Mälardalen University Licentiate Thesis
No.150

Satisfying Non-Functional Requirements in Model-Driven Development of Real-Time Embedded Systems

Mehrdad Saadatmand

May 2012

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

ISSN 1651-9256
ISBN 978-91-7485-066-6
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

Abstract

Design of real-time embedded systems is a complex and challenging task. Part of this complexity originates from their limited resources which incurs handling a big range of Non-Functional Requirements (NFRs). Therefore, satisfaction of NFRs plays an important role in the correctness of the design of these systems. 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 code generation. In this thesis, we identify some of the challenges that exist in model-driven development of real-time embedded systems with respect to NFRs, and provide techniques and solutions that aim to help with the satisfaction of NFRs. Our end goal is to ensure that the set of NFRs defined for a system is not violated at runtime.

First, we identify and highlight the challenges of modeling NFRs in telecommunication systems and discuss the application of a UML-based approach for modeling them. Since NFRs have dependencies, and the design decisions to satisfy them cannot be considered in isolation, we propose a model-based approach for trade-off analysis of NFRs. The approach enables the comparison of different design models with respect to the satisfaction level of their NFRs. Following the issue of evaluating the interdependencies of NFRs, we also propose solutions for establishing and maintaining balance between different NFRs. In this regard, we categorize our suggested solutions into static and dynamic methods. The former refers to a static design and set of features which ensures and guarantees (by construction) the balance of NFRs, while the latter means establishing balance at runtime by reconfiguring the system and runtime adaptation. Finally, we discuss the role of the execution platform in preservation and monitoring of timing properties in real-time embedded systems and propose an approach to enrich the platform with necessary mechanisms for monitoring them.

To my parents...

Acknowledgments

I would like to start by thanking and appreciating the work of all the people at IDT department for making it a very friendly and cooperative research and educational environment which I believe can serve as a role model for other institutes.

Many special thanks to my main supervisor, Mikael Sjödin, who has been very supportive and encouraging beyond just academical work and also to my assistant supervisors Antonio Cicchetti and Radu Dobrin for all their guidance. Working with you all is a great pleasure.

Thanks to Per Wolde who has been a great friend of mine since the very first days that I came to Sweden and his support for me to start this work. To my office mates, Federico and again Antonio, with whom I have had great memories especially from all the travels that we had together; thank you.

I would like to also thank my managers and colleagues at Enea & Xdin: Barbro Claesson, Erik Netz, Anders Törnqvist, Thomas Barnå, Celi, Mathias L., Daniel B., Joel H., Detlef; also all the great people at IDT whose acquaintance I treasure: Barbara, Farhang, Moris, Saad, Rafia, Åsa, Gunnar, Malin, Monika, Carola, Ingrid, Susanne, Andreas J., Thomas N., Ivica, Dag, Gordana, Jan C., Jan G., Hans, Frank L., Paul, Cristina S., Kristina L., Mats, Björn, Lars, Adnan, Aida, Aneta, Séverine, Nima, Jagadish, Yue Lue, Thomas L., Etienne, Mikael Å., Juraj, Josip, Ana, Luka, Leo, Hüseyin , Hongyu, Kivanc, Andreas G., Daniel, Sasi, Sara, Stefan, Abhilash...

A very special thanks to Nazanin for being by my side with her positive spirit; my life in Sweden would have not been the same without you.

My deepest gratitudes to my family who have always been there for me no matter what. Without them I could have never reached this far.

Mehrdad Saadatmand
Västerås, May, 2012

List of Publications

Papers Included in the Licentiate Thesis [1] [2]

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

Paper B *Modeling and Trade-off Analysis of NFRs*. Mehrdad Saadatmand, Antonio Cicchetti and Mikael Sjödin. MRTC report ISSN 1404-3041 ISRN MDH-MRTC-267/2012-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, April, 2012. [3]

Paper C *Modeling Security Aspects in Distributed Real-Time Component-Based Embedded Systems*. Mehrdad Saadatmand and Thomas Leveque. The Ninth International Conference on Information Technology : New Generations (ITNG), Las Vegas, Nevada, USA, April, 2012.

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

Paper E *The Role of Schedulers in Model-Driven Development of Real-Time Systems*. Mehrdad Saadatmand, Mikael Sjödin and Naveed Ul

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

[2] The included articles have been reformatted to comply with the licentiate layout.

[3] Under submission for conference publication. Also partially published as a WiP paper in the Sixteenth IEEE International Conference on Emerging Technology & Factory Automation (ETFA11), Toulouse, France, September, 2011.

Mustafa. MRTC report ISSN 1404-3041 ISRN MDH-MRTC-264/2012-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, March, 2012.

Additional Papers, Not Included in the Licentiate Thesis

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

- *Enabling Trade-off Analysis of NFRs on Models of Embedded Systems*. Mehrdad Saadatmand, Antonio Cicchetti and Mikael Sjödin. The Sixteenth IEEE International Conference on Emerging Technology & Factory Automation (ETFA11), WiP session, Toulouse, France, September, 2011.

- *A Methodology for Designing Energy-aware Secure Embedded Systems*. Mehrdad Saadatmand, Antonio Cicchetti and Mikael Sjödin. The Sixth IEEE International Symposium on Industrial Embedded Systems (SIES11), Västerås, Sweden, June, 2011.

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

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

Contents

I

Thesis

Chapter 1

Introduction

The main goal in the design of a software system is to deliver a product which satisfies all the requirements of different stakeholders. Requirements are generally categorized into functional and non-functional. Although non-functional requirements usually receive less attention in the design of many types of systems (for example in desktop applications) they play an important role in certain domains; particularly, real-time embedded systems. Satisfaction of non-functional requirements, such as timing, is critical in these systems and can determine the success or failure of the final product. In this thesis, we look at different challenges of satisfying non-functional requirements within the context of model-driven development of real-time embedded systems. In our work which extends from the system model down to the execution platform, we propose solutions and methods that help to better satisfy non-functional requirements.

1.1 Basic Terms and Definitions

In this section, we provide definitions for some of the key terms that are used throughout the thesis.

We consider two categories of requirements: *functional* and *non-functional*. In the simplest form, functional requirements are those which define *what* the system should do, while the term non-functional requirement is used for requirements which specify *how* a system should perform, or as suggested in [1] "a non-functional requirement is an attribute of or a con-

straint on a system". There is a big number of suggested definitions for non-functional requirements which are discussed in [2]. These requirements are usually described with terms that end with 'ility' such as availability, 'ity' such as atomicity while a few other ones such as performance and user-friendliness do not follow this pattern. According to the IEEE standards, 610.12-1990 and ISO/IEC/IEEE 24765:2010(E) [3, 4], the following definitions are provided for requirement, functional and non-functional requirement, and also *derived requirement*:

- 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.
- 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 thesis, we use the term extra-functional as a synonym to non-functional, however, to be consistent with the style that is used in the literature, we use it as an adjective for properties as in the phrase "extra-functional properties". This brings us to the next term which is *property*. It is defined in IEEE standard ISO/IEC/IEEE 24765:2010(E) [3] as:

> A responsibility that is an inherent or distinctive characteristic or trait that manifests some aspect of an object's knowledge or behavior (responsibility: A generalization of properties (attributes, participant properties, and operations) and constraints. An instance possesses knowledge, exhibits behavior, and obeys rules. These are collectively referred to as the instances responsibilities. A class abstracts the responsibilities in common to its instances. A responsibility may apply to each instance of the class (instance-level) or to the class as a whole (class-level) [5]).

In this work, we distinguish between requirements and properties by considering the former as an expression of a *need* (possibly informal), and the latter as a statement that is usually asserted formally and can be therefore proven and analyzed (e.g., calculated response time of a component). This implies that "a requirement can require that a certain property holds (e.g., absence of deadlock, meeting deadline, not overflowing a queue, etc.) and that in order for a property to hold a number of requirements may have to be met, which we normally neither express nor assert formally"[1].

Based on these definitions, we use the term *satisfaction* for requirements and *preservation* for extra-functional properties (i.e., to keep properties within their acceptable and valid ranges and protect them from violation). Accordingly, "response time of a component should not exceed 2ms" is a requirement, while "response time of component A never exceeds 2ms" and "response time of component B is equal to 1ms" are expressions of properties in a system.

Balancing trade-offs among requirements can incur adjusting system properties that are related to those requirements. Moreover, we consider a relationship between an NFR and extra-functional properties of the system in the sense that, in order to satisfy an NFR, its related extra-functional properties

[1] These definitions have been provided and formulated with the help of Prof. Tullio Vardanega.

should have valid values. For example, to satisfy performance requirements of a real-time system, execution and response time values (among others) should remain within a valid range. Moreover, a property per se does not tell much about its validity. Only when a property is considered along with its related NFR or NFRs, it becomes possible to evaluate whether it is valid for a specific system and design or not. For example, a component with the worst-case execution time of 100ms can be acceptable in designing one system but the same component can violate the requirements of another system and may not be appropriate for it.

1.2 Background and Motivation

Embedded computer systems are systems that are designed to operate as part of other devices. These systems are usually designed for specific and dedicated functions, and interact with their external environment through sensors and actuators [6, 7]. This interaction often brings along real-time requirements. However, besides real-time requirements, resource constraints in these systems also introduce other limitations with respect to properties such as energy consumption, performance, and safety. Due to resource constraints that these systems have, the correct functionality of the whole system depends heavily on the satisfaction of its NFRs. On the other hand, NFRs are often interconnected and cannot be considered in isolation. An example of such interconnection and dependency can be observed more explicitly between security and performance requirements in a system. Therefore, in satisfying a requirement, its impacts on other requirements should also be taken into account, and trade-off analysis among NFRs needs to be done [2, 8]. Handling a big range of requirements, establishing balance among them and performing trade-off analysis contribute to the high design complexity of real-time embedded systems and make it a challenging task [7].

Model-Driven Development (MDD) is a promising approach in raising abstraction level and reducing design complexity of real-time embedded systems, which also enables analysis of the system at earlier phases of development. This helps with the identification of problems in the system design before reaching the implementation phase [9, 10]. The implementation of the system can also be generated from the design models through a set of model transformations. In this context, non-functional requirements are captured by expression of extra-functional properties that are attached to model elements. In an integrated model-driven and component-based approach, the model elements

that extra-functional properties are annotated on can be components.

However, in order to ensure correctness of the generated system, it is important to ensure that in each step, extra-functional properties that are specified on the model are preserved. This means that for each transformation, input properties should be preserved in the output of the transformation. It also includes the system execution at runtime which is the result of executing the generated source code. This implies that the execution platform should be able to actively monitor and enforce 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 probable deviations.

1.3 Problems and Contributions Overview

There are several challenges in satisfying non-functional requirements using a model-driven development method for real-time embedded systems. These challenges include (but are not limited to) issues such as how to model NFRs, how to model their interdependencies, and also how to ensure that the implementation that is generated from the model behaves as expected during execution and not in a way that impairs the satisfactions of NFRs at runtime.

As the first step, we introduce an approach using Unified Modeling Language (UML) for modeling non-functional requirements and extra-functional properties in telecommunication systems (as an example and sub-domain of real-time embedded systems). It is done by suggesting adoption from and combining several already existing UML-based languages. In satisfying an NFR, its impacts on other NFRs should also be taken into account. To enable model-based trade-off analysis of NFRs, we propose a UML profile for modeling dependencies and impacts of NFRs and functional parts that contribute to the satisfaction of each NFR in a positive or negative way. The concepts that are provided in this UML profile are based on the Soft-goal Interdependency Graph (SIG) approach that is used in the NFR Framework [11].

As an example of interdependency of NFRs, we consider security and timing requirements in real-time embedded systems. To satisfy security requirements certain mechanisms, such as encryption, should be applied. However, adding such security mechanisms may result in the violation of the timing requirements of the system. In the context of an integrated model-driven and component-based approach, we propose an approach to automatically derive the component model of a system that includes components implementing se-

curity mechanisms (from an original component model without security components). This is done by identifying and annotating sensitive data flows in the model. The derived component model uses the same meta-model as the original model, and therefore, enables using the same timing analysis applicable to the original model. This way, timing impacts of added security components can be analyzed at earlier phases and before the implementation phase.

Modeling NFRs and including them along with functional elements, and analyzing the model do not guarantee that the system that is generated from the model is capable of satisfying those NFRs. Violations in this respect can not only happen during transformations to generate intermediate models and finally code, but also during execution of the generated code on the platform. To mitigate such violations, extra-functional properties of the system can be enforced and monitored during runtime. Upon detection of such violations, adaptation and reconfiguration of the system may be performed as a counteractive measure. We have developed an adaptive approach for balancing the security level of a system with its timing requirements at runtime. The approach is based on keeping a history of the timing behaviors of encryption algorithms and using a weaker but less time-consuming one when a timing violation occurs. The introduced adaptation mechanism is suitable for complex real-time systems where timing analysis is not practical or not much information about timing characteristics of each individual task is available.

On the other hand, in order to monitor and detect violation of extra-functional properties at runtime, the execution platform should have certain capabilities. We discuss these capabilities for timing requirements which are of great importance in real-time embedded systems. We also introduce the concept of second-layer scheduler for monitoring of real-time events, such as deadline misses and execution time overruns, on top of a real-time operating system with a priority-based preemptive scheduling policy to provide such capabilities.

1.4 Thesis Outline

The thesis is organized into two parts:
Part I includes three chapters. Chapter 1 provided an introduction to the thesis and formulated the research problem. In Chapter 2, the research overview describing detailed research goals and contributions is offered. In Chapter 3 we summarize the work and present suggestions for the future work.
Part II presents the technical contributions of the thesis in detail in the form of

research papers which are organized in Chapters 4-8.

Chapter 2

Research Overview

In this thesis, we target some of the challenges related to the satisfaction of non-functional requirements in the context of model-driven development of real-time embedded systems. Towards this goal, we also discuss techniques that help with the preservation of extra-functional properties in these systems and mitigate their deviations from the desired behavior. The importance of this research objective becomes more obvious and is motivated considering the following points:

- Satisfaction of non-functional requirements is important in the design of real-time embedded systems. A system design which cannot satisfy its non-functional requirements (e.g., timing requirements) can mean failure of the end product.

- The non-functional requirements are mapped to architectural models and captured by extra-functional properties.

- Analysis is performed on models of embedded systems in order to calculate extra-functional properties of the system and its components, such as response time, and to evaluate satisfaction feasibility of its non-functional requirements. Analyses are based on some assumptions and preconditions. Violation of these assumptions equals to the violation and invalidation of the 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 violation of the assumptions that were used to perform analysis [12].

- It may not be practical and/or economical to perform analysis on all types of extra-functional properties. In such cases, runtime monitoring and handling of violations can be used to *preserve* extra-functional properties.

In this context, the term *preserve* that we use in our work implies and is based on the assumption 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 design.

Satisfaction of non-functional requirements in real-time embedded systems has been considered in this thesis in the context of model-driven development and particularly Model-Driven Architecture (MDA) methodology that is suggested by Object Management Group (OMG) [13]. UML is the key modeling language at the heart of MDA. Figure 2.1 depicts the MDA design flow that defines the development context of this thesis.

As shown in figure 2.1, different types of analyses are performed at different levels 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.

2.1 Research Goals

Aligned with the context that was described in the previous section the following research goals have been defined:

G1: Evaluation of UML approaches for modeling NFRs in telecommunication systems: In order to include NFRs in the design models of a system, the modeling language that is used to design the system should provide concepts for modeling of NFRs. Considering the characteristics of telecommunication systems and the problems that were observed in developing such systems in terms of their NFRs, we have proposed a UML-based approach using a combination of several existing UML profiles [14] for modeling of NFRs

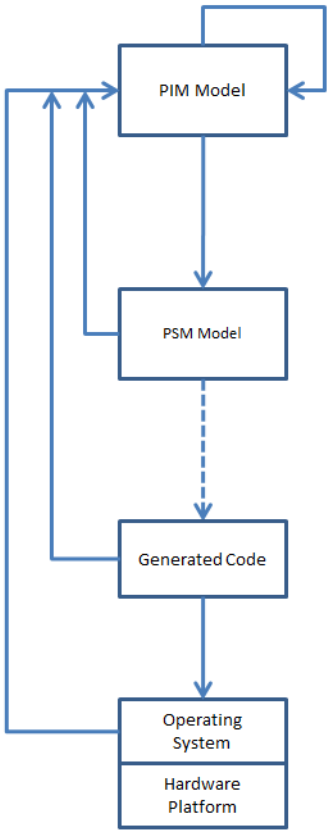

Figure 2.1: Model-driven development context

in these systems. We did this by looking at the EAST-ADL [15] modeling language which is defined for automotive domain using a similar approach and by adopting from a set of different UML profiles. We have also touched upon benefits and drawbacks of using UML profiles for defining new modeling concepts compared to defining a Domain-Specific Language (DSL) [16] from scratch.

G2: Providing an approach for model-based trade-off analysis of NFRs: To model interdependency of NFRs and identify their impacts on each other,

we have proposed a UML profile. The profile offers necessary concepts to model an NFR along with it 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 of 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 negative or positive impact that other NFRs in the system may have on it.

G3: Identification and development of methods for balancing extra-functional properties of real-time embedded systems: Timing properties are of utmost importance in real-time embedded systems. Also as mentioned earlier, interdependencies of different extra-functional properties cannot be neglected in the design of these systems. Therefore, for this research goal, we have specifically looked at the interdependency of timing and security properties as an example of such relationships and how we can establish balance among them. Basically, we categorize the approaches for the aforementioned problem into *static* and *dynamic* (adaptive), which are explained below and may also be applicable for other extra-functional properties as well.

- One approach to establish balance between timing and security properties in a system is to identify parts of the system (i.e., sensitive data) in the model that need to be protected, add security features to protect them, and finally perform timing analysis on the derived model to ensure that the added security features do not violate the timing requirements. This method leads to a *static* design in the sense that a specific security mechanism, which is analyzed, and thus, known to execute within its allowed time budget is always used in each execution. We have proposed and implemented a method to automatically derive the component model of the system including components that implement its security requirements. The original component model of the system is used as input for a transformation that considers the sensitive data flows in the original system model and adds appropriate security components. The key ideas here are to facilitate implementation of security features for non-security experts, bring security considerations into earlier phases of development, and thus most importantly enable timing analysis of the

system including security components at the model level. This helps designers to identify problems and imbalance between timing and security at the model level and fix it before reaching the implementation phase.

- The static method may not be practical for systems with high complexity which are hardly analyzable or systems with unknown timing behaviors of their components. Instead, for such systems, a *dynamic* way 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, we have suggested an approach, along with its implementation for selecting appropriate encryption algorithms at runtime (in terms of their timing behaviors) in an adaptive way. In this approach, the timing behavior of each execution of the encryption procedures is logged, and used as feedback for selecting a more suitable encryption algorithm in the next execution.

G4: Identification and development of methods for runtime monitoring of extra-functional properties, focusing on timing properties: The above mentioned methods are not enough per se for runtime preservation and enforcement of extra-functional properties and require some support mechanisms from the underlying platform, which brings us to the next problem that exists in achieving this goal. Many of the real-time operating systems that are used as execution platform provide support for a priority-based scheduling paradigm. However, in such platforms, there is often no explicit specification on how to define and introduce different types of real-time tasks (i.e., periodic, sporadic, aperiodic) in the system. Therefore, for example, periodic behavior is actually implemented by using timer interrupts and this way a periodic task is created. This means that the platform has no concept of periodic task and there is no observable runnable entity in the system as a periodic task. Since the platform has no awareness about the type of task it is scheduling and its timing parameters such as deadline and worst-case execution time, monitoring and detection of real-time events such as deadline misses and execution time overruns are not supported and need to be implemented by end users in arbitrary ways. Monitoring of timing behavior of the system at runtime is necessary in order to ensure preservation of timing requirements. We have proposed the idea of the second-layer scheduler which enables users to specify real-time tasks in detail using their timing parameters (period, deadline, etc.). By having awareness about the type of each real-time task that is defined in the system, the second-layer scheduler is capable of providing detailed monitoring information. This monitoring information, among others, include deadline misses, execution time overruns,

task preemptions, and completion times.

