A RESOURCE-AWARE COMPONENT MODEL FOR EMBEDDED SYSTEMS

Aneta Vulgarakis

2009

School of Innovation, Design and Engineering
Abstract

Embedded systems are microprocessor-based systems that cover a large range of computer systems from ultra small computer-based devices to large systems monitoring and controlling complex processes. The particular constraints that must be met by embedded systems, such as timeliness, resource-use efficiency, short time-to-market and low cost, coupled with the increasing complexity of embedded system software, demand technologies and processes that will tackle these issues. An attractive approach to manage the software complexity, increase productivity, reduce time to market and decrease development costs, lies in the adoption of the component based software engineering (CBSE) paradigm. The specific characteristics of embedded systems lead to important design issues that need to be addressed by a component model. Consequently, a component model for development of embedded systems needs to systematically address extra-functional system properties. The component model should support predictable system development and as such guarantee absence or presence of certain properties. Formal methods can be a suitable solution to guarantee the correctness and reliability of software systems.

Following the CBSE spirit, in this thesis we introduce the ProCom component model for development of distributed embedded systems. ProCom is structured in two layers, in order to support both a high-level view of loosely coupled subsystems encapsulating complex functionality, and a low-level view of control loops with restricted functionality. These layers differ from each other in terms of execution model, communication style, synchronization etc., but also in kind of analysis which are suitable. To describe the internal behavior of a component, in a structured way, in this thesis we propose REsource Model for Embedded Systems (R EMES ) that describes both functional and extra-functional behavior of interacting embedded components. We also formalize the resource-wise properties of interest and show how to verify whether the behavioral models satisfy them.
Abstract

Embedded systems are microprocessor-based systems that cover a large range of computer systems from ultra small computer-based devices to large systems monitoring and controlling complex processes. The particular constraints that must be met by embedded systems, such as timeliness, resource-use efficiency, short time-to-market and low cost, coupled with the increasing complexity of embedded system software, demand technologies and processes that will tackle these issues. An attractive approach to manage the software complexity, increase productivity, reduce time to market and decrease development costs, lies in the adoption of the component based software engineering (CBSE) paradigm. The specific characteristics of embedded systems lead to important design issues that need to be addressed by a component model. Consequently, a component model for development of embedded systems needs to systematically address extra-functional system properties. The component model should support predictable system development and as such guarantee absence or presence of certain properties. Formal methods can be a suitable solution to guarantee the correctness and reliability of software systems.

Following the CBSE spirit, in this thesis we introduce the ProCom component model for development of distributed embedded systems. ProCom is structured in two layers, in order to support both a high-level view of loosely coupled subsystems encapsulating complex functionality, and a low-level view of control loops with restricted functionality. These layers differ from each other in terms of execution model, communication style, synchronization etc., but also in kind of analysis which are suitable. To describe the internal behavior of a component, in a structured way, in this thesis we propose REsource Model for Embedded Systems (REMES) that describes both functional and extra-functional behavior of interacting embedded components. We also formalize the resource-wise properties of interest and show how to verify whether the behavioral models satisfy them.
To my parents
To my parents
Acknowledgements

I have always known that I wanted to get a higher degree than Master of Science, and I have always been fascinated by research. However, I never thought that I would ever live in Sweden. Coming from Macedonia, Sweden has always seemed to me just “too north”. But, then I was offered a Ph.D. candidate position at Mälardalen University, which I simply could not refuse. That is how my research journey started. I can not say that the journey has at all times been “a piece of cake” for me, but I can definitely say that I had great support from many people that made it a lot easier.

The work presented in this thesis would not have been possible without the encouragement and guidance of my supervisors. My deepest thanks goes to my main supervisor Ivica Crnković, for giving me the opportunity to be a Ph.D. student and believing in me. I am always impressed by your ability to work so much, and still be so positive and energetic. Second, I want to thank my assistant supervisor Paul Pettersson. I am amazed by your ability to make research topics seem less complicated. Last but not least, I want to thank my second assistant supervisor Cristina Seceleanu. You have not been just my supervisor, but an invaluable friend that has helped me in so many ways. Thank you so much for this!

I have authored and co-authored 16 different papers. I would never have done that without the help of very capable and hard working co-authors. Many thanks go to Tomáš Bureš, Jan Carlson, Aida Ćaušević, Michel Chaudron, Ivica Crnković, Sérénine Sentilles, Jagadish Suryadevara, Cristina Seceleanu and Paul Pettersson.

