL.Exit)
    ABSFL.w=8 ABSFL.wheelABS=1 ABSFL.torqueABS=0
    ABSFL.v=12 ABSFL.R=1/2
Transitions: ABSFL.Exit->ABSFL.idle { }
State:(ABSFL.idle)
    ABSFL.w=8 ABSFL.wheelABS=1 ABSFL.torqueABS=0
    ABSFL.v=12 ABSFL.R=1/2
```

Figure 11.7: Abstract Test-Case

The UPPAAL PORT model-checker generates the witness trace presented in Figure 11.7 automatically. The trace represents the execution of the pABSFL f_p. Initially, the TA is in location *idle* and all variables are zero. The first transition to state *Entry* is a *read* transition, where the latest variable values of w, $wheelABS$, and v are read. Since $v > 0$, the TA moves to the *CalcSliprate* location. On the transition to *Exit*, the $torqueABS$ variable is set to zero, and after the *write* transition, the TA returns to the *idle* location.

Model-checking this particular instance has involved exploring 154182 states out of 203384 stored ones, and the abstract test-case generation took 3.27 seconds on a 1.8 GHz Intel Core i5 processor, with 8 GB of RAM memory.

11.7.5 Python scripts generation

From the abstract test-case of Figure 11.7, the input variable values determining transitions in the TA model are identified automatically, and a Python test-script is generated. When executed by Farkle on the host system, the script sends the signals representing those input values to the target. An excerpt of the generated script is shown in Figure 11.8. Lines 6-10 in the script set the content of the input signal with the initial variable values, whereas line 11 encodes sending the signal to the target. The expected values of variables, as well as the expected order of visited states are defined in lines 13-18, according to the principles described in Section 11.6.2. The log information sent back to the host, from the target, in the form of a signal is then received (line 20). Again, following the aforementioned principles, a set of assertion statements for checking the returned values versus the expected values are also generated as the body of a loop, depicted in lines 23-39 of the script.

When the test-script is executed, an input signal is sent to the target. Upon the receipt of a signal, the process to which the signal is sent starts executing. The different states that a process enters are tracked and logged at runtime and during the execution of the code. This information is then sent back to the script, where it is checked whether the order of the states at runtime and also the value of variables after each state change match the specification in the abstract test-case generated from the TA models. To enable tracking of different states at runtime, the code is implemented in the form of deterministic state-machines, as shown in Figure 11.6. The Python script of Figure 11.8 delivers a "pass" verdict on the implementation of Figure 11.6.

```
import sys
import signals
import xmlrunner
...
# Sending input signal to ABSFL
sig_send_ABSFL = signals.ABSFL_INPUT_SIG()
sig_send_ABSFL.input.ABSFL_w = 8
sig_send_ABSFL.input.ABSFL_v = 12
sig_send_ABSFL.input.ABSFL_wheelABS = 1
sig_send_ABSFL.input.ABSFL_R = 1
self.linx.send(sig_send_ABSFL, self.pid_ABSFL)
# Expected values
ABSFL_w = [8, 8, 8]
ABSFL_wheelABS = [1, 1, 1]
ABSFL_torqueABS = [-1, -1, 0]
ABSFL_v = [12, 12, 12]
ABSFL_R = [0.5, 0.5, 0.5]
ABSFL_state = [1, 2, 3]
# Receive signals from test targets
sig_recv_ABSFL = self.linx.receive
    ([signals.ABSFL_OUTPUT.SIGNO])
# Testing of ABSFL
for i in range(sig_recv_ABSFL.num_states):
print "Transition %d:" %(i+1)
self.assertEqual(sig_recv_ABSFL.states[i].state,
    ABSFL_state[i])
print " state = %d" %sig_recv_ABSFL.states[i].state
self.assertEqual(sig_recv_ABSFL.states[i].w, ABSFL_w[i])
print " w = %d" %sig_recv_ABSFL.states[i].w
self.assertEqual(sig_recv_ABSFL.states[i].wABS,
    ABSFL_wABS[i])
print " wheelABS = %d" %sig_recv_ABSFL.states[i].wheelABS
self.assertEqual(sig_recv_ABSFL.states[i].tABS,
    ABSFL_tABS[i])
print " torqueABS = %d"
    %sig_recv_ABSFL.states[i].torqueABS
self.assertEqual(sig_recv_ABSFL.states[i].v, ABSFL_v[i])
print " v = %d" %sig_recv_ABSFL.states[i].v
self.assertEqual(sig_recv_ABSFL.states[i].R, ABSFL_R[i])
print " R = %d" %sig_recv_ABSFL.states[i].R
...