2.2 Research Contributions

The following published research results cover the research goals described in the previous section, as summarized in Table 2.1.

Research Goals	Papers
G1	A
G2	B
G3	C, D
G4	E, D

Table 2.1: Mapping of published papers to research goals

- **Paper A: 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 2011), Barcelona, Spain, October, 2011

  Abstract: In this paper, we propose a UML-based solution, consisting of different modeling languages, to model non-functional requirements in telecommunication domain, and discuss different challenges and issues in the design of telecommunication systems that are related to these requirements.

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

- **Paper B: Modeling and Trade-off Analysis of NFRs**; Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin; MRTC report ISSN 1404-3041 ISRN MDH-MRTC-267/2012-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, April, 2012

  Abstract: In this paper, we focus on establishing balance and enabling trade-off analysis of Non-Functional Requirements(NFR) and identify what information about NFRs is required in order to perform trade-off analysis. We propose and explain an approach to incorporate this information into system models in order to enable trade-off analysis. Our approach is based on UML profiling method to annotate model elements with necessary information.

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

- **Paper C: 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, Nevada, USA, April, 2012

  Abstract: This paper introduces concepts and mechanisms that allow to model security specifications and derive automatically the corresponding security implementations by transforming the original component model into a secured one taking into account sensitive data flows in the system. The resulted architecture ensures security requirements by construction and is expressed in the original meta model; therefore, it enables using the same timing analysis and synthesis as with the original component model.

  Contribution: I have been the initiator and co-author of the paper. The implementations were mainly done by the main author.

- **Paper D: 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'12), Eindhoven, The Netherlands, February, 2012

  Abstract: In this paper, we target the timing requirements of real-time embedded systems, and introduce an approach for choosing appropriate encryption algorithms at runtime, to achieve satisfaction of timing requirements in an adaptive way, by monitoring and keeping a log of their behaviors. The approach enables the system to adopt a less or more time consuming (but presumably stronger) encryption algorithm, based on the feedback on previous executions of encryption processes. This is particularly important for systems with high degree of complexity which are hard to analyze statistically.

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

- **Paper E: The Role of Schedulers in Model-Driven Development of Real-Time Systems**; Mehrdad Saadatmand, Mikael Sjödin, Naveed Ul Mustafa; MRTC report ISSN 1404-3041 ISRN MDH-MRTC-264/2012-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, March, 2012

  Abstract: Model-driven development is a promising approach to cope with the design complexity of these systems. It helps to raise abstrac-

tion level, perform analysis at earlier phases of development, and enables 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 timing behaviors of tasks, and schedules them can facilitate extraction of runtime information. This information is then used as feedback to the original model, in order to identify parts of the model that may require to be re-designed and modified. This is especially important for round-trip support in model-driven development of real-time systems. In this paper, we describe our work in providing these features by introducing a second layer scheduler on top of OSE real-time operating systems scheduler. The approach contributes to the predictability of the system 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.

2.3 Research Methodology

Figure 2.2 depicts and summarizes the key steps that have taken place in performing this research work.

Identification of NFRs in telecommunication systems, the general and specific challenges related to them, and the investigation of the state of the art and practice to understand what has been done to cope with those challenges resulted in a set of preliminary research goals. Providing and implementing solutions for those research goals also required investigation of the state of art and practice for other challenges and problems as well. The future work and directions which have been identified after implementing a solution for a research goal also resulted in new research goals which in turn required more investigation.

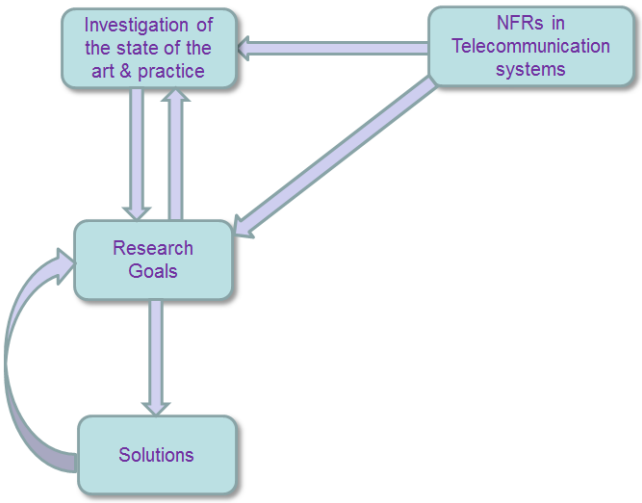

Figure 2.2: Research Methodology

Chapter 3

Conclusions

In this research work, we focused on the importance of NFRs in real-time embedded systems. We discussed different steps and points during the process of model-driven development of these systems where problems leading to the violation of the requirements can occur. It was shown how at the system model level, NFRs can be modeled along with other parts of the system, and traceability links among them can be established. Since the satisfaction of an NFR, especially in real-time embedded systems, can often affect the satisfaction of other NFRs of the system, we proposed a generic approach that enables system designers to compare design models with respect to the satisfaction level of their NFRs and perform trade-off analysis. This is achieved by considering the interdependencies of NFRs as well as the impact of the functional parts. This empowers system designers to make better decisions before continuing with the rest of the development process and generating implementations.

Focusing on timing and security requirements, two approaches for establishing balance between these NFRs were also introduced. These approaches target the problem of interdependency of NFRs and ensuring the balance among them. We believe that the suggested solutions can also be considered and modified for some other NFRs as well, but this needs to be further evaluated.

Regarding the proposed second layer scheduler, while it adds the missing necessary mechanisms that are required for detailed monitoring of timing events, its added overhead and performance costs should also be taken into account. For different systems, this additional overhead may be or not be acceptable. This solution is especially interesting for RTOSes where it is not

possible or not desirable to modify the kernel and the core scheduler. On the other hand, an alternative solution would be to instead include the features that we introduced as part of the second layer scheduler inside the core scheduler which helps with the reduction of the overheads. One point to note is that any added feature such as monitoring capabilities will have its performance costs anyway. This emphasizes the importance of efficient monitoring of system properties in real-time embedded systems which deserves further studies.

3.1 Future Work

While here we mainly looked at different steps during model-driven development of real-time embedded systems where violation of NFRs can occur, to further mitigate such violations it is also important to look at the transitions between each of these steps which in this context is model transformation (both model-to-model and model-to-text transformations). Investigation of transformation rules that preserve extra-functional properties of the system and thus can contribute to the satisfaction of NFRs is an interesting extension to this work. Such transformations that can be proven and provide preservation guarantees are of special interest in the development of safety-critical systems and in certification of the development process.

Also, by introducing our approach for trade-off analysis of NFRs and comparing different design alternatives, it would be interesting as a future work to study methods that help with the optimization of design models with respect to their NFRs. One of the problems in this direction that needs to be considered is the scalability of the approach for large systems and issues of state explosion type. Providing a more precise and detailed approach for quantification of NFRs is another topic for further research. Moreover, performing trade-off analysis of NFRs at runtime to re-configure the system according to different states/modes and match different Quality of Service (QoS) levels (e.g., if the available energy level goes below a certain limit) is another interesting future work.

Here we took timing and security requirements as an example of NFRs, to discuss dependencies and their impacts on each other and provided solutions (static and dynamic) for establishing balance among them. Another direction of this work could be to evaluate the applicability of the suggested methods for other NFRs such as timing and energy consumption.

Regarding the runtime monitoring of extra-functional properties, there are some issues that deserve special attention as future work. For some properties

and in some systems, the difference between the time point when the value of a property is requested and the time when the value is actually monitored and obtained can affect the accuracy and usefulness of the monitored value. This is important considering that the monitoring task needs also to compete with the (main) tasks that implement an application. Therefore, for such situations, we are considering to address this problem by providing a priority-based monitoring in the sense that by assigning priorities for different properties, the user can specify that the accuracy of which properties to monitor are more important for him. Based on these priorities, the monitoring task can perform differently to increase the accuracy of the monitored value while reducing the response time for obtaining it. We leave the implementation and evaluation of this method as a future work.

Bibliography

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

[2] 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.

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

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

[5] IEEE Standard for Conceptual Modeling Language Syntax and Semantics for IDEF1X/Sub 97/ (IDEF/Sub Object/). *IEEE Std 1320.2-1998*, 1998.

[6] S. Heath. Embedded systems design. EDN series for design engineers, ISBN: 9780750655460. Newnes, 2003.

[7] 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.

[8] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Nonfunctional 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] Martin Törngren, DeJiu Chen, and Ivica 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*. IEEE, August 2005.

[11] 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.

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

[13] Model-Driven Architecture (MDA). `http://www.omg.org/mda/`, Accessed: February 2012.

[14] Bran Selic. A systematic approach to domain-specific language design using UML. In *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, Washington, DC, USA, 2007. IEEE Computer Society.

[15] EAST-ADL Specification V2.1. `http://www.atesst.org`, Accessed: November 2011.

[16] Ingo Weisemöller and Andy Schürr. A comparison of standard compliant ways to define domain specific languages. pages 47–58, Berlin, Heidelberg, 2008. Springer-Verlag.

II

Included Papers

Chapter 4

Paper A: UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems

Mehrdad Saadatmand, Antonio Cicchetti and Mikael Sjödin
The Sixth International Conference on Software Engineering Advances (ICSEA 2011), Barcelona, Spain, October, 2011.

Abstract

Successful design of real-time embedded systems relies heavily on the successful satisfaction of their non-functional requirements. Model-driven engineering is a promising approach for coping with the design complexity of embedded systems. However, when it comes to modeling non-functional requirements and covering specific aspects of different domains and types of embedded systems, general modeling languages for real-time embedded systems may not be able to cover all of these aspects. One solution is to use a combination of modeling languages for modeling different non-functional requirements as is done in the definition of EAST-ADL modeling language for automotive domain. In this paper, we propose a UML-based solution, consisting of different modeling languages, to model non-functional requirements in telecommunication domain, and discuss different challenges and issues in the design of telecommunication systems that are related to these requirements.

4.1 Introduction

The nature of embedded systems such as resource constraints, close integration and interaction with the environment through sensors and actuators (which can also incur requirements on safety), timing characteristics and lack of traditional user interfaces all bring with themselves requirements that make the design of these systems complicated [1]. Much of this complexity is due to handling a big range of different requirements, solving conflicts and finding the right balance and trade-offs among them. Especially non-functional requirements such as security usually cross cut organizational structures and development teams. Thus traditional functional decompositions do not suit them. However, compared to functional requirements not much work has been done on non-functional requirements and lack of proper methods and techniques for modeling of non-functional requirements and their integration into the development lifecycle are felt [2].

UML profile for Modeling and Analysis of Real-Time Embedded systems (MARTE) [3] is one of the recent and major efforts on modeling Real-Time Embedded Systems (RTES) and the non-functional properties in these systems. MARTE enables detailed modeling of RTES and facilitates their analysis. On the other hand, there is a big variety of systems in RTES domain and to cover the specific aspects and needs of each group of those systems (subdomains), a customized modeling approach is necessary. Such an approach has been used in the automotive domain, leading to the definition of EAST-ADL profile [4] for modeling of vehicular systems.

This paper focuses on telecommunication systems and the aspects that modeling approaches for such systems should be able to cover regarding their non-functional requirements. We propose a UML-profiling approach consisting of features from different modeling languages to answer broader aspects in modeling non-functional requirements of telecommunication systems. One of these aspects is security. We will focus on security in this paper as an example for one of the intrinsic characteristics of telecommunication domain that is also not supported in EAST-ADL. Through an example, we show how it will be possible to model security requirements along with other aspects such as power, in one model while establishing traceability between high requirements and their refinements (lower level ones).

Regardless of the set of non-functional requirements that a subdomain in RTES has, modeling approaches for these systems should provide requirements traceability. This becomes even more important due to limited resources that systems in this domain have; while in other systems, it is usually a lot easier

to add extra resources to the system such as additional memory and that way fulfill a requirement. Therefore, a more careful balance and trade-off analysis between requirements is necessary in order to satisfy all of them in RTES domain. Having traceability links among requirements and also between requirements and design artifacts facilitates to perform impact analysis and identify the effects a change on one requirement can have on other parts of the system.

To cover different aspects regarding non-functional requirements in telecommunication systems, we suggest a UML profiling solution consisting of concepts from SysML [5] for traceability, and MARTE for modeling general non-functional properties and their analysis. For security requirements, which are inherent in telecommunication domain but are not covered by MARTE, we adopt from available UML profiles for security, namely UMLsec [6]. Also since MARTE, SysML and UMLsec are UML profiles, they are faster for developers using UML to catch on and they also serve as a possible unifying factor between development departments. A comparison of different ways to define Domain Specific Languages (DSL) and the benefits of each approach are provided in [7, 8]. It is also important to note here that combining different UML profiles is not a trivial task as it may seem and it can incur different problems such as semantic conflicts. These issues are discussed in an interesting work in [9].

The contributions of this paper can be summarized in the following points:

- Showing an approach on how to model non-functional requirements in telecommunication systems
- Identification of issues that should be taken into account in modeling those requirements and the modeling concepts to cover them

As a guideline we use our observations during a project we have done at Ericsson plus the results of other studies such as [10, 2] to describe the modeling challenges. In Section 4.2, we provide a deeper understanding of telecommunication systems, its characteristics and needs, and the problems observed around non-functional requirements in those systems. We discuss the related work and have a look at some modeling solutions in automotive domain in Section 4.3. In Section 4.4, we describe the ingredient concepts of our proposed approach for modeling non-functional requirements in telecommunication domain by highlighting some key relevant concepts offered in SysML, MARTE and UMLsec. Section 4.5 shows the application of the method as a usage example. In Section 4.6, we compare the features of our suggested approach with

those of EAST-ADL in automotive domain, and finally, in Section 4.7 we summarize the work and suggest different areas that need to be studied as future work.

4.2 Motivations

The observations and results in this section are achieved through collaboration with Ericsson engineers (Stockholm, Sweden) and gathered through several meetings with different teams, such as Radio Base Stations (RBS) development group, during CHESS project [11] at Ericsson.

4.2.1 Telecommunication Systems

As a type of real-time embedded systems, telecommunication systems have specific characteristics, which incur certain requirements and prioritization of some requirements over the others. These systems need to be secure, are highly distributed, have a dynamic nature, require massive processing capacity and high availability (99.999% availability, which is sometimes referred to as *five nines*), and need to be scalable. The distribution in these systems can be regarded in two perspectives: the distribution inside one node (such as using multicore solutions and distribution of software functions among different processing units) and also the geographical distribution of nodes across different regions and the communication among them.

Typically, telecommunication networks consist of many different types of nodes such as Radio Base Stations (RBS), Radio Network Controllers (RNC), Media Gateways (MGW) and others that span across a big geographical area and communicate over different kinds of lines.

Regardless of the integration and interconnection of different nodes in the network, design of each node is a big complex challenge in itself. For example, an RNC can easily contain between 500 to 700 CPUs, with software functions spanning across several CPUs. This number, however, is decreasing as new processors with higher capacities are produced. This reduction is important for the total cost, power consumption and heat generation of systems. As for functionality and services, in a typical telecommunication system a big number of connections should be established, routed and managed per second. Besides, cost calculation should also be done on them. Moreover, a typical telecommunication system can have a life span of about 20-30 years. Thus upgrade-ability and maintenance of such systems is also of great importance. Software upgrade

should be done in such a way to have the least effect on the availability of the system. That is why requirements such as hot-swapping and plugging and the ability to perform restarts at different granularity levels (a single board, collection of boards or a complete node) are highly desirable and demanded in this domain.

4.2.2 Problems with Non-Functional Requirements

Due to the hierarchical and subsystem structure of telecommunication systems, first overall non-functional requirements are defined on the system and then they should be refined several times and in each step more concrete and design-decision information is added. However, not all requirements get refined, and as discussed in [10], this leads to weak traceability chains. What can happen is that the requirements that are defined on the system model are *consumed* (meaning that they are read and implemented in the system) and no explicit connection between the design artifact and the requirement leading to that design decision gets established. Also for verification, most of the requirements are tested on a reference configuration and then if some requirements are not met, changes are applied on the system model and again a reference configuration is built with the new requirements. Basically, there are two general issues with this current approach:

- Poor support for traceability of requirements to design artifacts
- The feedback loop for analysis of non-functional requirements takes much time and effort and the wish is to be able to perform verification of non-functional requirements at earlier phases

The organization in large companies usually have a hierarchical structure, which suits the actual system hierarchical structure as mentioned above. According to the study done in [2], this organizational structure matches the system structure well, as subsystems tasks and modules are allocated to specific departments and thus is more suitable for functional requirements. However, this is not the case for non-functional requirements. The problem as mentioned in [2] is that the autonomy of departments at the lowest levels of hierarchy makes management of non-functional requirements harder and that the decisions about these requirements should be done at higher levels of hierarchy and aligned and managed from top to down. This problem becomes more obvious with certain types of requirements such as security, usability and user-interface characteristics, which should be aligned in different subsystems.

Thus, non-functional requirements can easily cross-cut organizational structure of a company and therefore a methodology that works for functional decomposition may stop to work for non-functional requirements.

In such organizations, it also happens that different teams may have different interpretations and definitions for some non-functional requirements, which can cause problems for communication between the teams. On the other hand, this also means that people from different teams may talk about a specific requirement using different terms. If we can provide a consistent way of modeling non-functional requirements and a mechanism to establish associations between a requirement or a design artifact and its source requirement, such problems can be mitigated and detected more easily. Also as can be implicitly noticed from the discussion in previous section, there are many requirements that have conflict with others, and trade-off decisions to balance them need to be taken. However, these compromisations and decisions, which may be made inside a subgroup, are somewhat unknown to upper levels and are only known by some engineers working in that section. For example, it is quite common that specific tweaking and settings in the code on bandwidth or memory usage are applied to ensure a certain level of balance between performance and maintainability of the system. Such decisions even if documented are hard to follow and track later on, especially for upper levels in the organizational hierarchy. On the other hand, some requirements that are decided on higher levels are lost in the transition to go to development teams in lower levels of hierarchy. This observation is also in alignment and confirmed by the study in [10], which states the problem as non-functional requirements ”are not always available when needed”. These issues can be alleviated by applying traceability (which can be traversed back and forth) between requirements and using a better form for representation and documentation of non-functional requirements.

4.3 Related Work

Requirement Modeling: Telecommunication Standardization Sector (ITU-T) have offered several languages for system modeling in telecommunication domain. Each of these languages try to target different aspects and phases in system development. For example, Message Sequence Chart (MSC) is used for modeling asynchronous interaction scenarios. Specification and Description Language (SDL), which has both textual and graphical representations, uses block, process, channel and signal concepts to describe behavior in communicating real-time systems. At higher abstraction layers and for model-

ing requirements, ITU-T has suggested User Requirements Notation (URN). URN consists of two notations: Goal-oriented Requirement Language (GRL) to model goals and non-functional requirements and Use Case Maps (UCM) to describe functional scenarios. GRL is used to capture informal system goals, specification and rationals. We refer interested readers to ITU-T website [12] for more information on these languages. Some efforts have been done to define these languages as UML profiles such as [13].

As for general UML-based approaches in RTES domain, MARTE with its expressive power and formal semantics enables capturing non-functional requirements in more formal ways and with necessary details for performing analysis earlier in system development phases. For system engineering, modeling general requirements and the relationships among them, SysML offers Requirements model, and semantics and notations for requirements traceability.