I would like to thank PROGRESS-ers Andreas Ermedahl, Hans Hansson, Björn Lisper, Kristina Lundqvist, Christer Norström, Sasikumar Punnekkat, Mikael Sjödin, and Gunnar Widforss. Without you PROGRESS would not have progressed to the point it is today. I would also like to thank Gordana Dodig-Crkovíc and Jan Gustafsson for introducing me to the research methodology.
Acknowledgements

I have always known that I wanted to get a higher degree than Master of Science, and I have always been fascinated by research. However, I never thought that I would ever live in Sweden. Coming from Macedonia, Sweden has always seemed to me just “too north”. But, then I was offered a Ph.D. candidate position at Mälardalen University, which I simply could not refuse. That is how my research journey started. I can not say that the journey has at all times been “a piece of cake” for me, but I can definitely say that I had great support from many people that made it a lot easier.

The work presented in this thesis would not have been possible without the encouragement and guidance of my supervisors. My deepest thanks goes to my main supervisor Ivica Crnković, for giving me the opportunity to be a Ph.D. student and believing in me. I am always impressed by your ability to work so much, and still be so positive and energetic. Second, I want to thank my assistant supervisor Paul Pettersson. I am amazed by your ability to make research topics seem less complicated. Last but not least, I want to thank my second assistant supervisor Cristina Seceleanu. You have not been just my supervisor, but an invaluable friend that has helped me in so many ways. Thank you so much for this!

I have authored and co-authored 16 different papers. I would never have done that without the help of very capable and hard working co-authors. Many thanks go to Tomáš Bureš, Jan Carlson, Aida Čaušević, Michel Chaudron, Ivica Crnković, Séverine Sentilles, Jagadish Suryadevara, Cristina Seceleanu and Paul Pettersson.

I would like to thank PROGRESS-ers Andreas Ermedahl, Hans Hansson, Björn Lisper, Kristina Lundqvist, Christer Norström, Sasikumar Punnekkat, Mikael Sjödin, and Gunnar Widforss. Without you PROGRESS would not have progressed to the point it is today. I would also like to thank Gordana Dodig-Crnković and Jan Gustafsson for introducing me to the research methodology,
Rikard Land and Frank Lüders for the stimulating collaboration in the courses Distributed Software Development and Software Engineering, and the administrative staff at the department, in particular Harriet Ekwall, Monica Wasell and Monika Matevska Stier.

Next, I would like to thank my officemates, Séverine Sénailles and Hongyu Pei Breivold for the talks we had, but especially for bering with my sometimes dancing behavior.

Having lunch and drinking coffee with the people from the department has been an enjoyable activity. Many ideas, mostly outside of the research were born during these breaks, such as time-machines and meta-printers. I want to thank Adnan Čaušević, Aida Čaušević, Aleksandar Dimov, Ana Petričić, Andreas Hjertström, Antonio Cicchetti, Batu Akan, Cristina Secelleanu, Dag Nyström, Damir Isović, Daniel Sundmark, Farhang Nemati, Hongyu Pei Breivold, Hüseyin Aysan, Iva Krasteva, Jan Carlson, Jagadish Suryadevara, Johan Fredriksson, Johan Kraft, Josip Maras, Juraj Feljan, Kathrin Dannmann, Lars Asplund, Leo Hatvani, Luka Lednicki, Nikola Petrović, Marcelo Santos, Mikael Åsberg, Mikael Åkerholm, Moris Behnam, Pasqualina Potena, Radu Dobrin, Séverine Sénailles, Stefán (Bob) Bygde, Thomas Nolte, Tiberiu Secelleanu and Yue Lu. Most of you have been more friends than colleges to me. Not surprisingly, I would especially like to thank Juraj for being there and making my days brighter!

Many thanks also to my Bulgarian friend Velemira Slaveykova, and my Macedonian friends Bojana Bislimovska and Marija Taškova.

To my sister Sofija, her husband Boris and my nephew Filip - you have given me positive energy when I needed it the most.

Finally, Mirjana and Janko, my parents. Thank you for always being with me, and guiding me through life. Your love and support means the world to me!

The journey continues...