```

Figure 11.8: Generated Python script

11.7.6 Conformance between the abstract test-case and the Python script

Our abstract test-case presented in Figure 11.7 can be represented operationally by a sequence of states and transitions as follows:
$(idle, w = 0, wheelABS = 0, torqueABS = -1, v = 0, R = 1/2)$

$\xrightarrow{read(v,wheelABS,v)}$
$(Entry, w = 8, wheelABS = 1, torqueABS = -1, v = 12, R = 1/2)$
$\xrightarrow{v>0}$
$(CalcSlipRate, w = 8, wheelABS = 1, torqueABS = -1, v = 12, R = 1/2)$
$\xrightarrow{v<5*(v-w/R),torqueABS=0}$
$(Exit, w = 8, wheelABS = 1, torqueABS = 0, v = 12, R = 1/2)$
$\xrightarrow{write(torqueABS)}$
$(idle, w = 8, wheelABS = 1, torqueABS = -1, v = 12, R = 1/2)$

In a similar manner, the Python scripts give rise to the following:
$(ABSFL\_state = 1, ABSFL\_w = 8, ABSFL\_wheelABS = 1,$
$ABSFL\_torqueABS = -1, ABSFL\_v = 12, ABSFL\_R = 1/2)$
$\xrightarrow{v>0}$
$(ABSFL\_state = 2, ABSFL\_w = 8, ABSFL\_wheelABS = 1,$
$ABSFL\_torqueABS = -1, ABSFL\_v = 12, ABSFL\_R = 1/2)$
$\xrightarrow{v<5*(v-w/R),torqueABS=0}$
$(ABSFL\_state = 3, ABSFL\_w = 8, ABSFL\_wheelABS = 1,$
$ABSFL\_torqueABS = 0, ABSFL\_v = 12, ABSFL\_R = 1/2)$

In the above we have the following: the *Entry* state in the TA model is $ABSFL\_state = 1$ in the Python script. Given this, we observe that the trace generated from executing the Python script is included in the trace generated by executing the TA model. Thus, it follows that we can actually use the Python script as a concrete test-case on the SUT, so the abstract test-case of Figure 11.7 has proven to be a feasible abstract test-case candidate.

11.8 Related Work

Model-based testing by model-checking is a technique introduced almost fifteen years ago [15] as an efficient way of using a model-checker to interpret traces as test-cases. Some approaches to testing with model-checkers are applied on real-time reactive systems. Hessel et al. have proposed test-case generation using the Uppaal model-checker for real-time systems [1] using timed automata specifications. In comparison, in our work we provide an approach suited to an architectural description language, and we offer an end-to-end tool-chain with support for test-

case generation and execution.

Over the last few years, researchers in the software testing community have been investigating how design components and architecture description languages (ADLs) can be used for testing purposes. This led several research groups to develop concrete testing techniques for ADLs[16, 17, 18]. Our framework allows the formal specification of both the interface, and the internal behavior of each EAST-ADL block as UPPAAL PORT TA. In addition, in this work we have provided functional test goals to be considered by the UPPAAL PORT model-checker, defined an executable semantics for the UPPAAL PORT TA, and described a method for generating code that preserves the semantics of the TA model.

While a number of groups have made a distinction between abstract and concrete test cases [19, 20, 21], there are also differences in each case. For instance, Peleska [20] has proposed the RT-Tester tool-suite along with the corresponding methodology, and discussed the two types of test-cases, abstract and executable. The major goal in RT-Tester is to execute test-cases against the models of the system. In our work, we have introduced an approach that generates abstract test cases and then concrete ones, which the latter are actually executable against the running system in its target environment. In [22] based on a subset and preliminary version of the approach (which is extended here in this work) and focusing only on the concrete test cases part, we have discussed how having such executable test scripts to test the system behavior also serves as a method to verify architectural consistency. Finally, it is worth noting that there are different static analysis methods that can be applied to ensure that expected properties in a system hold, thus increasing confidence in its correctness. However, despite the application of such methods, there are still situations/systems where the results of such analysis may be invalidated at runtime due to different factors [23, 24]. In this work, we have tackled this issue by complementing formal verification of the system at the model level with the testing of its behavior at runtime.

11.9 Conclusions and Future Work

In this paper, we have presented a methodology for testing system implementations, starting from EAST-ADL architectural models that are

extended by TA behavioral models. The methodology is supported by a tool-chain consisting of ViTAL and Farkle, which can produce and run test-cases automatically. The abstract test-cases for functional requirements result by model-checking the "TA enriched" EAST-ADL models in UPPAAL PORT. Next, such test-cases are transformed into Python scripts representing the executable test-cases that are finally run on the actual code that is implemented based on the (verified) formal model. Our work is an attempt to exercise the feasibility of test-case generation from EAST-ADL models. The method has shown encouraging results when applied on a Brake-by-Wire prototype from Volvo. As future work, we plan to also investigate abstract test-case generation for timing properties of EAST-ADL models, and integrate the results in our methodology. In addition, we envision extending our work towards other ADLs.

11.10 Acknowledgment

The authors would like to thank Elaine Weyuker for her valuable comments on this work. This research has received funding from the ARTEMIS JU, grant agreement number 269335, and from VINNOVA, the Swedish Governmental Agency for Innovation Systems, within the MBAT project, and also partially from the Swedish Knowledge Foundation (KKS) through the ITS-EASY industrial research school.

Bibliography

[1] Anders Hessel, Kim Larsen, Brian Nielsen, Paul Pettersson, and Arne Skou. Time-Optimal Real-Time Test Case Generation Using UPPAAL. In *Lecture Notes in Computer Science, Formal Approaches to Software Testing*, pages 114–130. Springer Berlin Heidelberg, 2004.

[2] Tretmans Jan. Model based testing with labelled transition systems. In *Formal methods and testing*, pages 1–38. Springer, 2008.