Modeling Security Requirements: There have been efforts on modeling and analysis of security aspects using UML to bring them into earlier phases of development . For example, SecureUML [14] focuses on modeling Role-Based Access Control (RBAC) by extending UML as a profile, while AuthUML [15] is a framework for analysis of access control in the specification phase and thus less suited for code generation. UMLsec on the other hand, uses stereotypes and tag values for modeling general security aspects such as secure links, connections, RBAC, secure information exchange, etc. to enable analysis and early automatic verification (which also matches our goal for early analysis of requirements). A comparison between SecureUML and UMLsec for modeling role-based access control is done in [16]. The UMLsec analysis tool suite can help to identify parts of the model that do not match a specified security requirement. This enables to perform a level of security analysis on the model and find inconsistencies before going into implementation phases. As for other works in this area, the study in [17], for example, introduces stereotypes to specify vulnerabilities so that developers can notice them and avoid in implementation. It also claims that these specifications can be used to generate test cases for security. Article [18] tries to merge Mandatory Access Control (MAC) and Discretionary Access Control (DAC) with RBAC. It is a good work for modeling access control aspects, but lacks other security aspects of UMLsec and their analysis. Doan and Demurjian [19], on the other hand, discuss security analysis based on RBAC and MAC in use-case and class diagrams. Houmb and Hansen [20] introduce SecurityAssessmentUML, which is intended to capture and document the results of risk (i.e., vulnarabiltiies, threats, etc.) identification and analysis. Discussion and comparisons of differ-

ent UML-based security models can be found in the related work sections in [14, 6, 17, 18, 19].

Requirement Modeling in Automotive Domain: As an example of a UML-based domain-tailored approach, EAST-ADL has been developed in automotive domain for modeling software architecture and electronic parts of a system. By complementing and making use of general available modeling solutions in RTES domain, EAST-ADL tries to cover the specific requirements of automotive domain. It adopts concepts from UML, AADL [21] and SysML to provide modeling semantics aligned with AUTOSAR [22] specification. AUTOSAR focuses on lower design levels such as component model, software modules, control units, APIs and implementation parts of automotive systems.

For modeling requirements, EAST-ADL makes use of SysML requirements semantics and specializes them to match automotive domain (e.g., definition of timing, delay and safety requirements). However, it does not provide enough features to enable some analyses such as scheduling and timing verifications earlier than implementation phase [23]. There are studies such as [23] that suggest decorating EAST-ADL models with some features from MARTE such as timing and allocation packages to enable early scheduling analysis. TIMMO project [24] is one of the efforts using this idea to complement timing model of EAST-ADL for automotive domain. In general, EAST-ADL and its requirement model may not be appropriate and compatible as a whole for requirements in telecommunication domain. It does not cover security aspects, which are important for telecommunication systems, is aligned with EAST-ADL's specific abstraction levels, and is based on concepts like ECU, VehicleFeature, AutosarSystem, and Sensor which are not relevant for telecommunication systems. In order to better capture requirements of telecommunication systems that originate from their specific characteristics such as intensive performance demands, distribution, use of multicore solutions, virtualization and hierarchical schedulers, etc. a tailored solution for this (sub)domain is required.

4.4 Suggested UML profile

Adopting a model-based approach for the development of telecommunication systems helps to raise the abstraction level and cope with the design complexity. This also targets the challenge to shorten the feedback loop and enable analysis in earlier phases of development.

In this section, the key concepts that a desired UML profile for telecommunication systems should be able to offer are discussed. We explain traceabil-

ity concepts from SysML, modeling general non-functional requirements with MARTE highlighting its relevant and interesting features for telecommunication domain and how to model security aspects along with an example of its analysis. Later in section 6, we compare the features of our suggested UML profile with EAST-ADL.

4.4.1 Modeling Traceability Using SysML

For modeling of requirements, SysML provides a specific diagram, which can be a solution to the issues regarding management of non-functional requirements of telecommunication systems identified in previous sections. An important feature of SysML is to represent requirements as first-class model elements. So requirements are included as parts of the system architecture and have semantics [25]. This also enables establishing relationships between requirements and other model elements showing, for example, design artifacts implementing and satisfying a requirement. It is possible to decompose requirements and create a hierarchy of requirements, which is needed to cope with the complexity of requirements faced in telecommunication domain. SysML provides different types of associations among requirements, which include: *copy*, *deriveReqt*, *satisfy*, *verify*, *refine* and *trace*.

The counterpart of these associations are *derivedFrom*, *satisfiedBy*, *refinedBy*, *tracedTo*, *verifiedBy* and *master* properties that a requirement element can have. For example, *satisfiedBy* property of a requirement element contains the information of the model element that satisfies this requirement (counterpart of *satisfy* association). This way, SysML facilitates traversing back and forth between requirements and also model elements from high level departments in organizational hierarchy to lower level departments and development teams.

Another feature that SysML provides is requirements table. Requirements table provides traceability information for requirements in a single view, which is very helpful in managing the big number of versatile requirements in telecommunication systems. In this tabular representation of requirements, information such as requirements properties and types, dependency relationships with other elements/requirements and other information such as design rationale and test procedures may be included. By going through this table, it is possible to analyze the change (e.g., modification, deleting) effect of one requirement on other requirements in the systems. So basically, by providing different types of association and dependency and the tabular representation of requirements, SysML can answer problems identified for traceability and

impact analysis of requirements in a complex and hierarchical telecommunication system. Moreover, by using stereotypes it is possible to extend SysML, which makes it very flexible to add new semantics such as new types of associations or requirements. An example of this extension is provided in [25], where three stereotypes for functional requirements, non-functional requirements and external interface are defined and used to model a system.

4.4.2 MARTE for Non-functional Requirements and Analysis Support

To represent the properties of non-functional requirements such as timing constraints in a formal way, MARTE provides rich modeling semantics. MARTE profile consists of different subpackages and in this section we try to identify packages and semantics in them, which serve to represent the type of non-functional requirements we identified in a telecommunication system.

MARTE NFP_Types, Value Specification Language (VSL) and the stereotypes defined in NFP package (Non-Functional Properties) help to define different non-functional properties specific to different domains. NFP package makes it possible to define percentage, dimensions, measurement precision and similar concepts for non-functional properties. Examples of basic NFP types already defined in MARTE type library can be power, frequency, duration, energy, weight, length, arrivalpattern (periodic,aperiodic, sporadic), price, etc. For time specifications, MARTE offers the time package and representation of time in MARTE can be in the form of a physical (continuous or discretized) or logical clocks (processor cycles, engine rotation, algorithmic steps...). The concept of multiform time provided in MARTE is very useful for telecommunication domain, which has already started heading for multi- and many-core solutions. The semantics to model the execution platform (operating system, virtual machines, hardware) are packaged in Generic Resource Modeling (GRM), Software Resource Modeling (SRM) and Hardware Resource Modeling (HRM). With SRM it is possible to model concepts such as resources, services, concurrency and mutual exclusion features in a Real-Time Operating System (RTOS) as well as virtual machines, which are used in telecommunication systems. The stereotypes in HRM package enable modeling of processing units, different levels of memory, devices and their physical aspects such as layout in the system, power consumption, and heat dissipation. These concepts can be used to target non-functional requirements such as cost sensitivity, execution capacity, and environmental requirements (layout, size, structure, etc.).

An interesting feature provided in GRM is the modeling of primary and

secondary schedulers, which enables modeling of systems having hierarchical schedulers. This is helpful for telecommunication domain in which, use of hypervisors and virtual operating systems on top of another operating system is common.

To model dependability requirements (reliability, availability, maintainability, safety), which is an important feature of telecommunication systems, there is a suggestion for an extension to MARTE, that is introduced in [26] as Dependability and Analysis Modeling (DAM) (sub)profile and offers relevant concepts such as threats (fault, failure, error, hazard, accident), maintenance, redundancy, etc.

4.4.3 Covering Security Aspects

Security in embedded systems is becoming more important and gaining greater attention. More mechanical parts are replaced by computer systems and the use of wireless technologies for communication between different units is becoming a dominant trend. In automotive domain for example, features such as traffic and accidents notification systems, built-in bluetooth devices and distance calculator between cars are representatives of such cases that require communication with other cars and devices. Such features along with electronic access controls (e.g., access to the vehicle internal bus and electronic locks) also open up the system for more security threats.

Due to the nature of systems in telecommunication domain, which naturally involve long distance communications and at a big level of distribution and scalability with many nodes and access points on the way, security aspects have always been an unavoidable part. A single telecommunication node such as a Radio Base Station (RBS) can serve different requests from different sources and these operations should be kept separate from each other keeping data intact and safe from interference. It becomes more critical when we add to the picture other services in the system such as call cost calculation for users and (recently) data traffic including images, emails, and other sensitive and personal information. However, in the design of a system, security considerations should not be considered as an add-on, but they should be taken into account from early phases of design.

UMLsec covers a broad scope and has a versatile tool suite for analysis. Using that we can complement modeling of security requirements as first class entities. With the help of SysML, relationships between them and other non-functional requirements and also design artifacts can be added and detailed non-functional properties using MARTE can be specified for them if neces-

sary. So for example, it becomes possible to model nodes in a system as resources using MARTE GRM package, and then define necessary users, roles and communication security requirements between the nodes using UMLsec profile. The relation between these elements and the source requirement element incurring such security design can be established using SysML requirements concepts.

UMLsec offers several stereotypes such as *Internet*, *wire*, *LAN*, *encrypted* for physical links between nodes. These concepts can be applied on communication links in telecommunication systems. Each link type in UMLsec is defined as prone to different types of threats (*read*, *insert*, *delete*) from different attackers. For example, a link stereotyped as *LAN* or *wire*, has no threat from a default (external) attacker. However, an insider attacker can still pose *read*, *insert* and *delete* threats regarding the packets and information transferred on such a link. If it is needed to define new types of links, attackers or threats, UMLsec allows this. The new concepts can be defined in the UMLsec appropriate format in an XML file, so that the analysis tool can perform correct analysis based on these custom concepts on a model that makes use of them. In this sense, the analysis tool is flexible and extensible. *Secrecy* stereotype that is used on dependency relationships applies a secrecy/confidentiality requirement on the elements of dependency base class. This way, we specify that there is a secrecy requirement for the involved elements.

4.5 Modeling a Security Requirement Using the Suggested Profile

In a typical 3G telecommunication network, different groups of Radio Base Station (RBS), Radio Network Controller (RNC) and Media Gateway (MGW) nodes are connected and communicate. When a Mobile Equipment (ME) wants to join the network, it starts communicating with RBS and authenticating itself to the system. Security operations such as key exchange take place through the communication path from the mobile equipment to the RNC. In our case study, we have two RBS 3202 nodes that communicate with an RNC. The output power requirements for RBS 3202 are as follows:

Req1: Optimized Power 15W, Standard Power 20W, High Power 30W and Dual High Power 60W.

One of the security requirements that exist for the connection between the RBS and RNC is:

Req2: Data communication between RBS nodes and RNC should only be

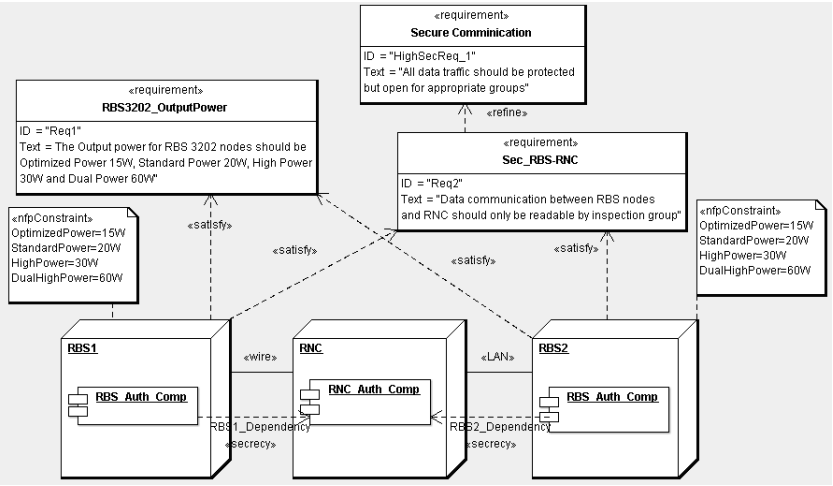

Figure 4.1: Security Requirement on RBS Nodes

readable by inspection group.

The second requirement incurs that no one from outside and also inside of the network should be able to read the data traffic on the links between RNC and RBS except users in the inspection group. Thus the data should be encrypted using a specific key for this group. We try to violate this in our example model by using unencrypted links and then perform analysis on the model.

As shown in Figure 4.1, the requirements and the relationships between them and design artifacts are modeled using SysML concepts. MARTE non-functional concepts (i.e., *nfp, nfpconstraint, PowerUnitKind and NFP_Power*) are used for modeling output power requirements of RBS nodes. Security concepts in our model are represented using UMLsec stereotypes. The link between RBS1 and RNC is marked with *wire* stereotype and the one between RBS2 and RNC is marked with *LAN* stereotype (in UMLsec wire and LAN are two different security stereotypes that can incur different security characteristics).

Doing analysis using UMLsec analysis tool on the model yields the result that is shown in Figure 4.2. The important part in this analysis output (marked with *) is that *LAN* and *wire* links are not readable by a default (external) attacker thus the model satisfies the secrecy requirement for this attacker type,

```
::::::::Against Default Attacker
=======Here begins the verification
The name of the dependency is RBS2_Dependency
The stereotype of the communication link of the dependency RBS2_Dependency is LAN
The stereotype of the dependency is: secrecy
* The UML model satisfies the requirement of the stereotype secure links.
...
::::::::Against Insider Attacker
=======Here begins the verification
The name of the dependency is RBS2_Dependency
The stereotype of the communication link of the dependency RBS2_Dependency is LAN
The stereotype of the dependency is: secrecy
* The UML model violates the requirement of the stereotype secure links, but
it has been fixed.
...
```

Figure 4.2: Result from UMLsec Analysis Tool

but an insider attacker on LAN or wire can access the information and therefore the model violates the requirement. Although UMLsec has a general *encrypted* stereotype to label encrypted communications, it is also possible to define a custom stereotype for example as "*Uniquely encrypted by SIM ID*" and define different threats that different attackers can pose on these links such that only inspection group users can have access. Then we can use this stereotype on the links instead of *LAN* and *wire* that we used earlier, to create a model that satisfies the requirement and verify it with the analysis tool.

4.6 Discussion

As mentioned in the related work section, EAST-ADL is a modeling solution for automotive domain that is built using a similar approach to what we proposed here by adopting from several UML profiles. It is successfully accepted in the automotive domain and its usage together with AUTOSAR is gaining more momentum. In table 4.1, a comparison of capabilities of our suggested solution using MARTE plus SysML and UMLsec against those of EAST-ADL is presented, with a focus on modeling concepts and features that are necessary for NFRs in telecommunication domain (e.g., processing capacity and memory consumption that are important for performance analysis). It summarizes the concepts we discussed and identified in previous sections. The star mark in the table is used to indicate that the feature is not enough/fully supported, such as the dependability modeling in our approach. However, it can be covered by using the DAM profile introduced earlier, which is built as an extension to MARTE. Modeling of time for schedulability analysis support in EAST-ADL needs also to be complemented (as is investigated in [24]).

Modeling Feature	Our Approach	EAST-ADL
Generic NFRs (SysML Style)	✓	✓
Traceability of NFRs	✓	✓
Timing, Clock, Schedulability Support	✓	*
Memory consumption	✓	✗
Processing capacity	✓	✗
Power consumption	✓	✗
Virtual machines and hierarchical schedulers	✓	✗
Hardware platform	✓	✓
Multicore	✓	✗
Allocation and Deployment	✓	✓
Communication media	✓	✗
Safety	✗	✓
Security	✓	✗
Variability (product families)	✗	✓
Methodology (e.g., abstraction levels)	✗	✓
Dependability (e.g., fault, error...)	*	✓
Synchronization mechanisms	✓	✗
Arbitrary Non-Functional Properties	✓	✗
Component model	✓	(AUTOSAR)

Table 4.1: Comparison of the suggested UML-based modeling solution with EAST-ADL of Automotive domain.

From the table, it can be seen that by *tailoring* a UML profile for telecommunication systems based on the concepts in the three available profiles we discussed in this paper (MARTE, SysML and UMLsec), it is possible to better cover the requirements of telecommunication systems, than just re-using only EAST-ADL modeling semantics from automotive domain, which are tailored for the needs of systems in that domain. Security is one of the specific needs of telecommunication systems that is not supported by EAST-ADL and has not been in the main focus in automotive domain (so far). While on the other hand, safety requirements, which are very important for automotive systems, are explicitly supported in EAST-ADL. For differences between safety and security requirements, interested readers can refer to [27].

While in this paper, we discussed a UML-based solution by adopting and tailoring already existing profiles, other methods of defining a specific language for modeling telecommunication systems are, of course, possible. However, although designing a domain specific language from scratch may match the needs of telecommunication systems better, it also implies the need to design dedicated modeling tools, and additional costs for training the users to learn the new language. On the other hand, some of the benefits of a solution based on UML are that many users are already familiar with UML, and thus,

the learning curve is smaller. Also, there are already many tools for creating UML models which can be used 'out of the box' [7, 8]. One point to remember though is that, as mentioned before, combining different UML profiles can be problematic in some cases. For example, there is FlowPort both in SysML and MARTE. However, the semantics of FlowPort in SysML are different from those of MARTE. A systematic approach is suggested in [9] to ensure consistency in merging UML profiles.

Regarding the management of models, based on the features of the modeling tool, there can be several scenarios. For example, different models can be created for different aspects of the system. This can also help with the analysis, as one model for each type of analysis can be created. However, maintaining consistency between different models of the system and redundant information modeling are some of the challenges of this approach. Another scenario could be to have one single model for the system, and then have the modeling tool provide different views of the core model. This way, a user can just focus on the aspects of his/her interest in each view, while modifications are persisted into one single model representing the system. This method is under development in CHESS project [11].

As for the analysis of the models, although this topic is not the main focus of this paper, but we provide some hints here. Basically, the process of analysis can be different for various analysis tools, and depending on which types of analysis are of interest for different end-users. In case of having just one single model of the system, if an analysis tool can ignore non-relevant model elements and perform analysis on the relevant parts, the model can be fed as input to the analysis tool directly. However, if non-relevant model elements may cause problems for the analysis, then it is possible to use model transformation to extract only the relevant ones into a new model appropriate as input for the analysis tool. Also, if the input model of any analysis tool has its own specific meta-model, then model transformation techniques can again be used to transform the original model into a new model conforming to the meta-model of the analysis tool.

4.7 Conclusion and Future Work

In this paper, we discussed several challenges in modeling non-functional requirements in telecommunication domain. We also suggested a modeling approach for representation of non-functional requirements and their properties in this domain. Our approach was to consider telecommunication systems

as a subdomain of RTES and therefore adopt from available modeling solutions for non-functional requirements and their analysis that already exist in RTES domain. Some concepts of MARTE that can cover the requirements of telecommunication systems were highlighted. For traceability aspects, SysML and the features it provides in establishing traceability in modeling of non-functional requirements were introduced. Finally, as a specific and intrinsic requirement in telecommunication domain, it was shown how it is possible to model and analyze security that is addressed in our suggested approach by adopting UMLsec. This way, we showed not only how it is possible to model different types of non-functional requirements, but also how model-based analysis can help with the need to perform analysis of non-functional requirements at earlier phases of development and therefore reduce time and cost. In CHESS European project [11], we are developing a similar solution by using subsets from MARTE, SysML and DAM profile (without security considerations yet) to generate code for telecommunication systems (in this case, Ericsson platforms) considering and preserving non-functional requirements modeled using the mentioned subsets.

As further studies, it is necessary to augment the suggested approach in this paper, such as introducing it as part of a well-structured methodology similar to the methodology suggested in [28]. This methodology is more suited for automotive domain as it makes use of EAST-ADL and its abstraction levels. Applicability of the same concepts to telecommunication domain could be an interesting topic to investigate. Especially that EAST-ADL offers concepts for modeling *variability* requirements, which can be very useful in telecommunication domain for modeling product families, targeting cost-sensitivity non-functional requirements and performing cost analysis.