[3] Manoranjan Satpathy, Michael Leuschel, and Michael Butler. ProTest: An automatic test environment for B specifications. *Electronic Notes in Theoretical Computer Science*, 111:113–136, 2005.

[4] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[5] Aditya Agrawal, Gyula Simon, and Gabor Karsai. Semantic translation of Simulink/Stateflow models to hybrid automata using graph transformations. *Electronic Notes in Theoretical Computer Science*, 109:43–56, 2004.

[6] John Håkansson and Paul Pettersson. Partial order reduction for verification of real-time components. In *Formal Modeling and Analysis of Timed Systems*, pages 211–226. Springer, 2007.

[7] Eun-Young Kang, Eduard Paul Enoiu, Raluca Marinescu, Cristina Seceleanu, Pierre-Yves Schobbens, and Paul Pettersson. A methodology for formal analysis and verification of EAST-ADL models. *Reliability Engineering & System Safety*, 120:127–138, 2013.

[8] John Håkansson, Jan Carlson, Aurelien Monot, Paul Pettersson, and Davor Slutej. Component-based design and analysis of embedded systems with uppaal port. In *Automated Technology for Verification and Analysis*, pages 252–257. Springer, 2008.

[9] Daniel Digerås. Integration between Optima and Farkle and verification with a use case about file storage stack integration in a quality of service manager in OSE - Master Thesis. http://liu.diva-portal.org/smash/record.jsf?pid=diva2:624122, April 2011.

[10] The ATESST2 ATESST2 Consortium. EAST-ADL Profile Specification, 2.1 RC3 (Release Candidate). pages 10–75. www.atesst.org, 2010.

[11] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *CONCUR'98 Concurrency Theory*, pages 485–500. Springer, 1998.

[12] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012.

[13] Tobias Amnell, Elena Fersman, Paul Pettersson, Hongyan Sun, and Wang Yi. Code synthesis for timed automata. *Nord. J. Comput.*, 9(4):269–300, 2002.

[14] Wang Yi. CCS+ time= an interleaving model for real time systems. In *Automata, Languages and Programming*, pages 217–228. Springer, 1991.

[15] André Engels, Loe Feijs, and Sjouke Mauw. Test generation for intelligent networks using model checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 384–398. Springer, 1997.

[16] Antonia Bertolino and Paola Inverardi. Architecture-based software testing. In *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints' 96) on SIGSOFT'96 workshops*, pages 62–64. ACM, 1996.

[17] Henry Muccini, P Inverardi, and A Bertolino. Using software architecture for code testing. *Software Engineering, IEEE Transactions on*, 30(3):160–171, 2004.

[18] Hassan Reza and Suhas Lande. Model based testing using software architecture. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 188–193. IEEE, 2010.

[19] C. Nebut, F. Fleurey, Y. Le-Traon, and J.-M. Jezequel. Automatic test generation: a use case driven approach. *Software Engineering, IEEE Transactions on*, 32(3):140–155, 2006.

[20] Jan Peleska. Industrial-Strength Model-Based Testing - State of the Art and Current Challenges. In *Proceedings of the Eighth Workshop on Model-Based Testing*, pages 3–28, March 2013.

[21] Wolfgang Prenninger, Mohammad El-Ramly, and Marc Horstmann. Chapter 15: Case Studies. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[22] Mehrdad Saadatmand, Detlef Scholle, Cheuk Wing Leung, Sebastian Ullström, and Joanna Fredriksson Larsson. Runtime Verification of State Machines and Defect Localization Applying Model-based Testing. In *Proceedings of the WICSA 2014 Companion Volume*. ACM, 2014.

[23] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. Design of adaptive security mechanisms for real-time embedded systems. In *Proceedings of the 4th international conference on Engineering Secure Software and Systems (ESSoS)*, pages 121–134, Berlin, Heidelberg, 2012. Springer-Verlag.

[24] S.E. Chodrow, F. Jahanian, and M. Donner. Run-time monitoring of real-time systems. In *Real-Time Systems Symposium (RTSS), 1991. Proceedings., Twelfth*, pages 74–83, 1991.

Chapter 12

Paper G: Testing of Timing Properties in Real-Time Systems: Verifying Clock Constraints

Mehrdad Saadatmand, Mikael Sjödin
The 20th Asia-Pacific Software Engineering Conference (APSEC), Bangkok, Thailand, December, 2013.

Abstract

Ensuring that timing constraints in a real-time system are satisfied and met is of utmost importance. There are different static analysis methods that are introduced to statically evaluate the correctness of such systems in terms of timing properties, such as schedulability analysis techniques. Regardless of the fact that some of these techniques might be too pessimistic or hard to apply in practice, there are also situations that can still occur at runtime resulting in the violation of timing properties and thus invalidation of the static analyses' results. Therefore, it is important to be able to test the runtime behavior of a real-time system with respect to its timing properties. In this paper, we introduce an approach for testing the timing properties of real-time systems focusing on their internal clock constraints. For this purpose, test cases are generated from timed automata models that describe the timing behavior of real-time tasks. The ultimate goal is to verify that the actual timing behavior of the system at runtime matches the timed automata models. This is achieved by tracking and time-measuring of state transitions at runtime.