Also other challenges that exist regarding non-functional requirements in a model-based development approach can be guaranteeing and preservation of these requirements on the target platform, introducing runtime adaptability and reconfiguration based on the requirements and handling their violations.

As a last note, in this paper we set the basis for a UML-based solution for telecommunication systems similar to EAST-ADL in automotive domain. While it was demonstrated how we can relate high-level and abstract representation of an NFR such as security with its lower level realizations and perform security analysis on it, a full scale solution needs contributions from different industrial partners active in the domain as has been done in the process of defining EAST-ADL and AUTOSAR.

Bibliography

[1] Tom Henzinger and Joseph Sifakis. The embedded systems design challenge. In *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*. August 2006.

[2] 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.

[3] MARTE specification version 1.0 (formal/2009-11-02). `http://www.omgmarte.org`, Accessed: August 2011.

[4] EAST-ADL Specification V2.1. `http://www.atesst.org`, Accessed: August 2011.

[5] OMG SysML Specifcation V1.2. `http://www.sysml.org/specs.htm`, Accessed: August 2011.

[6] Jan Jürjens. *Secure Systems Development with UML, ISBN: 978-3-540-00701-2*. Springer, 2005.

[7] Ingo Weisemöller and Andy Schürr. A comparison of standard compliant ways to define domain specific languages. pages 47–58, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Bran Selic. A systematic approach to domain-specific language design using uml. In *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, 2007.

[9] Florian Noyrit, Sébastien Gérard, François Terrier, and Bran Selic. Consistent modeling using multiple uml profiles. In *Model Driven Engineering Languages and Systems*. 2010.

[10] 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.

[11] CHESS Project: Composition with Guarantees for High-integrity Embedded Software Components Assembly. `http://chess-project.ning.com/`, Accessed: August 2011.

[12] Telecommunication Standardization Sector (ITU-T). `http://www.itu.int/en/pages/default.aspx`, Accessed: June 2011.

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

[14] 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, 2002.

[15] Khaled Alghathbar and Duminda Wijesekera. authuml: a three-phased framework to analyze access control specifications in use cases. In *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 77–86, New York, NY, USA, 2003. ACM.

[16] A. Cenys, A. Normantas, and L. Radvilavicius. Designing role-based access control policies with uml. In *Journal of Engineering Science and Technology Review*, volume 2, pages 48–50, 2009.

[17] Karine P. Peralta, Alex M. Orozco, Avelino F. Zorzo, and Flávio M. Oliveira. Specifying security aspects in uml models, September 2008.

[18] S. Demurjian, J. Pavlich-Mariscal, and L. Michel. Enhancing uml to model custom security aspects. In *Proceedings of the 11th International Workshop. on Aspect-Oriented Modeling*, October 2007.

[19] Thuong Doan and Steven Demurjian. A.: Mac and uml for secure software design. In *In: Proc. of 2nd ACM Wksp. on Formal Methods in Security Engineering*, pages 75–85. ACM Press, 2004.

[20] Siv Hilde Houmb and Kine Kvernstad Hansen. Towards a uml profile for security assessment. In *Workshop on Critical Systems Development with UML*, 2003.

[21] AADL. The Architecture Analysis & Design Language: An Introduction . `http://www.aadl.info/aadl/currentsite/aadlstd.html`, Accessed: August 2011.

[22] AUTOSAR Home Page. `http://www.autosar.org/`, Accessed: August 2011.

[23] Saoussen Ansi, Arnaud Albinet, Sara Tucci-Pergiovanni, Chokri Mraidha, Sebastien Gérard, and François Terrier. Completing east-adl2 with marte for enabling scheduling analysis for automotive applications. In *Embedded Real Time Software and Systems Conference (ERTS)*, Toulouse, France, 2010.

[24] TIMMO Project. `http://www.timmo.org/`, Accessed: August 2011.

[25] Michel dos Santos Soares and Jos L. M. Vrancken. Model-driven user requirements specification using sysml. *Journal of Software*, 3:57–68, 2008.

[26] S. Bernardi, J. Merseguer, and D. Petriu. A dependability profile within marte. *Journal of Software and Systems Modeling*, 2009.

[27] Eirik Albrechtsen. Security vs safety. NTNU - Norwegian University of Science and Technology `http://www.iot.ntnu.no/users/albrecht/`, Accessed: May 2011.

[28] A. Albinet, J-L. Boulanger, H. Dubois, M-A. Peraldi-Frati, Y. Sorel, and Q-D. Van. Model-based methodology for requirements traceability in embedded systems. In *Procs. of ECMDA'07*, Haifa, Israel, June 2007.

Chapter 5

Paper B: Modeling and Trade-off Analysis of NFRs

Mehrdad Saadatmand, Antonio Cicchetti and Mikael Sjödin
MRTC report ISSN 1404-3041 ISRN MDH- MRTC-267/2012-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, April, 2012.

Abstract

In this paper we introduce a generic approach to analyze system design models with regard to the satisfaction of their Non-Functional Requirements (NFRs) to enable the evaluation of their NFRs' trade-offs. NFRs and their satisfaction become especially critical and deserve more attention in certain application domains such as real-time and embedded systems. This is mainly due to the constraints and resource limitations in these systems. A design that cannot achieve the functionality of the system under these limitations can mean a failure. However, one big issue is that NFRs are interconnected and cannot be considered in isolation as they can have direct impacts on each other like security and performance. This means that a careful balance and trade-off analysis among NFRs is necessary. In doing so, the role of functional parts that contribute and are implemented to satisfy an NFR should also be taken into account. We focus on these needs and identify what information about NFRs is required in order to perform trade-off analysis and comparison of design models. We propose and explain our approach to incorporate this information into system models using UML profiling method to annotate model elements with necessary information and then calculate satisfaction values of NFRs using model transformation technique.

5.1 Introduction

The role of Non-Functional Requirements (NFR) throughout the software development process is gaining more and more attention, especially that the improper handling of NFRs has been identified as an important source of failure in many projects [1, 2]. However, NFRs are still rarely taken into account seriously and treated as first-class elements in software development, and are mostly considered as an after-thought in the final phases of development [3, 4]. There are several reasons that contribute to this fact. For example, NFRs are usually stated in an informal way and at a high abstraction level, therefore, appropriate methods and tools are required that can incorporate them at earlier phases of development and along with functional requirements. The importance of the integration of NFRs with functional parts becomes more apparent when we realize that different NFRs on the same functionality can result in different design decisions and implementations [5].

Proper handling of NFRs is even critical in certain domains such as embedded and real-time systems. Successful design of these systems depends heavily on the satisfaction of their non-functional requirements. This is mainly due to the constraints and limitations of these systems in terms of available resources [6]. Therefore, an embedded system needs to achieve its functionality under these limitations and NFRs. The problem is that each design decision can have impact on the system's NFRs and the system designer should be able to identify and evaluate these impacts. For example, if there is an execution time constraint on a component responsible for sorting numbers, then choosing a slower algorithm than a more time-optimized one can lead to the violation of system requirements and deviation of its behavior from the desired one. This situation becomes more complicated when we realize that NFRs not only crosscut different parts of a system, but can also have mutual impacts on each other. For example, choosing a time-optimized and faster sorting algorithm can help with the satisfaction of the timing requirements but may require more memory and thus violate memory constraints. Also, an NFR such as security crosscuts different parts of a system (e.g., user interface, database, communications and network transmissions) and affects some other NFRs like performance [7]. Therefore, the tools and methods that are suggested for handling of NFRs should not only be able cover and cope with their informal nature, high abstraction level, and integration with functional ones, but also should be able to help with identifying their dependencies and impacts on each other to enable analysis of their trade-off.

Model-Based Development (MBD) helps to raise the abstraction level and

cope with the design complexity of systems. This can make it especially interesting for the design of real-time embedded systems which can have high degree of complexity. By raising the abstraction level, MBD also enables analysis at earlier phases of development which in turn enables identification of problems before reaching the implementation phase [8]. Since NFRs have a high abstraction level [3] and also MBD provides views of the system at higher abstraction levels, providing an MBD approach to incorporate NFRs into models is deemed more appropriate especially in integrating NFRs with functional aspects of the system.

In this paper, we propose a UML profile [9] for trade-off analysis of generic NFRs. The analysis in this work implies analyzing the dependencies and impacts of NFRs (irrespective of their type, e.g., performance, security...) as well as the functional features of the system to provide system designers with a better insight into how good a system design is in terms of the satisfaction level of its NFRs. The main contributions can thus be summarized in the following points:

- providing an approach to capture and model NFRs in an open and generic way and include them with functional features as well as the dependencies and relationships among them,
- enabling designers to evaluate and compare different design models with regard to their NFRs and identify designs which can result in better overall satisfaction of NFRs,
- identifying deviations in the satisfaction of system's NFRs and highlighting potentially problematic parts that deserve more attention.

Enabling trade-off analysis of NFRs through a UML profile has several key benefits. UML is a standard modeling solution already adopted by industry and there are many design and analysis tools based on it. By offering a UML-based solution for trade-off analysis of NFRs, it becomes possible to make use of currently available tools. Also, the learning curve for developers already using UML will be less which implies both cost and time savings for companies [9]. Moreover, the approach we propose here tries to extend design models with necessary information to enable trade-off analysis of NFRs, and thus can enable using already designed system models and can save modeling efforts. We propose stereotypes in our profile by which model elements can be decorated and related with the necessary information that enable designers to consult models for specific information about NFRs and trade-off analysis of them.

The remainder of the paper is structured as follows. In Section 5.2, we have a look at NFRs in general, including their definition, characteristics, and the issues around them. In Section 5.3, we discuss the required characteristics and information that a solution for managing trade-offs of NFRs should be able to offer. Section 5.4 explains the profile structure and concepts. Section 5.5 shows an application of the profile in a portion of a mobile phone design and the analysis is then performed on the annotated model. In Section 5.6, we will have a look at other related works and finally in Section 5.7, conclusions are made and future directions are explained.

5.2 Non-Functional Requirements

Requirements are generally divided into two main categories: functional and non-functional. In the simplest form, functional requirements are those which define what the system should do, while the term non-functional requirement is used for requirements which specify how a system should perform or as suggested in [10] "a non-functional requirement is an attribute of or a constraint on a system". There is a big number of suggested definitions for non-functional requirements which are discussed in [11]. These requirements are usually described with terms that end with 'ility' such as availability, 'ity' such as atomicity, while a few other ones such as performance and user-friendliness do not follow this pattern. According to the IEEE standards, 610.12-1990 and ISO/IEC/IEEE 24765:2010(E) [12, 13], the following definitions is provided for non-functional requirements: "a software requirement that describes not what the software will do but how the software will do it (i.e., design constraints)".

One concept that is often confused with NFR is the concept of Non-Functional/Extra-Functional Property (NFP/EFP). The former can be considered as an expression of a need (possibly informal), and the latter as a statement that is usually asserted formally and can be therefore proven and analyzed (e.g., calculated response time of a component). This implies that "a requirement can require that a certain property holds (e.g., absence of deadlock, meeting deadline, not overflowing a queue, etc.) and that in order for a property to hold a number of requirements may have to be met, which we normally neither express nor assert formally" [1]. For example, "response time of a component should not exceed 2ms" is a requirement, while "response time of component

[1] These definitions have been provided and formulated with the help of Prof. Tullio Vardanega, University of Padova, Italy

A never exceeds 2ms" and "response time of component B is equal to 1ms" are expressions of properties in a system. In [14], 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.

NFRs have several characteristics that make their consideration in software development process a challenging task. While Functional Requirements (FRs) are typically realized and implemented one by one and step by step as part of the software product, NFRs do not usually follow this pattern and the design decisions taken to implement the functionality of the software affect their satisfaction. Similarly, while FRs normally have localized effects, NFRs are basically specifications of global constraints (e.g., performance, security, availability, etc.) to be satisfied by the software [3]. In this regard, NFRs can crosscut different parts of the system (e.g., security). Also, NFRs are interconnected and there are dependencies among them which implies that the satisfaction of one NFR can conflict and impair the satisfaction of other ones. Therefore, trade-off analysis is required to identify such impacts and establish balance among NFRs in a system.

On the other hand, NFRs are usually specified in an abstract and informal way [3, 14] and thus, providing a more formal approach using model-based development which tries to raise the abstraction level of systems can help with the treatment of NFRs during the software development. There are several reasons for incorporating NFRs in the development process and explicitly dealing with them. Among them are the increasing number of applications such as in real-time embedded systems where NFRs play a critical role, and also the strong interaction between functional and non-functional requirements. Moreover, an explicit treatment of NFRs facilitates prediction of quality properties of the final product in a more reliable and reasonable way [14].

The approaches that are suggested for the explicit treatment of NFRs are sometimes categorized into two groups: product-oriented and process-oriented [15]. 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 [15, 14].

5.3 Characteristics of the Solutions

Based on the identified challenges that were discussed previously, in this section we formulate the key characteristics that a model-based solution for representation and trade-off analysis of NFRs should have.

Traceability of design decisions related to an NFR: considering that NFRs usually crosscut different parts of the system, the designer should be able to understand which parts of the system contribute (both positively and negatively) to a specific NFR; for example, an encryption component that is intended to satisfy security requirements. With this information, after trade-off analysis, the designer can identify parts of the systems that should be removed, replaced, improved or kept.

Traceability among NFRs: throughout the whole design process of a system, higher level NFRs are refined and broken down into more concrete ones, particularly in a top-to-bottom approach. For example, a high level and abstract NFR such as security can be refined into access control or encryption requirements at lower levels. Therefore, in order to check the satisfaction of security in the system, it is necessary to keep track of its refinements and lower level requirements that cover different aspects of security along with information on how much each one contributes towards its parent NFR.

Satisfaction level of an NFR: it should be possible to evaluate the total satisfaction level/degree of an NFR in the system. This is necessary to compare current design against system specification and customer requirements as well as different design alternatives. In the end, the goal is that the designer gets an idea to what extent each NFR is satisfied. Judging where this level is acceptable or not is done as the next step after applying the approach we suggest in this paper and probably by checking it with the stakeholders.

Impact of an NFR on other NFRs: as mentioned before, NFRs cannot be considered in isolation in a system without taking into account their impacts on each other. Therefore, it is required to identify and evaluate the effects that a model element and design decision that is used to satisfy one NFR, has on another NFR in the system. For example, performing heavy computations by an encryption component in an embedded system can also mean consuming more battery. Therefore, the side effects of such design decision should be identifiable for the designer.

Priority of an NFR: not all NFRs have the same importance in the system. In order to increase the overall satisfaction of NFRs and also to resolve conflicts among NFRs (reduce the impact of one NFR in favor of another), it is necessary to know the importance of each NFR and compare them. Consid-

ering priority for NFRs is also important in order to capture the preferences of customers/users. Similarly, priorities can also be considered for different features implemented to satisfy an NFR.

Coherent terms for NFRs: one subtle problem with NFRs that is more often noticed in large enterprises is that different (sub)departments may have different interpretations for an NFR term or use different terms to refer to an NFR [16]. Therefore, it is important to provide a coherent and consistent notation for defining NFRs and relating them to design elements.

Coherent measurements of NFRs: to compare different NFRs and perform trade-off analysis among them, evaluation of the satisfaction degree and impacts of NFRs should follow a coherent representation. In other words, the criteria or metrics 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).

5.4 Suggested Profile

Figure 5.1 shows the structure of our profile to include the necessary information mentioned in the previous section in models to enable trade-off analysis of NFRs.

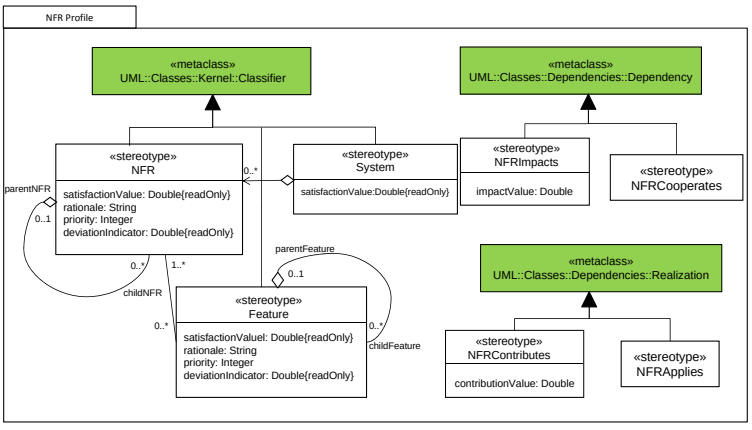

Figure 5.1: Profile for Trade-off analysis of NFRs

`System`: this stereotype is used to annotate the system which is the context of the analysis and to which different NFRs belong. It includes satisfac-

tionValue tagged value to represent quantitatively the total satisfactions of the system's NFRs.

`NFR`: each NFR is identified using this stereotype. A higher level NFR (in terms of abstraction level) can be refined into one or more other NFRs. Therefore, there is an association relationship to itself (reflexive aggregation).

`Feature`: this stereotype represents a feature in the system that contributes to the satisfaction of an NFR and is an equivalent of *Operationalization* concept in NFR framework and Softgoal Interdependency Graph (SIG) [17] (described later in the paper) or *tactics* as used in [4].

`NFRContributes`: this stereotype which is used on relationships between model elements shows that an element (NFR or Feature) contributes directly to the satisfaction of another element. The contributionValue property of this stereotype is used to specify 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`: is used to relate two or more elements that cooperate together to satisfy an NFR. This is similar to the AND relation in NFR framework and SIG.

`NFRApplies`: the relation between NFR related model elements and functional ones can be modeled using NFRApplies stereotype (e.g. an NFR that applies to a component).

`Rationale`: this property and tagged value in NFR and Feature stereotypes can be used to include the description and rationale for an NFR, its refinements into other NFRs or Features implementing it.

`Priority`: this property in NFR and Feature stereotypes is used to capture customer preferences and priorities in terms of the importance of NFRs and Features. This helps to identify less important parts in case modification and removal of some features or NFRs is necessary.

`DeviationIndicator`: is a read-only property which is calculated and set automatically. DeviationIndicator takes into account the priority and satisfaction value, and provides a number which indicates to the designer the importance and magnitude of how much the satisfaction of an NFR or Feature has deviated/violated. While the satisfaction value does not reflect user preferences and priorities, the deviation indicator value shows which parts of the system deviated more from the specification (i.e., being fully satisfied) and

need to be modified to achieve a better satisfaction level.

There are also several rules on the elements and their relationships described above:

- The allowed range of values to set is between -1 and 1 (except for priority). For example a negative value on the NFRImpacts relationship shows the negative impact of the source element on the target element.
- The satisfaction value for each leaf node is considered to be 1.
- Allowed values for priority are: 1 (very low), 2 (low), 3 (medium), 4 (high), 5 (very high).
- The sum of contribution values of the links connecting children nodes (refinement/lower level elements) to their parent should be less or equal to 1 (maximum is 1).
- Contribution of a child node to its parent is calculated as the satisfactionValue of the child node multiplied by its contributionValue or impactValue.
- The satisfactionValue of a node is, therefore, calculated as the sum of the contributions from all of its children nodes plus the sum of the impacts of all other nodes, divided by the total number of NFRImpacts relationships to it plus 1. In other words, 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), and i_j is the impact value of another node on this parent node, the satisfaction value of the parent node is calculated as:

$$\frac{\sum s_k * l_k + \sum^n i_j}{n+1} \tag{5.1}$$

  This calculation is performed starting from leaf nodes (considering that the satisfaction of leaf nodes is 1) and is calculated recursively upwards toward the top element which is the system.

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

$$\begin{aligned} &DeviationIndicator \\ &Priority - Priority * SatsifactionValue \end{aligned} = \tag{5.2}$$

Based on this calculation and considering that the SatisfactionValue is always between -1 and 1 and priority is an integer value between 1 and 5, the value of DeviationIndicator will be in the range of $[0, 10]$. 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.