12.1 Introduction

In building real-time systems, it is very important to ensure that timing properties are in accordance with the specified timing requirements and constraints. The correctness of these systems is not only dependent on the correctness of the logical results of computations, but also on the time at which the results are produced [1]. The criticality of this issue can vary from one real-time system to another, such as in a real-time media player vs. the airbag system in an automobile. Different methods have been suggested in order to verify the correctness of timing properties in real-time systems. For example, there are diffeent schedulability analysis methods [2] that help to determine whether a set of real-time tasks are schedulable or not. There are several assumptions that are taken into account in performing such analyses, for instance, Worst-Case Execution Times (WCETs) of tasks. Some of the approaches for determining execution times can result in assigning very pessimistic values [3]. Moreover, at runtime, situations may occur that lead to the violation of the assumptions that were taken into account for performing the analyses, and thus invalidation of the analysis results [4, 5]. It should also be noted that for complex systems, such as those in telecommunication domain with huge number of concurrent tasks handling big loads of calls, data connections, billing, routing, and so on, it can be very hard in practice to get such detailed information about each task in the whole system in order to perform schedulability analysis [4, 6].

In this paper, we introduce an approach to test the timing behavior of real-time systems at runtime. The main intention is to basically verify that the timing properties of the tasks constituting the system match the specifications by testing the running system. This is achieved in our approach by consulting the timed automata models [7, 8, 9] of the system and automatically generating test cases from them[1]. Timed automata models representing the timing constraints are used as the source for the generation of test cases and determining pass/fail criteria for them. The test cases when executed against the running system determine whether the observed timing behaviors match what is specified for the tasks in

[1] In this paper, the term *verify* is used as its ordinary meaning in English and not to refer to 'formal verification' in software engineering, unless explicitly stated. Moreover, the term *state machine* is used as a synonym to refer to a timed automaton whenever the main concern is only the states and transitions in the model regardless of the timing specifications.

the timed automata models or not. The focus in this work is mainly on the verification of the clock constraints that are specified in the models.

While most of the methods for verification of timing properties mainly target development phases before the actual execution of a real-time system, such as static analysis and model checking methods [1], our approach provides a way to dynamically test and verify the actual running system. This is especially important to alleviate the issues mentioned above, particularly scenarios which may occur at runtime that can cause invalidation of the results of static analysis methods, and to identify such misbehaviors. On the other hand, the approach can also be very well used to complement static analysis methods to gain more confidence in the correctness of designed systems in terms of their timing properties. We demonstrate the applicability of our approach by using it for testing of timing properties in a Brake-By-Wire (BBW) system from automotive industry. As the platform for implementation of the system, we have used OSE Real-Time Operating System (RTOS) [10], on top of which, BBW application is developed in C/C++.

The remainder of the paper is structured as follows. In Section 12.2, background information about the used technologies and the BBW system is provided. Section 12.3 describes the details of the proposed approach and in Section 12.4, we discuss how we have implemented the approach and applied it on the BBW system. In Section 12.5, related works are discussed and possible scenarios for combination of our approach with those works are also identified. Finally in Section 12.6, a summary of the paper along with highlights and conclusions are provided.

12.2 Background Context

12.2.1 Timed Automata

Timed Automata (TA) are essentially finite state machines which are annotated and extended with real-valued clocks [7, 8, 9]. The clocks, initially set to zero at system start-up, progress and increase synchronously and at the same rate. The value of clocks can also be reset if needed. Timed automata are used to provide an abstract model of real-time systems. Timing constraints are specified using clocks whose values can be checked in the form of guards and invariants. Invariants can be regarded as progress conditions and are used to restrict the way that time

may elapse at a state (location). For example, using invariants, it can be specified that the system is not allowed to stay in a state more than some time units and the transition has to be taken by the specified amount of time. Guards are specified on a transition (edge) as conditions to restrict its temporal occurrence.

Figure 12.1 shows an example timed automata for modeling a real-time lamp introduced in [8]. In this system, there is a timing requirement on how the user presses a button. The action of the user in pressing the button is modeled with the right-hand automaton. The timing requirement in this example states that if the user presses the button, the lamp is switched off or on (on in the low mode). However, if he is fast enough in pressing the button again, the lamp is switched on and ends up in the bright mode. The decision whether the user has been fast in pressing the button or not is determined by using the clock y.