5.5 Implementation and Usage Example

The profile concepts described in the previous section were implemented in MDT Papyrus [18]. In Figure 5.2 an example usage of the profile is shown on parts of a mobile phone system. Two NFRs are defined. One on the quality of the taken camera picture and the other one on the battery life. There are two features that contribute to the quality of taken photos: usage of a flash and type of the lens. Also two features are considered for `Battery Life`: adjustment of screen's brightness level and auto standby feature. The contribution of each feature to its parent NFR is annotated using NFRContributes stereotype and its contributionValue attribute. In this system, the use of flash consumes lots of battery. The impact of the `Flash` feature on `Battery Life` is therefore specified using NFRImpacts stereotype and its impactValue attribute which is -0.8. Preferences of the customer in terms of the relative importance of NFRs and Features are captured through the priority property of each model element. For example, `Battery Life` has priority level 5 which means it is more important than the `Camera Picture Quality` which has priority level 4. Satisfaction and DeviationIndicator values are initially 0 as no calculation has yet been done on the model.

One point here is that, although these values are assigned subjectively by the system designer, there are methods such as sensitivity analysis [4] that help to increase the confidence in the chosen decision. Now what we need to understand is the impact that having the `Flash` feature has on the system.

By having the necessary information in the model, it is now possible to perform analysis on the model to determine impacts of each design decision on the system and evaluate their dependencies. To traverse the model and perform calculations, we have developed a model-to-model (M2M) transformation using QVT Operational language (QVT-O)[19] in Eclipse [20]. It reads as input a UML model annotated with our profile, traverses the nodes and calculates satisfaction values and writes the results back in the same model. In other words, we use an in-place transformation (i.e. input and output models are the same).

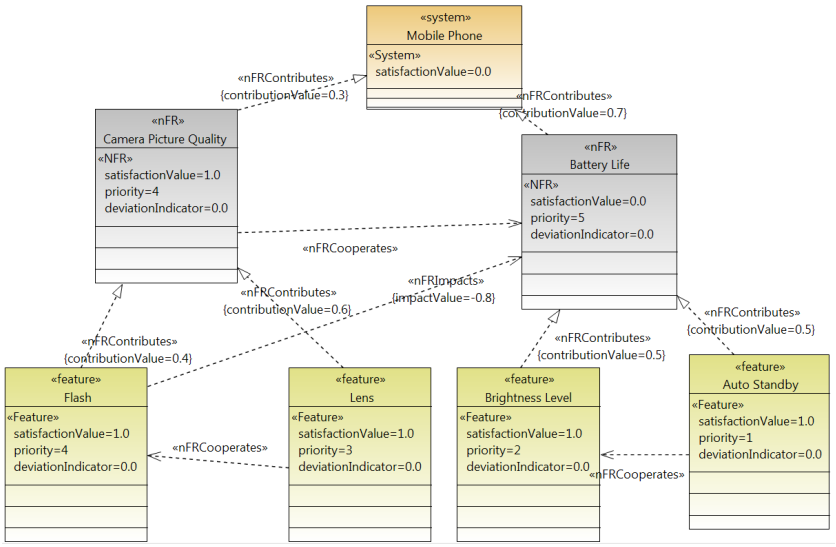

Figure 5.2: Example usage of NFR profile for a Mobile Phone (before calculations)

To calculate the satisfaction values, a recursive algorithm is used in the model transformation based on the rules mentioned in the previous section. For each node, all the incoming links that have NFRImpacts or NFRContributes stereotypes are retrieved. If a node does not have any such links (meaning that it is a leaf node), its satisfaction value is set to 1. Otherwise, the source of the link, which is another node, is retrieved and the algorithm continues by calculating the satisfaction value of the source node; hence the recursion.

The algorithm basically applies Formula 5.1 which is mentioned in the previous section. For example, in Figure 5.2, the satisfaction values for `Auto Standby` and `Brightness Level` features are set to 1, since they are leaf nodes. The satisfaction value of `Battery Life` is then calculated as the satisfaction value of `Brightness Level` (1) multiplied by the value on the link that connects it to `Battery Life` (0.5), plus the same multiplications for `Auto Standby` (1*0.5) and `Flash` (1*-0.8) which results in 0.20. The discrepancy observed between this calculated value (0.20) and the one in Figure 5.3 (0.199...) which is calculated automatically through the in-place model transformation on the model is due to the OCL implementation of real

numbers that QVT-O uses.

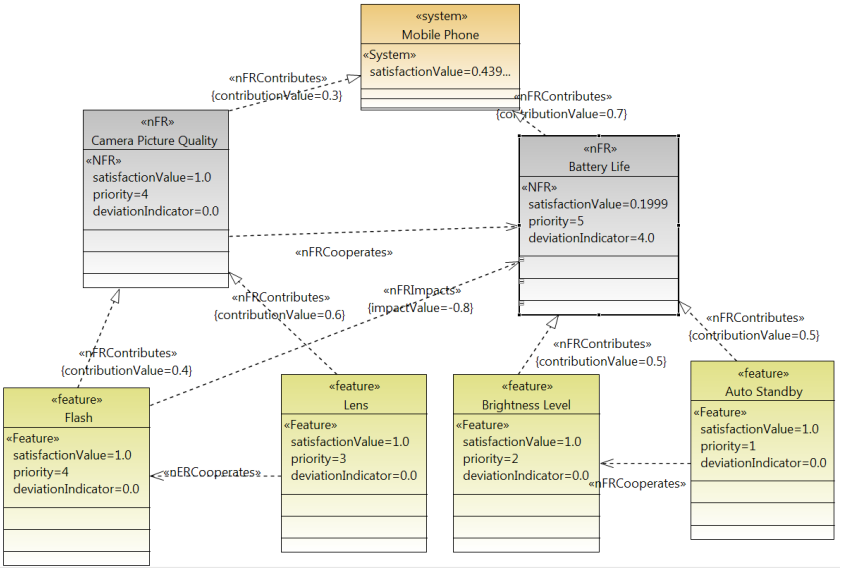

Figure 5.3: Analyzed model of the system

By performing the transformation on the whole model, the satisfaction values are calculated for each node and propagated upwards toward the system element as shown in Figure 5.3:

$$(1 * 0.4 + 1 * 0.6) * 0.3 + ((1 * 0.5 + 1 * 0.5) - 0.8) * 0.7 = 0.44$$

Therefore, the total satisfaction value in this case will be 0.44. Trying the procedure again on the same model but without having the `Flash` feature results in:

$$(1 * 0.6) * 0.3 + (1 * 0.5 + 1 * 0.5) * 0.7 = 0.88$$

Similarly, as soon as the satisfaction value for a feature or NFR is calculated, the deviationIndicator property value is also calculated using Formula 5.2 and set in the model. For the leaf nodes, since their satisfaction values are always 1, their deviationIndicator value will always result in 0. For the `Camera Picture Quality` NFR, the deviationIndicator value is calculated as:

$$4 - 4 * 1.0 = 0$$

which means that there has been no deviation from the desired design. However, for the `Battery Life` NFR, this value will be:

$$5 - 5 * 0.2 = 5 - 1 = 4$$

This value shows that there exists some deviation which in this case is due to the side effects of having the `Flash` feature on `Battery Life`. Based on the specification of the actual system that is being developed, this could, for instance, imply that the type of flash is not good enough in terms of energy consumption for this system and needs re-consideration.

Since the calculation of the deviation indicator is based on the priority of each NFR or Feature, in a larger model where deviations in several parts of the system are observed, the user can understand on which parts of the model, modifications should be done and in which order. In other words, parts with higher deviation value indicate more critical problems and deserve more attention.

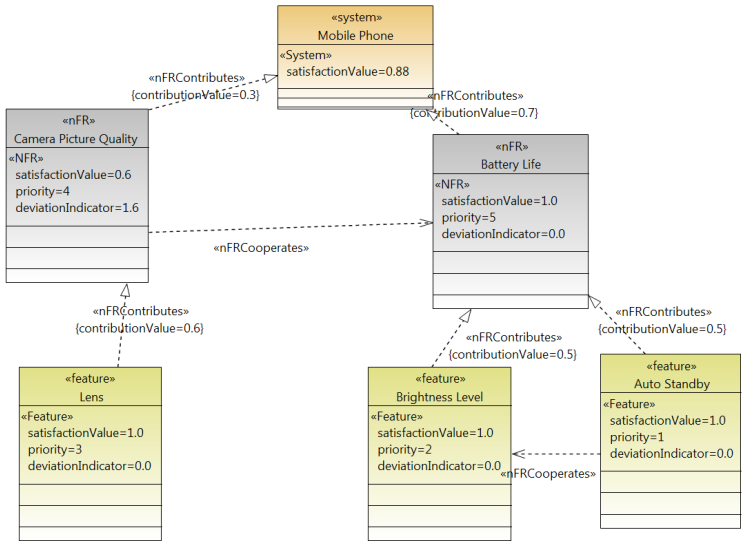

Figure 5.4: Analyzed model of the system without the Flash feature

Figure 5.4 shows performing the analysis on the same model but without the Flash feature. This model could, for example, represent another family of mobile phones or another usage scenario. It can be seen that the total satisfaction level of the system is higher in this case: 0.88 versus 0.44 in the previous case. The removal of the `Flash` feature, however, also causes some deviation (1.6) for the `Camera Picture Quality` NFR.

Based on these calculations, our suggested approach enables a more accurate comparison of design alternatives considering the interdependencies and trade-offs among different NFRs of the systems helping designers to make better decisions. Also the introduced modeling concepts and calculations above provide for several interesting features. One feature is the possibility to optimize the total satisfaction of the system considering different design alternatives. For example, in the mobile phone system described, if the designer needs to select among several possible solutions that contribute to better image quality, he/she can find the ones that lead to the highest satisfaction value of the system. Another possibility is to have run-time adaptability or re-configuration based on different quality of service levels. For example, if the battery level goes low beyond a certain limit, the system can go into a power-saving mode using features that incur minimum impact on battery consumption or replacing active components with back-up/standby ones which may provide lower quality/fewer services but consume less battery (e.g., in design diversity techniques [21]). Without the analysis introduced here, such a decision may not only be hard, but also will be blind in the sense that the side effects of a feature/component replacement on other aspects of the system will be unknown.

5.6 Related Work

There are versatile research works that try to target different issues regarding NFRs. [22] focuses on the problem of informal and separated documentation of design decisions and NFRs. To alleviate the problem, it introduces two profiles to model design decisions and generic NFRs to treat them as first-class entities in software architectures and maintain the traceability of design decisions and architectural elements. NFR framework proposed in [17] is one of the fundamental works in the area of NFRs which is a process-oriented and goal-oriented approach. It uses Softgoal Interdependancy Graph (SIG) to represent NFRs, their refinements and entities that NFRs are applied to (termed as *Operationalization*), and the interdependencies among them to include their impacts and relations. The dependencies and contributions of NFRs are spec-

ified 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. It provides notations to mark critical NFRs in the graph and also an evaluation procedure to determine satisfaction and conflicts of NFRs. The approach suggested in NFR framework is basically a qualitative approach. Moreover, the criticality concept in NFR framework seems 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[4]. [23] offers a UML profile for modeling SIG and concepts of NFR framework to represent NFRs as UML elements in order to integrate them with functional parts of the system (that are modeled in UML). T. Marew et al. [4] introduces Q-SIG which is a quantified version of SIG that enables quantitative evaluation of impacts and trades-offs among NFRs. In this paper, we introduced modeling concepts that enable designers to apply the Q-SIG approach in the form of UML models, and provided a tooling solution for evaluation and trade-off analysis of NFRs on these models using this approach. ProcessNFL that is introduced in [3] is a textual language for describing non-functional properties during the software development. It offers templates for describing three abstractions that capture different aspects of non-functional properties: NF-Attributes, NF-Properties and NF-Actions. Moreover, the language enables to express the relationships between these abstractions to include dependencies and impacts of non-functional properties in the system. The language is an effort for explicit treatment of non-functional properties during software development.

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 [1, 24] 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. [2] makes use of Requirements Hierarchy Approach (RHA) as a quantifable method to measure and manipulate the effects that NFRs have on a system. It does so by capturing the effects of functional requirements. In [25], an approach for quantifying NFRs based on the characteristics of and information from execution domain, application domain and component architectures is suggested.

SysML [26] which is a UML profile for system engineering also includes a package for generic modeling of requirements (both NFRs and FRs) and the relationships among them. These relationships mainly capture the traceabil-

ity and hierarchy of requirements (e.g., refinement). While SysML does not specifically focus on NFRs and analysis of them, our approach and SysML can be used together to complement each other.

5.7 Conclusion and Future work

In this paper, a UML profile for modeling NFRs and their dependencies was introduced to enable performing trade-off analysis among them. It was shown how it can help to compare different design models, determine which ones achieve a higher satisfaction of the NFRs, and identify parts of the model which might be good candidates for modification to reach a higher satisfaction level in the system.

One point to note about the proposed approach is the issue of scalability and evaluating how it can be applicable for very complex and large systems. While this is basically a general concern in model-driven engineering, there are some solutions for management of large models which can be considered such as using a multi-view approach and providing better degree of separation of concerns by defining different views over the model of a system [27]. Also in this work, we assumed that the designer can provide values (though subjective) for contribution and impact relationships among NFRs and functional features. As mentioned, there are some techniques that help system designers in providing such quantitative information. However, this may also imply that our suggested approach can be especially more applicable for component-based systems where systems are built out of already existing components and thus more knowledge about their characteristics and behaviors are available (e.g., memory usage, execution time). Moreover, our approach can also be useful in the analysis of software architecture evolution, when new requirements or features are introduced into a system or existing ones are modified [28].

As the continuation of this work, we plan to develop an analysis tool as an Eclipse plug-in that can read as input, models annotated with our NFR trade-off profile and provide total satisfaction values for different NFRs, identify parts contributing negatively to an NFR, and perform calculations for overall optimization of NFRs in the system considering different design alternatives and scenarios. With the help of the tool, when some parts of the system need to be changed and updated, the user can identify the side effects of such changes on other parts and the system as a whole in terms of NFRs, at model level and before implementing the intended changes. The defined profile depicted in this paper is the first step toward enabling such features.

Using the introduced modeling concepts here along with a back-annotation mechanism in model-based development of embedded systems will also be an interesting topic to further investigate and work on. Having such a mechanism, it would be possible to monitor the system and provide feedbacks to the design model about possible violations in the system (or any of its subsystems) in terms of satisfaction levels of their NFRs. Also, the usage of the suggested approach for run-time adaptability and re-configuration of systems is another direction for further investigation.

5.8 Acknowledgements

This work has been partially supported by the CHESS European Project (ARTEMIS-JU100022) [29] and Xdin AB [30].

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] Bran Selic. The pragmatics of model-driven development. *IEEE Software Journal*, 20:19–25, September 2003.

[9] 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.

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

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

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

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

[21] 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.

[22] 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.

[23] 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.

[24] M. Kassab, M. Daneva, and O. Ormandjieva. Early quantitative assessment of non-functional requirements, June 2007.

[25] 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.

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

[27] Marco Panunzio and Tullio Vardanega. A metamodel-driven process featuring advanced model-based timing analysis. In *Reliable Software Technologies Ada Europe 2007*, volume 4498 of *Lecture Notes in Computer Science*, pages 128–141. Springer Berlin / Heidelberg, 2007.

[28] Hongyu Pei-Breivold, Ivica Crnkovic, and Magnus Larsson. A systematic review of software architecture evolution research. *Journal of Information and Software Technology*, July 2011.

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

[30] Xdin AB. `http://xdin.com/`, Accessed: April 2012.

Chapter 6

Paper C: Modeling Security Aspects in Distributed Real-Time Component-Based Embedded Systems

Mehrdad Saadatmand and Thomas Leveque
The Ninth International Conference on Information Technology : New Generations, Las Vegas, Nevada, USA, April, 2012.

Abstract

Model Driven Engineering (MDE) and Component Based Software Development (CBSD) are promising approaches to deal with the increasing complexity of Distributed Real-Time Critical Embedded Systems. On one hand, the functionality complexity of embedded systems is rapidly growing. On the other hand, Extra-Functional Properties (EFP) must be taken into account and resource consumption must be optimized due to limited resources. However, EFP are not independent and impact each other. This paper introduces concepts and mechanisms that allow to model security specifications and derive automatically the corresponding security implementations by transforming the original component model into a secured one taking into account sensitive data flow in the system. The resulted architecture ensures security requirements by construction and is expressed in the original meta model; therefore, it enables using the same timing analysis and synthesis as with the original component model.

6.1 Introduction

Design of real-time embedded systems is a challenging task. This is mainly due to the complexity of these systems that originate from different range of extra-functional properties that they need to satisfy, while taking into account their limitations of resources. This gets even more complex when we realize that the extra-functional properties in these systems are tightly inter-connected and cannot be considered in isolation [1]. Due to the nature of real-time embedded systems (e.g. usage of sensors and actuators and interacting with the environment), timing properties in these systems are of utmost importance. However, implications of other properties and aspects, such as security, on timing properties should also be taken into account to ensure a correct design.

Security aspects in embedded systems are gaining more and more attention especially when they are distributed. Introducing security in the design of an embedded system has impacts on other properties such as timing, performance, memory usage, and energy consumption. On the other hand, the usage and hostile operational environment of embedded systems makes them also exposed to specific attacks that might not be that relevant for other systems [2]. For example, smart cards and wireless sensor networks which are physically exposed, are quite tamper-prone compared to a bank database server which is protected from access and isolated physically in a separate room. There is a need to include security properties in the design of a distributed real-time embedded system while still enabling prediction of timing properties.

Model-Driven Engineering (MDE) and Component-Based Development (CBD) are two promising approaches that can be used orthogonally to alleviate the design complexity of real-time embedded systems. Component-Based Development enables reuse of already existing software units (components) by developing a system as an assembly of components instead of building the system from scratch. Model-Driven Engineering, on the other hand, helps to raise the abstraction level and perform analysis at earlier phases of development. This enables the identification of problems in the system design before reaching the implementation phase [3, 1, 4].

Using benefits of these two approaches, we propose to specify security needs as annotations on the ProCom component model [5] and derive the equivalent component model which implements the security specification. Using our approach, the designer specifies sensitive data flows with required security properties and selects an implementation strategy to fulfill these requirements. Based on this information, a component model conforming to the original meta-model is generated which satisfies the security specification and the

Worst Case Execution Time (WCET) of the resulted components is computed. Therefore, same tools and analyses such as timing analysis and synthesis are applicable for the original component model and the derived one. As a result, timing implications of specified security properties are predictable. This approach facilitates system designers in bringing security aspects into the design model.

The remainder of the paper is structured as follows. In Section 6.2, motivation of this work and security challenges in the design of embedded systems are discussed. Section 6.3, introduces Automatic Payment System as an example of distributed real-time embedded systems with security requirements. The suggested approach is described in detail in Section 6.4. Implementation and analysis results are explained in Section 6.5. Section 6.6 discusses related work. Finally, we conclude the paper and describe future work and directions of this work in Section 6.7.

6.2 Motivations

Security is an aspect that is often neglected in the design of embedded systems. However, the use of embedded systems for critical applications such as 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 [6] for 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 such as physical and side channel attacks [7] that might be less relevant for other types of systems. Increase in use and development of distributed 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 distributed networked environment.

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 properties and requirements 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 empha-

sizes 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. Considering the characteristics of embedded systems, major impacts of security features in these systems are on performance, power consumption, flexibility and maintainability, and cost [7]. Therefore in the design of embedded systems, implications of introducing security decisions should be taken into account and analyzed.

Today, security is mostly taken into account at code level which requires detailed knowledge about security mechanisms while there is a need to raise the abstraction level to deal with the complexity of security implementation.