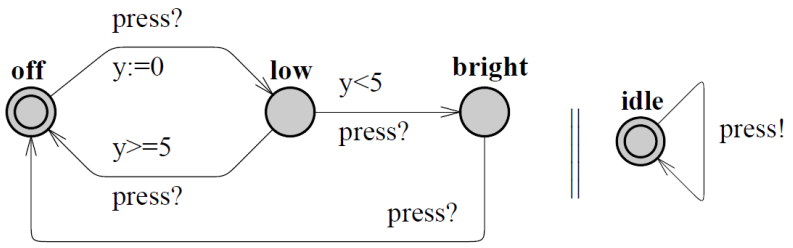

Figure 12.1: TA model of a real-time lamp [8]

12.2.2 OSE RTOS & Farkle

OSE is a commercial and industrial real-time operating system developed by Enea [10] which has been designed from the ground specifically for fault-tolerant and distributed systems. It provides preemptive priority-based scheduling of tasks. OSE offers the concept of direct and asynchronous message passing for communication and synchronization between tasks, and OSE's natural programming model is based on this concept. Linx, which is the Inter-Process Communication protocol (IPC) in OSE, allows tasks to run on different processors or cores, utilizing the same message-based communication model as on a single processor. This

programing model provides the advantage of not needing to use shared memory among tasks. The runnable real-time entity equivalent to a task is called *process* in OSE, and the messages that are passed between processes are referred to as *signals* (thus, the terms process and task in this paper can be considered interchangeably). Processes can be created statically at system start-up, or dynamically at runtime by other processes. Static processes last for the whole life time of the system and cannot be terminated. Types of processes that can be created in OSE are: interrupt process, timer interrupt process, prioritized process, background process, and phantom process. A process can be in one of the following states: ready, running or waiting. One interesting feature of OSE is that the same programming model based on signal passing can be used regardless of the type of process.

Farkle is a test execution framework that is originally developed for testing systems built using OSE. It enables testing embedded systems in their target environments. This capability has become possible by using the signal passing mechanism of OSE which allows Farkle to run on a host machine and communicate with the target by passing signals. In other words, Farkle basically enables testing an embedded system by providing certain inputs to the target in the form of signals and receiving the result as signals containing output values. The test scripts that are used to send and receive signals, and also decide the verdict of test cases are implemented in Python. Figure 12.2 provides an overall idea on how Farkle works.

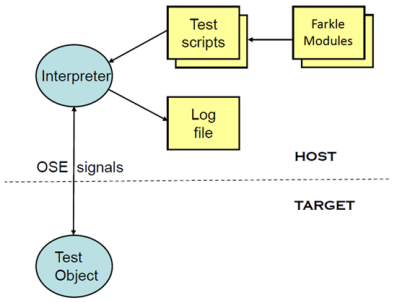

Figure 12.2: Farkle test execution environment.

12.2.3 Brake-by-Wire System

The Brake-by-Wire (BBW) is a braking system in which mechanical parts are replaced by electronic sensors and actuators and thus removing the hydraulic connections between the brake pedal and each wheel brake. Anti-lock Braking System (ABS) is usually an inherent functionality provided by BBW systems [11]. The purpose with the ABS subsystem is to prevent locking of the wheels by controlling braking based on calculated *slip rate* value. Slip rate is calculated according to the following formula (where r is the radius of the wheel):

$$s = (vehicleSpeed - wheelSpeed * r)/vechileSpeed$$

If s is greater than a certrain limit, then the brake actuator is released and no brake is applied, otherwise the requested brake torque is used. The Electronic Control Unit (ECU) for each wheel will have three application software components: one is a sensor to measure the wheel speed, one is an actuator for the brake, and the third one implements the ABS controller for the wheel. A schematic view of the BBW system components is shown in Figure 12.3 considering only one wheel for brevity.

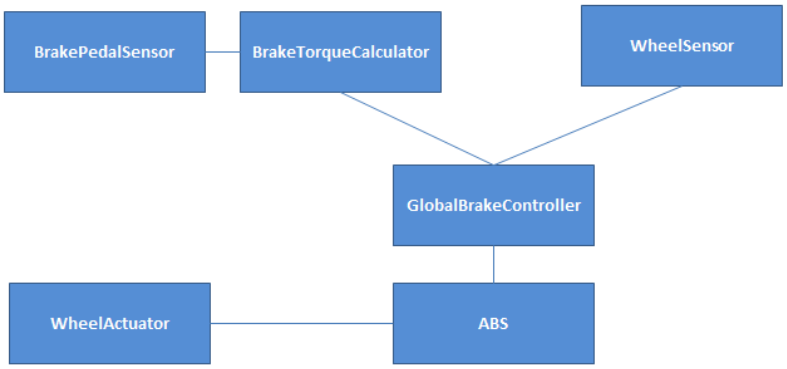

Figure 12.3: Components composing a BBW system

There are several timing requirements in BBW systems. For example, the total brake reaction delay could be specified not to be more than

200ms (in a sample implementation of the BBW system), or requirements on the periods of samplings done by sensors. Generally, BBW is an example of safety-critical, distributed and real-time systems in which meeting timing requirements has also direct impacts on the safety of the system.

A Timed Automata (TA) model, designed in UPPAAL tool [8], describing the internal behavior of the ABS component of BBW system is shown in Figure 12.4. In this model, y is a clock whose specification on the states indicates the amount of time units that can be spent in each state (non-deterministically, between 0 and the specified value) before a transition has to be made to the next state. These timing specifications are naturally derived from high level timing requirements of the BBW system and its components. The values in the TA model here are just samples, and the exact values for each implementation of the BBW system might be different.