6.3 Automatic Payment System

An example of a distributed embedded system with real-time and security requirements is the Automatic Payment System that is being designed for toll roads as depicted in Figure 6.1. In this system, the purpose is to reduce the waiting time and thus traffic that is caused by that at tolling stations.

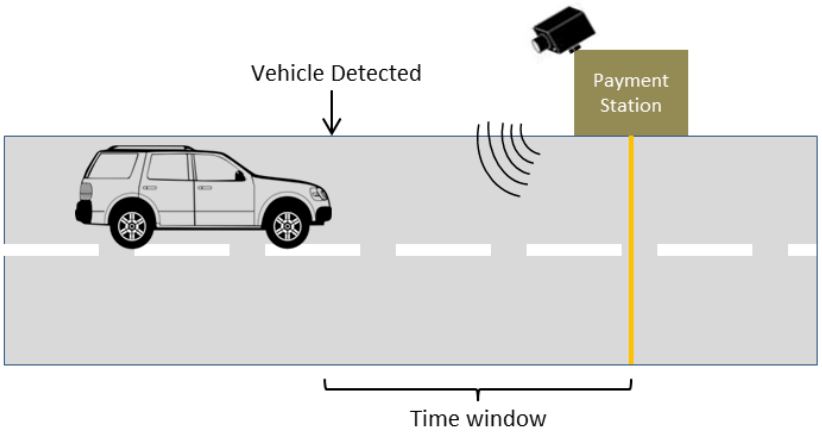

Figure 6.1: Automatic Payment System for Toll Roads.

For each tolling station, a camera is used that detects a vehicle as it 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. The vehicle shows the amount to pay to the driver 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 (through a wired Internet connection at the station). The driver is then notified about the success of the transaction and receives an *OK* message to go. Different objects in this system and the interactions between them are shown in Figure 6.2.

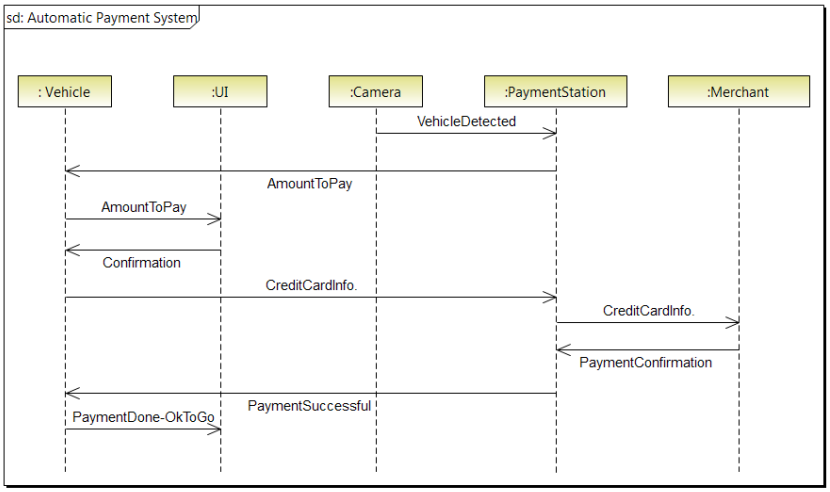

Figure 6.2: Automatic Payment System for Toll Roads.

The system should perform all these operations in a certain time limit in order to allow a smooth traffic flow. Such time constraints can be calculated considering the specifications of camera and required time for detection, 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:

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.

6.4 Approach

6.4.1 General Approach

Our approach aims to introduce security concerns in the design of embedded systems. The main idea is to identify the different data entities which need to be confidential and/or that the sender must be authenticated. From the specification of security needs at data level and physical platform level, a transformation is applied on the component model in order to add security implementation. The resulted system ensures the security specification.

Figure 6.3 shows the overall process of the approach. The system is described in several models based on ProCom component model [5]. While the approach is not ProCom specific, it relies on the main assumption that a component model transformation to introduce security implementation exists which has been proved in ProCom but remains as future work for other component models. In the remaining, we will refer to component model to represent the architecture model. Security needs are specified as annotations on top of the data model and the physical platform model. A benefit of the ProCom component model is his ability thanks to its attribute framework to extend any element with additional attribute. We use this mechanism to specify our annotations. Having this information in the model, the following steps are then performed:

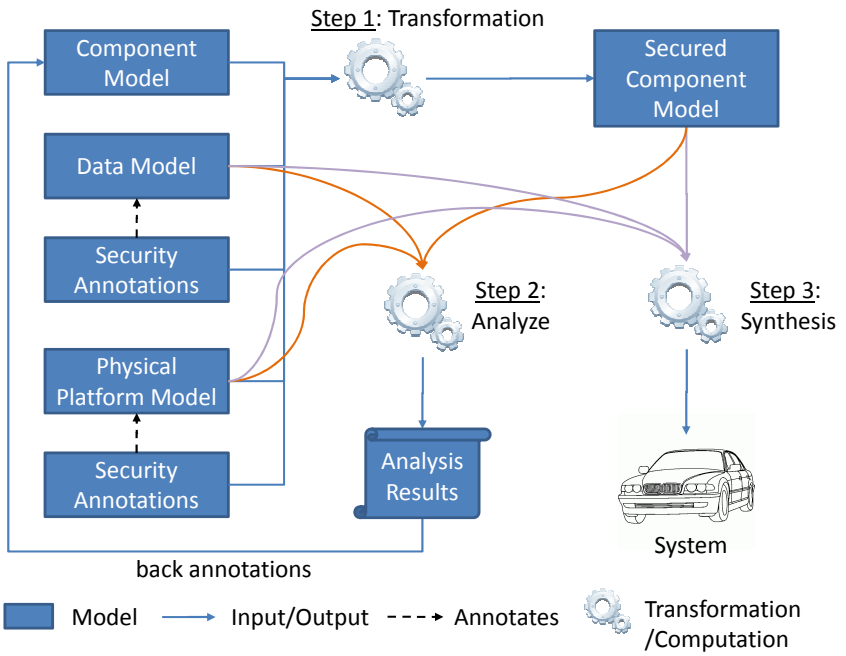

Figure 6.3: Approach Process.

1. The component model which specifies the functional and extra-functional part of the system is transformed in a functional equivalent model with added security implementations;
2. Analysis can be performed on the secured component model whose result is back annotated (for example with timing properties) to the original model; and
3. Finally, the system is synthesized.

The considered process is iterative and allows to refine security specification after evaluating resulted system properties such as timing properties.

6.4.2 ProCom Component Model

While the approach principles seem to be 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 6.4 shows ProCom design of the Automatic Payment System example.

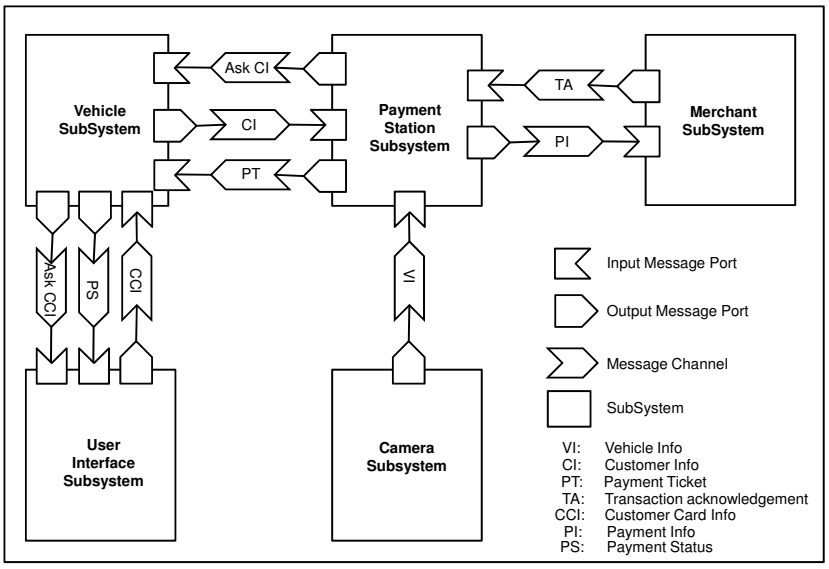

Figure 6.4: Component Model of the System using ProCom.

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 6.5 shows the implementation of the subsystem E as an assembly of two component C1 and C2. Data input- and output ports are 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.

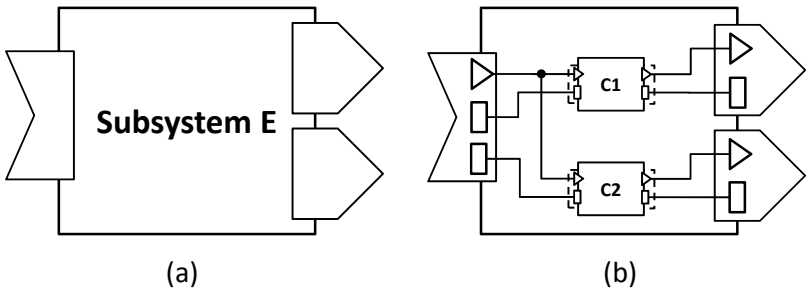

Figure 6.5: ProCom SubSystem Implementation.

6.4.3 Data Model

As components are usually intended to be reused, data should also be reused. To this end, we propose to extend the approach described in [8]. Every data entity is stored in a shared repository. Its description 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. Table 6.1 and Table 6.2 show data entities of our example. As described in the last section, subsystems communi-

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

Table 6.1: Primitive Data Entities.

cate 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 6.3 presents 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

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

Table 6.2: Composite Data Entities.

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

Table 6.3: Mapping between Data Entities and Message Channels.

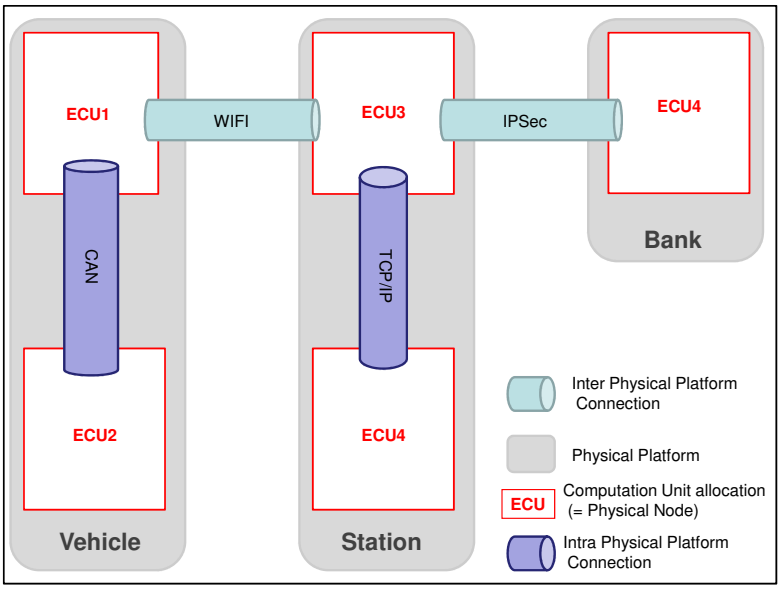

Figure 6.6: Physical Platform Model of the System.

ports of message ports and data entities is based on naming convention which enables to distinguish between 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 used in the related design.

6.4.4 Physical Platform And Deployment Modeling

The physical entities and their connections are described in a separate model called Physical Platform Model (see Figure 6.6). This model defines the different Electronic Computation Unit (ECU) called Physical Node including their configuration such as processor type and frequency, the connections between physical nodes and the physical platforms which represents a set of ECU fixed together.

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 6.7 defines the physical platform and related mapping of Automatic Payment System model. To simplify the example, we assume one to one mapping between virtual node and physical node.

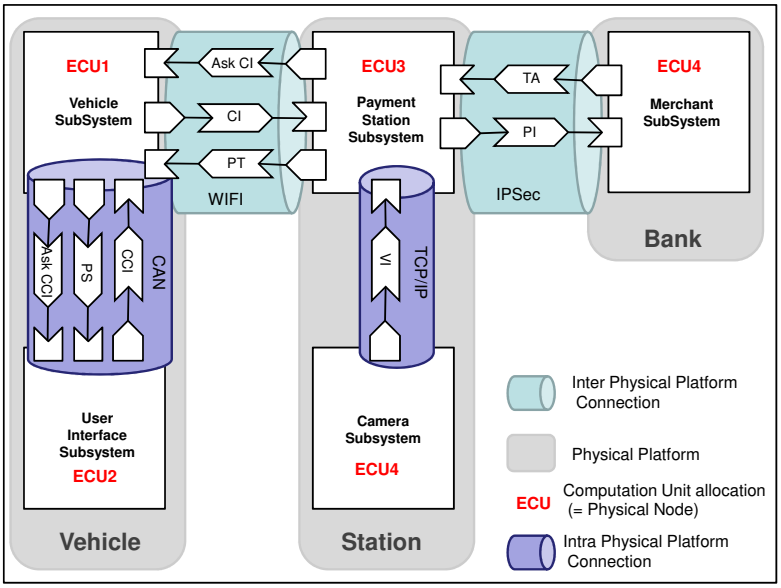

Figure 6.7: Deployment Model of the System depicting allocation to Physical Platforms.

6.4.5 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 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 can not be read by any external person of the system; and
- **Authentication** which ensures that the considered information comes from the expected sender.

Table 6.4 shows security annotations associated to data entities for our example. In addition to security properties on the data model, we define the

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

Table 6.4: Data Entity Security Properties.

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:

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

Using these properties, the responsible of 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. 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 6.5 shows the security properties of Automatic Payment System physical platforms.

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

Table 6.5: Security Properties of Physical Entities.

6.4.6 Cost of Security Implementations

To satisfy the identified security properties in the system, different security mechanisms, namely encryption/decryption algorithms in this paper, can be used. As stated before, using a security mechanism in the system has its own costs in terms of timing and performance, power consumption and so on. Therefore, choosing an appropriate security mechanism is critical in order to ensure the satisfaction of timing requirements of the system while fulfilling the security requirements. For this purpose, and to take into account the timing costs of different security mechanisms, we rely on the results of studies such as [9] 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 6.6. We assume that execution time can be computed knowing targeting platform, algorithm, key size and data size. We suggest to provide a timing estimation toolkit which enable to provide estimate based on measurements. It can be observed that depending on the targeted platform, some algorithms may not be supported.

Table 6.6 shows estimates based on results presented in [9]. These results are related to a specific platform. We do not aim to explain how to get such table. However we assume that it is possible to get such estimates.

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)

Table 6.6: Execution times and strength ranking of different security algorithms for a specific platform

6.4.7 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 between all possible security implementations:

- The **StrongestSecurity** strategy selects the strongest security implementation available on the platforms;
- The **StrongestSecurityAndLimitImplemNb** strategy selects the strongest security implementation available on the platforms which ensures that we use as few as possible different security implementations since each message channel can use a different encryption algorithm;
- 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 which ensures 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. The StrongestSecuritySchedulable strategy is hard to implement and will be part of our future

works. However, it is the most interesting one. More complex security implementation strategies can be considered but are not covered by this paper. In particular, our future works will try to define required security strength associated to data and message channels.

6.4.8 Transformation

The transformation is performed in four steps:

1. First, we identify the part of message which needs to be confidential or authenticated and on which communication channels;

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

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

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

It relies on the following assumptions:

- The confidentiality is ensured using asymmetric keys; and

- The authentication is ensured using electronic certificates.

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 6.8. We assume that only data D1 need to be confidential. The pseudo algorithm of the transformation is described in Listing 6.1.

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 digest used for authentication. The decryption component (DeD1) ensures that all message data will be available at the same time thought 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.

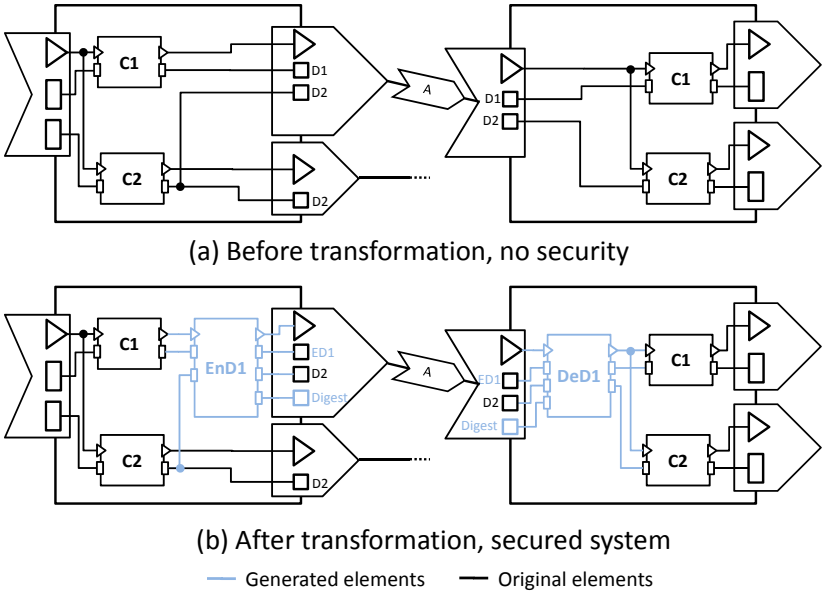

Figure 6.8: Transformation.