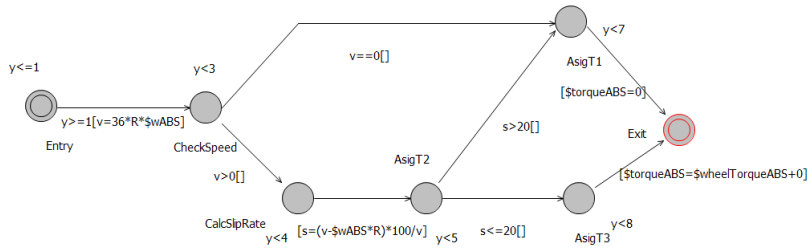

Figure 12.4: Timed automata model of the ABS component

12.3 Proposed Approach

In this section we describe our proposed approach to generate test cases in order to verify that the timing properties of the system at runtime match the clock constraints specified in the timed automata models of the system. This is made possible basically by annotating the code so that state changes can be determined and tracked at runtime, and measuring the time difference between the state changes. The following are the steps that constitute the approach:

1. Based on the automata models, C/C++ enumerations (enum) that represent each state machine and their internal states are generated. These enumeration structures are stored in a C/C++ file along with a helper function called `set_state_wtime(StateMachine, State)`. The file is then included in the implementation code of the target application (i.e., to be tested).

2. The states in the timed automata model are mapped to the code using the above helper function. This is done by adding calls to the `set_state_wtime()` helper function at places where a state change occurs in the code. The helper function basically logs the new state belonging to the specified state machine along with the time stamp at which the transition and change to the new state has occurred.

3. According to the timing specifications in the timed automata model, a test script is generated which verifies the measured time difference between (pairs of) consecutive states against the model. In other words, if the time difference and also the order of state changes match the model, then the result of the test is determined as *pass*, otherwise a *fail* verdict is decided.

The generation of executable test scripts described in Step 3 is done in the following way:

- Minimum number of paths covering all clock constraints in the timed automata model are identified (clock constraint coverage). For example, in case of the TA model of the ABS component shown in Figure 12.4, the following paths are selected: $Entry \rightarrow CheckSpeed \rightarrow AsigT1 \rightarrow Exit$ and $Entry \rightarrow CalcSlipRate \rightarrow AsigT2 \rightarrow AsigT3 \rightarrow Exit$.

- For each of the identified paths a test script is generated. The script basically provides and sends necessary input(s) causing the state machine to start from the *Entry* state, taking the states and transitions constituting the selected path until reaching the *Exit* state. Considering the example in Figure 12.4, this means providing a value for $wABS$ variable resulting in a value for V causing the desired transition and path to be taken. Based on the log information generated by the helper function, the script checks whether the order of the states and the time spent in each state with a clock

constraint match the extracted path from the timed automata or not.

12.4 Application & Implementation of the Approach

We have implemented the BBW system on OSE 5.5.1. An OSE process is created and developed for each component shown in Figure 12.3, and the communication between them is implemented by defining and passing appropriate OSE signals. For example, the signal definition for passing wheel speed information between processes is shown in Listing 12.1.

Listing 12.1: Signal definition for wheel speed

```
#define WHEEL_SPEED_SIG 1026
typedef struct WheelSpeedSignal{
        SIGSELECT sigNo;
        float WheelSpeed;
} WheelSpeedSignal;
```

To test and verify the clock constraints in the running system, the approach described in the previous section is followed. First information about different states is extracted from the timed automata models and enumeration structures (C/C++ enums) representing them are automatically generated:

```
enum StateMachines {BrakePedalSensor, BrakeTorqueCalculator,
      GlobalBrakeController, WheelActuator, ABS, WheelSensor};
enum States {Entry, CheckSpeed, CalcSlipRate, AsigT1, AsigT2,
      AsigT3, Exit, Brake, NoBrake, Reac, ...};
```

The result of this step is a C/C++ file containing the generated data structures along with the aforementioned helper function (that has a fixed implementation), which is then included in the implementation code of the BBW-ABS system. The next step is to map the states in the timed automata models to the code by adding calls to the helper function. The result is depicted in Figure 12.5 in case of the ABS process.

This is the only manual step in the whole approach. This step may also be automated if in a model-driven development methodology, for

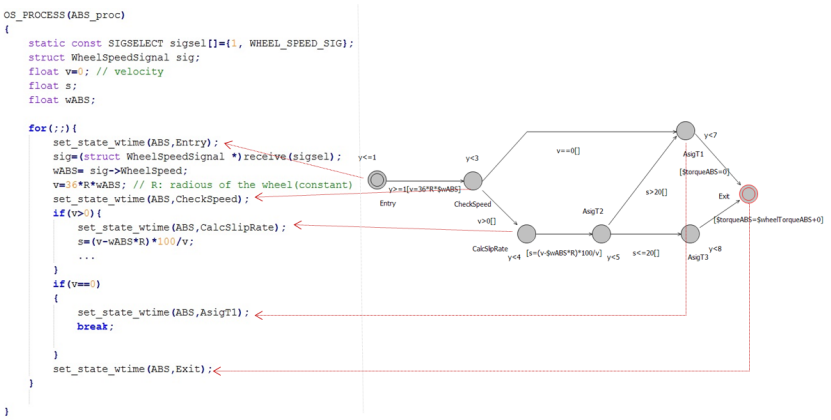

Figure 12.5: Mapping of states to the code (ABS function)

example, the code is generated taking into account the timed automata models of the system and thus aware of different states and transitions.