Listing 6.1: 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
}
```

6.5 Implementation

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[10] which allows to introduce additional attribute to any model element in ProCom. The model transformation has been implemented using a QVTo[11] transformation plugged at the end of the process described in [12]. These experiments aims to show the benefits at 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 analysis of estimate

accuracy is out of the scope of this paper.

This paragraph presents some ideas to generate code of security components. In order to keep the approach generic, we intend to let certificate specification and other encryption algorithm specific parameters to be filled in the generated code. One generator is associated for each algorithm. The suitability for timing analysis of the generated component code need 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 to react to authentication errors. This problem could be partially solved by letting opportunity to the developer to add code to manage authentication errors in the generated code which leads to define what must be the output data in this specific case.

6.6 Related Work

With the growing complexity of real-time embedded systems, management of run-time data in these systems has become more important than before and has gained more attention. It has become extremely hard for one person to keep track of all data that is passing through different parts of the systems. Currently, most design methods based on component models focus on functional structuring of the system without considering data flow meaning and semantics[8]. [8] introduces a data-centric approach which models data and proposes to use real time database for data management at runtime in real-time embedded systems. It introduces an architectural view for data entities to complement component-based views. Unfortunately, it does not address extra-functional properties such as security. Our work follows a similar approach to model data entities as a basis to define security specification.

For modeling security features in general, several solutions have been offered such as UMLsec [13] that is a UML profile for the specification of security relevant information in UML diagrams. It is one of the major works in this area and also comes with a tool suite which enables evaluation of security aspects and their violations. There are other similar approaches that have narrower focus like SecureUML [14] that enables modeling of role-based access controls. However, modeling security requirements in isolation (from other aspects of the system) is not enough and become problematic to predict impact on other extra-functional properties especially for real-time embedded. There are works such as [2] and [7] that discuss security issues unique to embedded systems. In [15] we have proposed and discussed benefits of extending MARTE

[16] modeling language with security annotations to cover the modeling needs of embedded systems. This work focused on providing UML stereotypes to specify confidentiality property of message communication and related timing estimates. Contrasting, our work models confidentiality and authentication properties at higher abstraction level enabling the designer to focus on sensitive data without thinking about the security implementation which will be automatically generated.

In [17], a method is introduced to specify security requirements on UML models and check their satisfaction by relating model-level requirements to code level implementations. UMLsec is used to include security requirements at model level, and JML annotation language is used to relate code blocks back to the security requirements specification, therefore enables evaluation of security requirement assurance. While this work is also an MDE based approach for defining security requirements, it does not provide timing impacts of security implementation and does not automatically derive security implementation.

The work described in [18] considers security issues in model-based development of service-oriented applications. It proposes a process meta-model for modeling service orchestration as a workflow and another one for modeling high-level security properties. In addition, a more detailed model is used to specify low level security specification and the tool generates the security implementation from these models. Our approach is similar to this work in the sense that it aims to define security properties at high level of abstraction. While they focus on automatic service selection and other security properties such as integrity, our objective is to compute timing impact of security implementation and target another application domain. In addition, we identify sensitive part of messages which need to be secured.

The work in [19], highlights the need to identify sensitive data and introduces an extension to include security concerns as a separate model view for web-services based on Web-Services Business Process Execution Language (WS-BPEL). However, it does not take into account consequences of security design decisions on timing.

Studies [9, 20] are two examples of works that measure the costs of security algorithms. [9] provides performance comparisons of several encryption algorithms, and [20] compares energy consumptions of encryption algorithms on two sensor motes used for building wireless sensor networks. While the focus in these works is not on system design, MDE, and CBD methods, in our work they serve as hints and examples on how to get costs of security algorithms.

6.7 Conclusion

Modeling of data in embedded systems is mainly done by specifying the type of data, while the semantics of transferred data is needed for security concerns such as identification of sensitive data. Security, as a non-functional requirement, spans different parts of a system and needs to be modeled at all levels (component architecture, data model, physical platform...). In this paper, we presented an approach which allows to define security specification of embedded systems at high level of abstraction and to produce automatically the security implementation.

Our contributions are

- To propose to model data semantic on top of ProCom architecture model;
- To propose a way to model security properties and needs as annotations on the different models;
- To provide a transformation of component model which ensures by construction security specification; and
- To enable timing analysis on secured component model by computing timing estimates of security components.

All these features help system designers to focus more on system architecture and timing properties which are critical in real-time embedded systems, and at the same time, consider and apply security mechanisms in the design models, without the need to have deep knowledge about how to implement different security mechanisms. This also contributes to the aim and trend of bringing security considerations in higher levels of abstraction.

As future works, we aim to provide analysis to compute security system properties such as robustness of the system to a specific attack, to provide an algorithm for the StrongestSecuritySchedulable, evaluation on an industrial use case and to consider other extra functional properties such as power consumption of security implementations and to provide trade-off analysis between security properties and other extra-functional properties.

Bibliography

[1] 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.

[2] 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, pages 7:1–7:6, New York, NY, USA, 2010. ACM.

[3] Martin Törngren, DeJiu Chen, and Ivica 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*. IEEE, August 2005.

[4] Bran Selic. The pragmatics of model-driven development. *IEEE Software*, 20:19–25, September 2003.

[5] 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.

[6] Eirik Albrechtsen. Security vs safety. NTNU - Norwegian University of Science and Technology `http://www.iot.ntnu.no/users/albrecht/`, Accessed: May 2011.

[7] 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, New York, NY, USA, 2004. ACM. Moderator-Ravi, Srivaths.

[8] 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.

[9] A. Nadeem and M.Y. Javed. A performance comparison of data encryption algorithms. In *Information and Communication Technologies, 2005. ICICT 2005. First International Conference on*, pages 84 – 89, aug. 2005.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] MARTE specification version 1.0 (formal/2009-11-02). `http://www.omgmarte.org`.

[17] 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.

[18] Stéphanie Chollet, Philippe Lalanda, and Gabriel Pedraza. Secure Integration of Service-Oriented Application. September 2009.

[19] 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.

[20] 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.

Chapter 7

Paper D: Design of Adaptive Security Mechanisms for Real-Time Embedded Systems

Mehrdad Saadatmand, Antonio Cicchetti and Mikael Sjödin
The Fourth International Symposium on Engineering Secure Software and Systems (ESSoS), Eindhoven, The Netherlands, February, 2012.

Abstract

Introducing security features in a system is not free and brings along its costs and impacts. Considering this fact is essential in the design of real-time embedded systems which have limited resources. To ensure correct design of these systems, it is important to also take into account impacts of security features on other non-functional requirements, such as performance and energy consumption. Therefore, it is necessary to perform trade-off analysis among non-functional requirements to establish balance among them. In this paper, we target the timing requirements of real-time embedded systems, and introduce an approach for choosing appropriate encryption algorithms at runtime, to achieve satisfaction of timing requirements in an adaptive way, by monitoring and keeping a log of their behaviors. The approach enables the system to adopt a less or more time consuming (but presumably stronger) encryption algorithm, based on the feedback on previous executions of encryption processes. This is particularly important for systems with high degree of complexity which are hard to analyze statistically.

7.1 Introduction

Security is gaining more and more attention in the design of embedded systems. Embedded systems are nowadays everywhere. They are used in controlling systems of power plants, vehicular systems and medical devices, as well as, mobile phones and music players. The operational environment, physical accessibility, and mobility of embedded systems make them prone to certain types of attacks which might be less relevant for ordinary computer systems, such as side channel attacks, time and power analysis to determine security keys and algorithms [1]. Also, the increasing use of embedded devices as parts of networked and interconnected devices makes them prone to new types of security issues [2].

On the other hand, introducing security in embedded systems requires careful considerations, trade-off analysis, and balance with other aspects such as performance, power consumption, and so on. This is mainly due to the resource constraints and limitations that these systems have. For example, choosing an encryption algorithm that performs heavy computations, requires lots of memory, and, as a result, consumes more energy, may impair the correct functionality of the system and violate its specified requirements. This is basically because of the fact that non-functional requirements, such as security, are not independent and cannot be considered in isolation [3]. Therefore, it is important to understand the impacts and consequences of designed security mechanisms on other aspects of the systems.

In real-time embedded systems, where timing requirements are critical, choice of security mechanisms is important in terms of satisfaction of timing requirements. One way to achieve this, is to find a security mechanism that fits and matches the timing requirements of the system (e.g., by performing timing analysis), and then implement it [4]. This method leads to a static design in the sense that a specific security mechanism, which is analyzed, and thus, known to execute within its allowed time budget, is always used in each execution. However, this method may not be practical for systems with high complexity, which are hardly analyzable or systems with unknown timing behaviors of their components. Instead, 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. In this paper, we introduce this approach, and describe its implementation for selecting appropriate encryption algorithms at runtime (in terms of their timing behaviors) in an adaptive way, using OSE real-time operating systems [5]. To this end, the timing behavior of each execution of the encryption procedures is logged, and

used as feedback for selecting a more suitable encryption algorithm in the next execution.

The rest of the paper is structured as follows. Section 7.2 describes the motivation and background of this work. In Section 7.3, we discuss the approach, describe how the adaptation mechanism in the proposed approach works. Implementation and experimental results are also explained in this section. Section 7.4, discusses the context where the proposed approach can be more applicable and suit well. In Section 7.5, related work is discussed, and finally in Section 7.6, conclusions are drawn, and pointers to future directions of this work are provided.

7.2 Background and Motivation

Designing security for real-time embedded systems is a challenging task. This is due to the fact that security features have impacts on other aspects of the system, such as timing, and if these impacts are not identified, analyzed, or managed properly, they can lead to violations of other non-functional requirements, and thus failure of the system. While in small and simple systems timing and schedulability analysis, covering security algorithms, can be done to ensure satisfaction of timing requirements, when it comes to very complex systems, such static and offline analyses might not be practical and feasible [6]. Even in cases where they are feasible, the results of such analyses may be invalidated at run-time, due to several factors such as transient loads, difference between the ideal execution environment (taken into account for analysis) and the actual one, which leads to violation of the assumptions that are used to perform analysis [7].

Telecommunication systems are examples of systems with high complexity that require massive execution capacity and have security requirements. Due to the complexity of these systems, the main focus in their design is to be able to handle massive connection requests and data loads that arrive in a bursty and unpredictable fashion, than just trying to merely perform analysis of all possible conditions in the system [8]. Therefore, it is important that such systems are designed in an adaptive way, so that they can reconfigure themselves at runtime to continue providing their services, although under different Quality-of-Service (QoS) levels. Also, most of the real-time tasks in these systems have soft deadlines. This means it is acceptable, in general, for the functionality of the system, if some tasks complete their jobs within a reasonable margin after their deadlines, and the result of a single deadline miss is not catastrophic.

OSE is a Real-Time Operating System (RTOS) developed by Enea [9], which is used heavily in telecommunication systems, especially by Ericsson, from Radio Network Controllers, and Radio Based Stations (RBS) to mobile devices. In this paper, focusing on the needs of such systems that require run-time adaptation, an approach is suggested to select encryption algorithms at runtime based on how they behave in terms of their time constraints. To implement and evaluate the approach, OSE is used as the base platform. It is an example of a RTOS which is designed from scratch to provide the necessary determinism level required for fault-tolerant real-time embedded systems with high availability, particularly in telecommunication domain.

7.3 Approach

To design a system that can adapt itself and adjust the balance between its time constraints and security level, the approach depicted in Figure 7.1 is suggested.

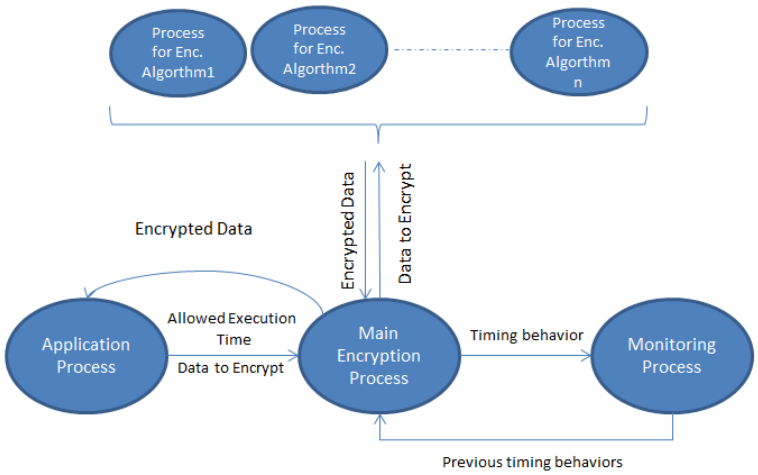

Figure 7.1: Adaptive design of encryption algorithms

When an application needs to encrypt some data, it sends the data along with the allowed execution time for the encryption procedure to the main encryption process. Based on the received time constraint, this process will then

try to encrypt the data with an appropriate encryption algorithm, by invoking the corresponding process implementing that algorithm. This is done by first consulting the log information generated by the monitor process plus a predefined table for ranking of preferred encryption algorithms. An example of such a table is shown in Table 7.1. The table is used to capture the preferences for different encryption algorithms, but it has to be in descending order in terms of execution times. As we will see, this is important for the correct behavior of the system. Comparison of execution times for different encryption algorithms can be obtained from the result of studies such as [10], which has performed performance measurements of different encryption algorithms. This table is, therefore, filled by the user, using the result of such studies. For each algorithm in the table, there is a process that implements it (named in Figure 7.1 as 'process for encryption algorithm 1'...'process for encryption algorithm n').

Rank	Encryption Algorithm
1	AES
2	3DES
3	DES
...	

Table 7.1: Preferred encryption algorithms in descending order in terms of execution times.

Then, when an appropriate encryption algorithm is decided (the following section describes in detail how this is done), the main encryption process passes the data to the process implementing that encryption algorithm to perform the actual encryption of the data. The time taken from the point that the main encryption process sends the plain data it has received from the application process to one of the processes implementing an encryption algorithm, until encrypted data is returned to the main encryption process by it, is sent as part of the log information to the monitoring process. This log information will be used as feedback in the next invocation of the main encryption process.

The assumption that is implicit in this design is 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 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 procedures. This is why the encryption algorithms in Table 7.1 need to be sorted according to their execution times. In this

way, the system first tries to encrypt all data with the algorithm at the top of the table. If it is observed that it cannot fulfill the specified time constraint, the second ranked algorithm (which has shorter execution time) will be chosen for encryption in the next invocation.

On the other hand, if an encryption algorithm completes its job sooner than its specified time limit, the unused portion of its time budget is used to calculate and determine whether it is feasible to go back to the previous higher ranked algorithm for the next encryption job or not. Using this approach, the system tries to adapt itself based on the feedback it receives regarding its timing behavior. Therefore, when a burst of processing loads arrive, the system adapts itself to this higher load, and when the processing load decreases, it can gradually go back to using more time-consuming (and presumably more secure) encryption algorithms. This is, for example, very useful in telecommunication systems, where lots of other services (than the one(s) using encryption) are active and need to be responsive at the same time, where task pre-emption and context switches increase dramatically and interfere with the encryption job.

7.3.1 Log Information and Adaptation Mechanism

The information that is logged by the monitor process has the following format:

Timestamp, Encryption algorithm, Time constraint, Actual execution time

An Example of the generated log information is shown in Table 7.2.

```
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
```

Table 7.2: Sample log information.

The table shows that the system has first performed encryption using AES algorithm, with a time constraint of 50 time units, but the actual execution time has been 90; in other words, it has violated its time constraint. Therefore, in the next execution, 3DES is automatically chosen for encryption (as it is the first lower ranked algorithm under AES in Table 7.1), which has finished its job in 70 time units while having 80 as its time constraint. Hence, no change

in the encryption algorithm is observed and the same algorithm (3DES) is used for the third execution as well. The fifth and sixth rows in Table 7.2 (which represent the fifth and sixth executions) show that the system has chosen to apply a higher ranked encryption algorithm (fourth to fifth: DES->3DES, and fifth to sixth: 3DES->AES).

In the log that is kept in memory from the log information generated above by the monitoring process, if an encryption algorithm is again selected for the next invocation, its latest log record will replace the previous one. In other words, for each two consequent log records for a certain encryption algorithm, only the most recent one is kept. This is done to only keep the information that is necessary, which also leads to having only log information indicating changes of encryption algorithms being stored. Table 7.3, shows what is actually necessary from the generated log information shown in Table 7.2, for making adaptation decisions.

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

Table 7.3: Necessary portion of log information for making adaptation decisions.

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:

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.*

For example, in Table 7.3, applying formula (i) on row 1 (i.e., 90>50) shows that AES has taken more time than it was allowed to; therefore, a lower ranked algorithm (3DES) is used for the next invocation. However, for row 4, and the higher ranked algorithm just before it, which is AES at row 1, formula (ii) holds (i.e., 90-40>abs(90-50)); therefore, at the next invocation (corresponding to row 5), AES algorithm was used again.

7.3.2 Implementation Details

The implementation of the approach is done on OSE real-time operating systems [5]. The OSE edition that is used is OSE Soft Kernel (OSE SFK), which is a simulation of the actual environment to be downloaded to the target embedded hardware, and is possible to run on a host machine (e.g., on Windows or Linux).

The execution unit in OSE corresponding to a real-time task is called process. For all the processes depicted in Figure 7.1, an OSE process is implemented. OSE offers two types of processes: static and dynamic. Static processes are created upon system startup, cannot be terminated, and last as long as the system is up and running. On the other hand, dynamic processes can be created and killed on the fly by another process using OSE APIs. Main application, main encryption process, and monitoring process, are created as static processes. Each encryption algorithm is implemented as dynamic processes, which are, in turn, created by the main encryption process at runtime as needed. This is just a design choice to reduce the number of active processes in the system, as encryption algorithms can also be well created as static processes, in which case, the overhead of creating them for each invocation will be reduced, while increasing the total scheduling overhead and memory usage of the system.

An interesting feature of OSE is offering the concept of *load module*. Load modules are relocatable program units that can be loaded into a running system and dynamically bound to that system. A loadable module can be uploaded, rebuilt, and quickly downloaded while the remainder of the system continues to run [5]. A load module can be considered similar to Windows .exe or .dll files. Once installed, the program (consisting of one or more processes) that they contain, can be created and started. In the case of our system, this means that security designers can add new encryption algorithms to the system dy-

namically or update them on the fly, while the system is active and running. Such a feature is very important in high-availability systems, where updates and upgrades (e.g., patching security issues) should affect the up-time of the system to the least degree possible.

Also, the direct and asynchronous message passing mechanism in OSE, makes this real-time operating system a great choice for use in distributed systems. This further facilitates the scalability of systems that are built on OSE. Data between processes are passed as signals using three basic OSE APIs for signal passing: *send*, *receive*, and *alloc* (to create signals containing data). The Inter-Process Communication protocol (IPC) used in OSE, makes the location of processes transparent to the user; meaning that no matter whether two processes are located on the same board or on different ones, the communication between them is done using the same set of APIs and code. Using this feature, the proposed approach is implemented without the need to use shared memory among processes. The communication between processes is implemented using the three above-mentioned APIs. This brings along the possibility to deploy the processes shown in Figure 7.1 on different processors and boards without affecting the generality of the approach, which is important for highly-distributed real-time embedded systems such as telecommunication systems.

Using the aforementioned features, several signals have been defined for synchronization, and to pass data between processes. For example, signal HISTORY_INFO_REQUEST is defined which is sent from the main encryption process to the monitoring process to request log information. In case of receiving this signal, the monitoring process sends last log record, and the log record for the first encryption algorithm with a higher rank, used before the last record, to the main encryption process using HISTORY_INFO_REPLY signal. Using this signaling mechanism also allows to deploy processes on different nodes if needed. This is possible since the necessary information to make adaptation decisions such as the time it takes to encrypt is passed between processes as part of the signals, making the actual location of processes in different nodes unimportant and transparent for the approach to work. The rank table for encryption algorithms is actually implemented using enumerations in C/C++ in the main encryption process.

7.3.3 Evaluation

To test the behavior of the system, a tool called CPU Killer [11] was used to create desired percentage of CPU loads at desired times. Moreover, Optima, which is a debugging, profiling, and monitoring tool developed by Enea for

OSE, was also used to monitor and observe, in the form of graphs and tables, CPU usage levels at different system ticks from the startup of OSE. The system was run two times: once without having adaptation and the second time using adaptation. At each time, CPU loads of 10%, 50%, 70%, and then back to 50%, and 10% were applied. The results are shown in Figure 7.2.

(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 7.2: Results of running the system with and without adaptation.

The columns for each log record identify: system time (ticks), encryption algorithm, time constraint (ticks), and actual execution time (ticks). As mentioned in the previous section, an enumeration in the form of *"enum algorithms { AES=1, THREEDES=2, DES=3 };"* was used to represent the information of Table 7.1 in the code. The logs are decorated here with additional marks to facilitate explanation and understanding of the results.

In case I, where no adaptation was applied, AES (as number 1 in the sec-

ond column) algorithm is constantly used for encryption. This is because the goal is to provide the maximum level of security, and therefore, the system is designed to prefer and choose the topmost encryption algorithm (whenever possible) from the table, which represents the strongest one. The system was started while applying 10% CPU load, and as can be seen from the figure, encryption is done within its time constraint of 300 and no violation is observed. However, when CPU load is increased to 50%, encryption starts violating its time constraint. Violations are marked with * mark in the figure. In case of the first violation, it can be seen that encryption was completed in 305 ticks while the time constraint is 300 ticks. Violations get worse (with more time margins) at 70% CPU load. It is only after going back to 10% CPU load, that encryption can meet its constraint again.

In the second case (II), where adaptation was used, it can be easily understood at the first sight that the number of violations have decreased. The first violation occurs when the CPU load is set to 50%, however, the system adapts itself to this new load and uses 3DES (as number 2 in the second column), which helps the system to perform within its time constraint again. When CPU load is set to 70%, violations are again observed. Therefore, the system adapts itself by using DES (as number 3 in the second column) instead of 3DES, to stay within the time constraint. Even using DES, the system still fails to meet its constraint, however, within a smaller time margin. In spite of violations, since no lower rank algorithm than DES was defined in this experiment, the system keeps using it as the last possible choice. When the CPU load is reduced back to 50%, using the two formulas described at the beginning of the section, the system realizes that it can go back to using a higher ranked algorithm (3DES in this case; hence number 2 is again observed in the second column as the used algorithm) without causing any violations. Finally, by reducing the CPU load further to 10%, the systems goes back to using AES again.

Two rows are marked with A (at time 2111 and 5950) to show anomaly in the results. The first one which shows a violation for AES algorithm under 10% load, while previous rows show that it can meet its constraint under this load. This can be due to some background services doing some work in the system at that point, which has affected AES to perform encryption as before, and thus, resulted in a violation. Or, it can be because of a relatively slow movement of the slider in the CPU Killer application, which is used to raise the CPU load to 50% manually, and thus resulting in the system working in a CPU load between 10% and 50% for a short while at that moment. This can also be the reason for the anomaly observed in the next row marked with A (at

time 5950). In this case, this row shows that DES has managed to complete within its time constraint under 70% CPU load. While, it could not perform so in the previous rows related to this algorithm (having number 3 in the second column). This can again be because of the relatively gradual decrease of the CPU load from 70% to 50%, causing the system to work at some CPU load in between.

Also another interesting observation from this result is that, as the consequence of using adaption, more encryption jobs have been performed in the second case (II), under a shorter period of time.

7.4 Discussion

Our suggested approach and the way we implemented it, gives this flexibility to have different time constraints for each invocation of encryption procedures. This may not, however, be needed in all systems, and only a fixed value (e.g., originating from a system level requirement) might seem to be enough for many situations, but other systems can well benefit from this flexibility.

The whole adaptation mechanism can also be used as an option; in the sense that, if a system detects certain patterns in CPU load variations and violation of timing constraints in the applied encryption algorithm, it can *turn on* adaptation mechanism and let the system decide which encryption algorithm to use. Moreover, use of the suggested adaptation mechanism may be most beneficial when there are many requests for encryption and frequency of CPU load changes are such that they make the overhead of adaptation mechanism acceptable. On the other hand, if there are only a few encryption requests or there are not big changes in CPU load (or the range of changes is very small and known beforehand), using a fixed encryption algorithm may be more desirable (to remove the overhead cost of adaptation).

The security level of the system, originating from the choice of encryption algorithms, is actually determined by the list of encryption algorithms that designers choose to include in the described encryption algorithms table. So, for example, if for a system only AES and 3DES are acceptable, the table can be constructed using only these two algorithms. This also defines what is the range of strongest and weakest encryption algorithms that the system may be using at any moment. Moreover, while we only focused on the algorithm itself, and did not discuss key length or block length explicitly, these factors (even the number of rounds), where applicable, can easily be taken into account using the table. For example, instead of just having AES, we can put AES256 and

AES128 as items in the table, to bring into picture the role of key length, and the system will choose each when decided appropriate.

Providing adaptations on encryption algorithms, also automatically leads to some sort of security through obscurity (note the famous quote of "security through obscurity is not security") [12, 13]. One interesting topic that we leave as a future direction of this work to be investigated, is that whether adaptive mechanisms, as the one described here, can lead to weaknesses in the system and facilitate the job of attackers. For instance, issues such as this can be analyzed more throughly that if attackers get to know the details of adaptation mechanisms, they might force the system into adopting the lowest ranked (weakest) encryption algorithm by creating CPU loads, and making it easier for themselves to break into the system.

7.5 Related Work

Designing security features for embedded systems has its unique challenges and requires specific engineering methods and considerations. These issues have been the subject of many studies such as [14, 1, 2]. [14] and [1], focus on these unique challenges of security in embedded systems in general, and discuss them under the *processing gap*, *battery gap*, *flexibility*, *tamper resistance*, *assurance gap*, and *cost* titles. [14] also provides workload analysis of SSL protocol, and examples for energy consumption of different ciphers, to discuss and illustrate the impacts of security features in embedded systems. The vision for security engineering of embedded systems in the scope of a project is described in [2].

[15] and [10] are examples of works that study the impacts of security mechanisms on specific aspects of a system. Measurement and comparison of memory usage and energy consumption of different encryption algorithms on two different sensor nodes (MicaZ and TelosB motes) are performed and discussed in [15], and [10] offers performance and timing comparisons of encryption algorithms on two Pentium machines. The approach we proposed in this paper, relies on the results from the performance and timing comparisons of encryption algorithms as provided in the aforementioned study.

In this paper, an adaptive way to deal with the timing costs and requirements on security mechanisms was introduced. It should, however, be mentioned that there are other ways for taking into account these timing costs, which might suit very well other types of systems than discussed here. In systems with less complexity which are analyzable, a static and non-adaptive

structure can be designed (i.e., a fixed set of security features will be used all the time e.g., to encrypt data). The idea we proposed in [4] is basically an example of such approach and systems.

The use of adaptive approaches and feedback mechanism for better CPU utilization and task scheduling in dynamic systems, where execution times of tasks can change a lot at runtime, has also been the topic of many studies in the real-time systems domain, such as [16]. One of the interesting works in the area of security for real-time embedded systems which uses an adaptive method is the study done in [17]. One difference between our work and [17] is that, there, the focus is on a set of periodic tasks with known real-time parameters, whereas, our main target is complex systems consisting of periodic, sporadic and aperiodic tasks. Therefore, the analysis and formulas they offered in that work may not be applicable or need to be extended to support the type of systems we discussed here. Also, they consider the security level of the system as a QoS value explicitly, while in this paper, it is considered implicitly and left to the user through the use of a sorted table for encryption algorithms. Moreover, the main adaptation component of the system in that work is key length, while in our work it is the encryption algorithms that are adaptively replaced, and can easily include key length or any other relevant parameters as well. One of the interesting and close studies to our work is [18]. They basically use a similar type of adaptation mechanism as ours. However, the main focus in this work is on client-server scenarios using a database, and to manage performance of transactions. Also, security level adjustment in this work is done periodically using a security manager component, while in our method, adaptation mechanism executes per request and is not active when application has no request for encryption. Moreover, in that work, while security level switch is occurring, it can lead to use of an inappropriate encryption method by a client, which is solved by rejecting it, through passing several acknowledgment messages and repeating the process. Therefore, synchronization and message loss due of out of order arrival of messages are problematic for the security manager, which is handled by re-sending of data and applying other means.

As another approach for managing security in real-time systems, Tao Xie and Xiao Qin, has basically incorporated timing management of security mechanisms as part of the scheduling policy and developed a security-aware scheduler in [19].

7.6 Conclusion and Future Work

In this paper, we discussed security, as a non-functional requirement, in the design of real-time embedded systems, and particularly, how the choice of encryption algorithms, can affect timing requirements in these systems. An adaptive approach for selection of encryption algorithms was suggested for systems which need to balance their services at runtime in order to achieve their time constraints. It was shown how the approach can help the system to react to different processing loads and perform its encryption procedures within the defined time constraints. While, OSE RTOS was used as the base platform for implementation of the approach, there is nothing that stops it from being implemented on other platforms.

In the suggested approach here, a gradual increase or decrease of the rank of encryption algorithms was used. In other words, in each adaptation step, the system chooses either the next higher or lower ranked algorithm. As a future work, it can be evaluated how the approach would perform, if in each adaptation, the lowest or highest ranked algorithm was selected instead. For example, if it is observed that the system completes its encryption job earlier than its time constraint, it jumps to the top of the rank table and chooses the highest ranked algorithm for the next invocation. It would be interesting to study for which systems/situations, each of these methods work better and categorize systems accordingly. Calculating the overhead of adaptation mechanisms and optimizing them is also left as a future study.

Moreover, in this work we focused on complex systems with not much information about timing properties of each individual task in the system to perform analysis. This situation was observed in the design of a telecommunication sub-system during our work in the CHESS project [20]. Accordingly, the approach that is suggested in this paper tries to improve satisfaction of timing constraints of the system by keeping a history of the timing behavior of the system. There is room to improve the suggested adaptation mechanism by taking into account more information about the system than was used here, and also more knowledge about the task that requires encryption when available.

Another direction of this work is to introduce other factors besides time for performing adaptations. These factors may include energy consumption, memory usage, and even situations where a system is under attack. Moreover, it was discussed whether and how adaptation mechanisms might actually help attackers to break more easily into a system. This issue can serve as an interesting topic for more careful investigations.

Bibliography

[1] 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.

[2] 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, pages 7:1–7:6, New York, NY, USA, 2010. ACM.

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

[4] 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)*, October 2011.

[5] Enea. The architectural advantages of enea ose in telecom applications. `http://www.enea.com/Templates/Landing.aspx?id=27011`, Last Accessed: September 2011.

[6] Anders Wall, Johan Andersson, Jonas Neander, Christer Norström, and Martin Lembke. Introducing temporal analyzability late in the lifecycle of complex real-time systems. In *Proceedings of RTCSA 03*, February 2003.

[7] 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.

[8] 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.

[9] Enea. `http://www.enea.com`, Last Accessed: September 2011.

[10] 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.

[11] CPU Killer. `http://www.cpukiller.com/`, Last Accessed: September 2011.

[12] Rebecca T. Mercuri and Peter G. Neumann. Security by obscurity. In *Commun. ACM*, volume 46, New York, NY, USA, November 2003. ACM.

[13] Hissam S, Weinstock C., Plakosh D., Jayatirtha A. Perspectives on open source. Software Engineering Institute, Carnegie Mellong `http://www.sei.cmu.edu/library/abstracts/reports/01tr019.cfm`, Published: November 2001, Last Accessed: September 2011.

[14] 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.

[15] Jongdeog Lee, Krasimira Kapitanova, and Sang H. Son. The price of security in wireless sensor networks. In *Journal of Computer Networks*, volume 54, pages 2967–2978, New York, NY, USA, December 2010. Elsevier North-Holland, Inc.

[16] Nima Moghaddami Khalilzad, Thomas Nolte, Moris Behnam, and Mikael Åsberg. Towards adaptive hierarchical scheduling of real-time systems. In *16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'11)*, September 2011.

[17] 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.

[18] Sang H. Son, Robert Zimmerman, and Jörgen Hansson. An adaptable security manager for real-time transactions. In *Euromicro Conference on Real-Time Systems*, pages 63–70, 2000.

[19] 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.

[20] CHESS Project: Composition with Guarantees for High-integrity Embedded Software Components Assembly. `http://chess-project.ning.com/`, Last Accessed: August 2011.

Chapter 8

Paper E:
The Role of Schedulers in Model-Driven Development of Real-Time Systems

Mehrdad Saadatmand, Mikael Sjödin and Naveed Ul Mustafa
MRTC report, ISSN 1404-3041 ISRN MDH- MRTC-264/2012-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, March, 2012.

Abstract

Design of real-time embedded systems is a complex and challenging task. 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 timing behaviors of tasks and schedules them can facilitate 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 require to be re-designed and modified. In this paper, we describe our work in providing these features by introducing a second layer scheduler on top of OSE real-time operating system's scheduler. The approach can also contribute to the predictability of systems 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 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 means generation of periodic, sporadic and aperiodic tasks. However, most (industrial) Real-Time Operating Systems (RTOS) such as VxWorks, RTEMS, RT-Linux, Windows CE, OSE, and FreeRTOS do not explicitly support 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. While in theory, a real-time task is simply specified by its timing parameters, in practice and when it comes to implementations, 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. Therefore, when we look at these systems at runtime, no tangible and single runnable entity as a real-time periodic task is actually observed. 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 taken into account and 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 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 gen-

erators 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 top-level scheduler schedules the 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 implementation of real-time tasks (e.g., as timer interrupt) are modified, it will not require any changes in the code generators, and also specification of real-time tasks. This improves the portability of the generated code and transformation engines, making them as *platform-independent* [2] as possible.

In this work, we highlight the role of schedulers in model-driven development of real-time systems and how they fit and contribute to different phases of this approach. We also describe the suggested second layer scheduler we built on OSE real-time operating system [3], and demonstrate how it improves monitoring 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.

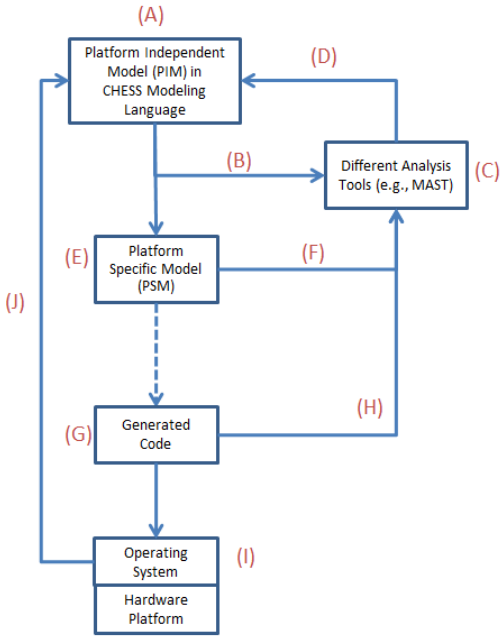

Figure 8.1: CHESS methodology

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 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, 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 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 reusability 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.

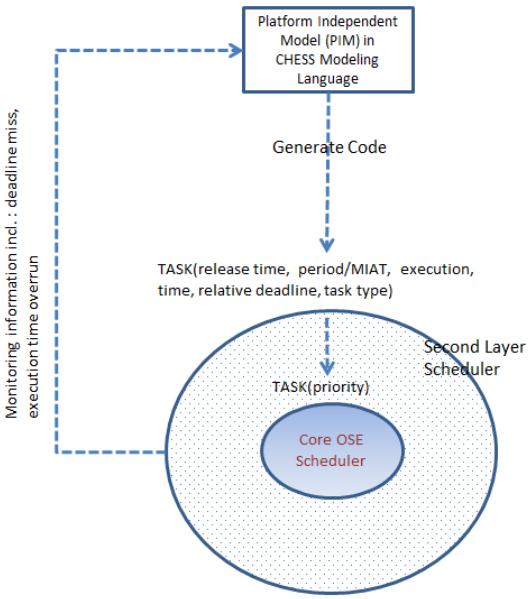

Figure 8.2: Interfaces between different parts in the suggested approach

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.

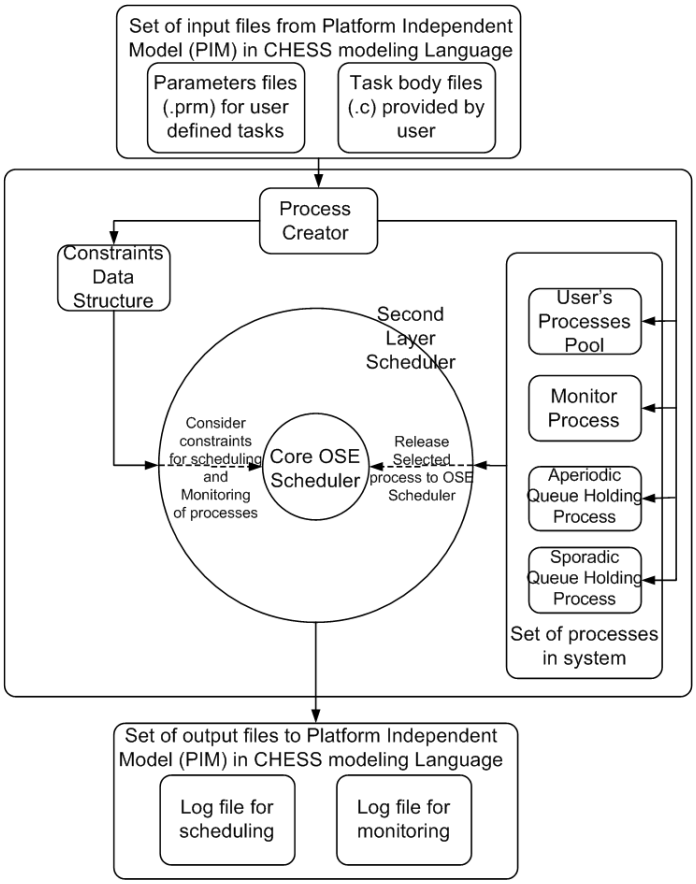

Figure 8.3: Components of The Design

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.

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. Different task types and the values by which, each task type is represented in the system are mentioned in Table 8.1.

Task Type	Value
Periodic	0
Sporadic	1
Aperiodic	2
Polling Server	3

Table 8.1: Values for different task types

- **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 sched-

uler 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.

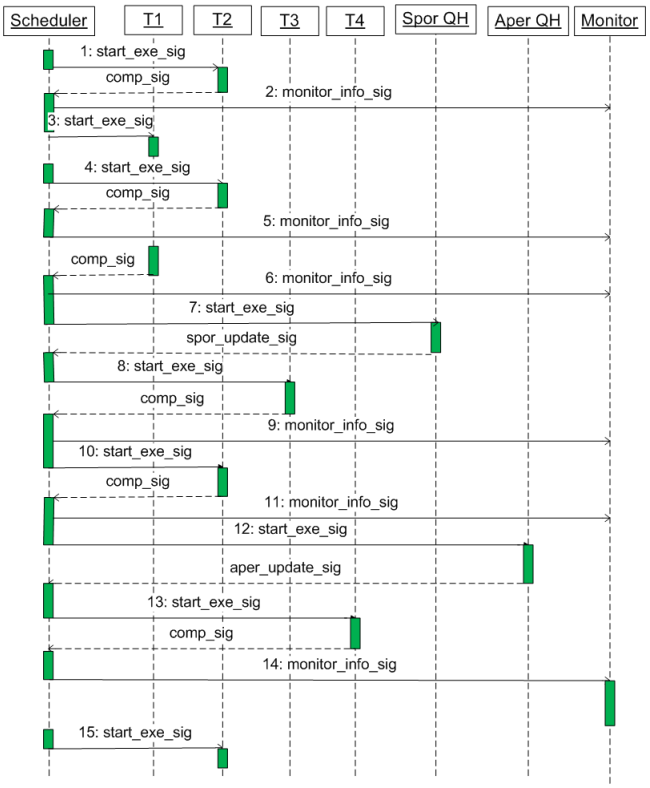

Figure 8.4: Sequence diagram to demonstrate operation of the scheduler

8.4 Experiment and Monitoring Results

The described approach has been implemented and tested on OSE SoftKernel (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 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

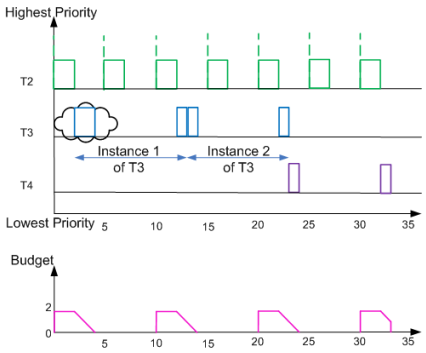

Figure 8.5: Schedule generated by the design using second layer scheduler

ticks is already elapsed (10+2=12). 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.

The work described in [12] is an attempt to implement sporadic servers in Ada. It also uses the concept of queues to manage sporadic and aperiodic tasks as we did here. However, it does not discuss real-time applications built on this design and using these servers; i.e., how sporadic and aperiodic tasks are defined and introduced to the system. [13], which also targets sporadic tasks in Ada, highlights the problem that we mentioned in this paper which many real-time platforms suffer from. It states the issue by claiming that the underlying kernel needs to provide complex execution time monitoring mechanisms at runtime and that "such mechanisms are not generally supported by Ada 9X".

One of the closest works to our approach is the implementation of a new scheduling class called *SCHED_DEADLINE* for the Linux kernel that adds EDF scheduling policy support to Linux [14]. 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 more knowledge about their timing constraints to the scheduler regardless of the scheduling policy. 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) [15, 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 (sub-system) 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 [16], 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. 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, and how it can contribute to ease code generation and enable back-annotation of data, 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 ob-

serve 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 Enea Services Stockholm AB [8].

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: February 2012.

[5] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Nonfunctional 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: February 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 pre-emptive 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] B. Sprunt and L. Sha. *Implementing sporadic servers in Ada*. Technical report. Carnegie Mellon University, Software Engineering Institute, 1990.

[13] A. Burns and A. J. Wellings. Implementing analysable hard real-time sporadic tasks in ada 9x. *Ada Letters.*, XIV:38–49, January 1994.

[14] 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.

[15] Thomas Nolte, Moris Behnam, Mikael Åsberg, Reinder J. 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.

[16] 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.