The final step is the generation of executable test scripts. For each identified path for covering the clocks constraints in the model, a test script is generated which, by investigating the generated log information about state changes, verifies that:

1. the order of visited states is in accordance with the timed automata model,

2. the amount of time spent in each state matches the timing specifications in the timed automata model. This is done by consulting the time stamps associated with and logged for each state change.

These details (order of states and timing specifications) are inherent and present in the timed automata models and are extracted from them in generating test scripts.

The test scripts are generated in the form of Python scripts which are then executed by the Farkle test execution framework (described in Section 12.2.2). Farkle which runs on the host machine enables test scripts to communicate with the target system. The scripts send signals to the BBW processes running on the target. These signals (e.g., wheel speed signal in Listing 12.1) contain input values which are received,

extracted and acted upon by the recipient process. In terms of state machines in the models, passing these input values invoke transitions in the recipient process's internal states. Whenever during the execution of a process the `set_state_wtime(StateMachine,State)` helper function (added during the mapping step) is called, a log record is created for tracking the states and the time point that a state change has occurred. This is used by test scripts to determine test verdicts. Finally, the *pass* or *fail* results returned by executing the test scripts show whether the actual behavior of the system at runtime has been in accordance with and respecting the constraints specified in the timed automata models or not.

12.5 Related Work

In [12], we have previously introduced an approach for monitoring of timing properties of real-time tasks at runtime by enriching schedulers with the necessary monitoring mechanisms. The approach does not only enable provision of information about timing properties such as actual execution time (vs. estimated WCET), response time, deadline misses, and observed period and Minimum Inter-Arrival Time (MIAT) of tasks, but also helps to *preserve* and *enforce* such properties at runtime. In other words, if, for example, a task is taking too much time than its allowed time budget, this time budget is enforced by preempting the task, to let other tasks in the system perform as expected and predetermined. Although in [12], we have not explicitly discussed and dealt with testing of such timing properties, the approach and the monitoring mechanisms introduced there can be regarded as the core functionality needed for dynamic testing of properties such as execution time of tasks, determining occurrence of deadline misses in the system, violation of period and MIAT values, and so on. In this paper, however, we more directly addressed testing of real-time systems. Moreover, we focused mainly on timing properties specifiable in the form of timed automata describing the internal behavior of real-time tasks, as well as the system in general (e.g., end-to-end deadlines). Extending the work done in [12] for testing purposes, and combining it with the approach suggested here to provide a comprehensive testing framework for timing properties is left as future directions of this work to investigate. Beside the monitoring method we have introduced in [12], we have also provided a survey of

different available methods and tools for monitoring of timing constraints in [13].

Uppaal is a tool suite for modeling and verification of real-time systems modeled as networks of timed automata [8, 14]. Three main parts that constitute Uppaal are a description language, a simulator and a model-checker. Uppaal is an example of prominent tools and methods for verification of real-time systems at the modeling level and does not concern implementation code and actual execution of the system. Uppaal TRON [15, 16] is a testing tool based on the Uppaal engine that is designed for black-box conformance testing of real-time systems. The testing approach provided by TRON is similar to ours in the sense that both communicate with and execute test cases against the running system, and also both use timed automata models. Uppaal TRON derives, executes, and evaluates test cases against the implementation of the system in real-time. However, the main difference between the testing method of Uppaal TRON and our approach is that Uppaal TRON addresses online generation and execution of test cases based on the results of previous executed test cases, and focuses mainly on observable input and output actions considering the Implementation Under Test (IUT) as black-box and assuming that its internal states are not observable [16]. While in our approach, test cases are only generated offline, and also the IUT is considered as white-box whose internal state changes are fully tracked and compared against the TA models. The mapping of states to code in our approach is thus needed and introduced to enable this feature. Also what is tested in our approach is not the output from the IUT and the time at which it is produced, but the order of states and the time spent in each one which has a time constraint. From this perspective, there seems to be potentials for combining our testing approach with Uppaal TRON. Shin, Drusinsky and Cook present in [17] an approach for white-box testing of timing properties. They introduce assertion state charts as a way to specify and keep track of timing constraints. From these state charts, some test cases are derived manually (e.g., testing single use case scenarios) and some are automatically generated (e.g., testing the state chart under test itself). Their approach is however closer to Uppaal TRON with respect to what is actually testable and tested in the running system.

In [18], we have discussed the general idea of combining static analysis and testing. Its potentials and different combination scenarios along with an example are also provided in that work.

12.6 Conclusion

In this paper, we introduced an approach for dynamic testing of timing properties in real-time systems. By tracking state changes at runtime, the approach allows for more detailed testing of timing properties of real-time systems and their tasks whose internal behaviors are represented in the form of timed automata. Testing and runtime verification of timing properties becomes especially important in cases where static analysis methods are hard to apply in practice or can be invalidated at runtime. In general, however, both approaches (static analysis methods and testing) can be considered complementary and used together to gain more confidence in temporal correctness of real-time systems.

Timed automata models are used in our approach as a representation and source for timing requirements and constraints from which test cases are automatically generated. We demonstrated the applicability of our approach on the ABS subsystem of Brake-By-Wire system, verifying the timing constraints specified on different states constituting its behavioral model. Also, as for the implementation of our approach, we used OSE as the core platform and real-time operating system, C/C++ for implementing the BBW application, and Python as the language for generating executable test scripts. Using OSE along with its test execution environment, Farkle, enables to test an embedded system in its target environment. This is a valuable feature considering that resource constraints originating from the execution environment (e.g., battery/power, layout and size, heat generation, available memory, CPU, etc.) affect and dictate, to a great degree, how an embedded system should be designed and perform. It should, however, be noted that the approach is not necessarily dependent on these technological choices and may be well implemented differently.

As mentioned in the paper, the mapping step of the approach, in which state changes are annotated and marked in the code using the helper function, is the only step which needs manual intervention. As a future work, we are working on solutions for automating this step, and thus automating the whole testing approach. Also, in this work, since we were only interested in the time difference between state changes, the execution time of the added calls to the helper function (to log and timestamp state changes) had no effect on the test result and was simply ignored. However, if, for example, end-to-end response times are to be tested, it is important to take into account the added timing-cost of

the helper function. Considering that the helper function has a fixed execution time, though, its impacts on the execution time of a task to which it is added may be easily predicted and reduced from the task's total execution time. Also, it might be possible to claim that in some systems, if tests are passed while having helper function calls in the code, they might also work fine in terms of timing properties when the helper function calls are removed, e.g., in the release and final version of a product. Careful investigation of such claims and scenarios, and side effects of adding helper functions to enable testing, as well as different ways to mitigate them are left as other future directions of this work.

12.7 Acknowledgements

This work has been supported by the MBAT European Project [19] and Xdin Stockholm AB (formerly Enea Services Stockholm AB) [20] through the ITS-EASY industrial research school [21]. The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement no 269335 (see Article II.9. of the JU Grant Agreement) and from the Swedish Governmental Agency for Innovation Systems (VINNOVA). We would also like to thank Raluca Marinescu and Cristina Seceleanu for their technical tips and support for this work.

Bibliography

[1] Joachim Wegener and Matthias Grochtmann. Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing. *Real-Time Syst.*, 15(3):275–298, November 1998.

[2] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011.

[3] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem-overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.

[4] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. Design of adaptive security mechanisms for real-time embedded systems. In *Proceedings of the 4th international conference on Engineering Secure Software and Systems*, ESSoS'12, pages 121–134, Eindhoven, The Netherlands, 2012. Springer-Verlag.

[5] S.E. Chodrow, F. Jahanian, and M. Donner. Run-time monitoring of real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 74 –83, dec 1991.

[6] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems. In *The Sixth International Conference on Software Engineering Advances (ICSEA)*, October 2011.

[7] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994.

[8] Gerd Behrmann, Re David, and Kim G. Larsen. A tutorial on Uppaal 4.0. `http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf`, November 2006.

[9] Johan Bengtsson and Wang Yi. Timed Automata: Semantics, Algorithms and Tools. In W. Reisig and G. Rozenberg, editors, *In Lecture Notes on Concurrency and Petri Nets*, Lecture Notes in Computer Science vol 3098. Springer–Verlag, 2004.

[10] Enea. `http://www.enea.com`, Last Accessed: June 2013.

[11] S. Anwar. An anti-lock braking control system for a hybrid electromagnetic/electrohydraulic brake-by-wire system. In *American Control Conference, 2004. Proceedings of the 2004*, volume 3, pages 2699–2704 vol.3, 2004.

[12] M. Saadatmand, M. Sjodin, and N.U. Mustafa. Monitoring capabilities of schedulers in model-driven development of real-time systems. In *17th IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–10, Krakow, Poland, 2012.

[13] Nima Asadi, Mehrdad Saadatmand, and Mikael Sjödin. Run-Time Monitoring of Timing Constraints: A Survey of Methods and Tools. In *The Eighth International Conference on Software Engineering Advances (ICSEA)*, Venice, Italy, October 2013.

[14] Uppaal. `http://www.uppaal.org/`, Accessed: August 2013.

[15] Uppaal for Testing Real-Time Systems Online (TRON). `http://people.cs.aau.dk/~marius/tron/`, Accessed: August 2013.

[16] Kim G. Larsen, Marius Mikucionis, Brian Nielsen, and Arne Skou. Testing real-time embedded software using UPPAAL-TRON: an industrial case study. In *Proceedings of the 5th ACM international conference on Embedded software*, EMSOFT '05, pages 299–306, New York, NY, USA, 2005. ACM.

[17] Man-Tak Shing, D. Drusinsky, and T.S. Cook. Quality assurance of the timing properties of real-time, reactive system-of-systems. In

2006 IEEE/SMC International Conference on System of Systems Engineering, pages 6 pp.–, 2006.

[18] Mehrdad Saadatmand and Mikael Sjödin. On Combining Model-Based Analysis and Testing. In *10th International Conference on Information Technology : New Generations (ITNG 2013)*, Las Vegas, NV, USA, April 2013.

[19] MBAT Project: Combined Model-based Analysis and Testing of Embedded Systems. `http://www.mbat-artemis.eu/home/`, Accessed: June 2013.

[20] XDIN AB. `http://xdin.com/en/about-xdin/enea-experts/`, Accessed: June 2013.

[21] ITS-EASY post graduate industrial research school for embedded software and systems. `http://www.mrtc.mdh.se/projects/itseasy/`, Accessed: June 2013.