Aneta Vulgarakis
Västerås, September, 2009

This work has been supported by the Swedish Foundation for Strategic Research (SSF), via the research centre Progress.
List of Publications

Publications Included in the Licentiate Thesis


Other publications, not included in the thesis

Conferences and workshops:


MRTC reports:


# Contents

I Thesis 1

1 Introduction 3
   1.1 Preliminaries .................................. 5
       1.1.1 Component Based Software Engineering ...... 5
       1.1.2 Formal Analysis ................................ 7
   1.2 Thesis Overview .................................. 11

2 Research Problems 15
   2.1 Problem Description ................................. 15
   2.2 Research Questions ................................. 16

3 Research Results 19
   3.1 Classification of Component Models ................. 19
   3.2 The REMES Behavioral Model ........................ 21
   3.3 The ProCom Component Model ........................ 22
   3.4 Questions Revisited ................................ 25

4 Research Method 27

5 Related Work 29
   5.1 Component Models for Embedded Systems ........... 29
   5.2 Resource Modeling and Analysis .................... 33

6 Conclusions and Future Work 37
   6.1 Contributions ..................................... 37
   6.2 Future Research Directions ........................ 38

Bibliography 41
II Included Papers

7 Paper A:  
A Classification Framework for Component Models  
7.1 Introduction ........................................ 49
7.2 The classification framework  
7.2.1 Lifecycle ......................................... 51
7.2.2 The constructs ..................................... 53
7.2.3 Extra-Functional Properties ....................... 54
7.2.4 Domains .......................................... 57
7.2.5 The classification overview ...................... 57
7.3 Survey of component models ......................... 61
7.3.1 “Almost” component models ...................... 62
7.3.2 Component models ................................ 65
7.4 The comparison framework ......................... 66
7.4.1 Life-cycle classification ........................ 66
7.4.2 Constructs classification ........................ 67
7.4.3 Extra-functional properties classification ........ 68
7.4.4 Domains classification .......................... 70
7.5 Related work ........................................ 71
7.6 Conclusion ......................................... 75
7.7 Survey of component models ......................... 79
Bibliography ........................................... 80

8 Paper B:  
A Component Model for Control-Intensive Distributed Embedded Systems  
8.1 Introduction ......................................... 95
8.2 The ProCom two layer component model .......... 97
8.2.1 ProSys — the upper layer ......................... 98
8.2.2 ProSave — the lower layer ....................... 99
8.2.3 Integration of layers — combining ProSave and ProSys 102
8.3 Example ............................................. 103
8.4 Conclusions ......................................... 104
Bibliography ........................................... 107

9 Paper C:  
Embedded Systems Resources: Views on Modeling and Analysis  
9.1 Introduction ....................................... 109
<table>
<thead>
<tr>
<th>Paper</th>
<th>Title</th>
<th>Sections</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>REMES: A Resource Model for Embedded Systems</td>
<td>10.1 Introduction</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>Preliminaries</td>
<td>10.2 Priced Timed Automata, Multi Priced Timed Automata</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>REMES: The Proposed Resource Model</td>
<td>10.3 Classes of resources, Introducing REMES, Composition of REMES models</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td>Analyzing REMES-based Systems</td>
<td>10.4 Analysis model for REMES, Feasibility Analysis, Optimal and Worst-Case Resource Consumption, Trade-off Analysis</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>Example: A Temperature Control System</td>
<td>10.5 A REMES Model of TCS, A PTA model of TCS, Formal Analysis of the PTA model</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>Discussion and Related Work</td>
<td>10.6</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>Conclusions and Future Work</td>
<td>10.7</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>10.8</td>
<td>154</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Paper</th>
<th>Title</th>
<th>Sections</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Formal Semantics of the ProCom Real-Time Component Model</td>
<td>11.1 Introduction</td>
<td>159</td>
</tr>
<tr>
<td></td>
<td>The Component Model</td>
<td>11.2 ProCom, Particularities of ProCom</td>
<td>161</td>
</tr>
<tr>
<td></td>
<td>Formal Semantics of Selected ProCom Architectural Elements</td>
<td>11.3</td>
<td>166</td>
</tr>
</tbody>
</table>
## Contents

11.3.1 Formalism and Graphical Notation .......................... 167
11.3.2 Formal Semantics of the FSM Language ....................... 168
11.3.3 Overview of ProCom Formalization ............................. 169
11.3.4 Services .................................................... 170
11.3.5 Data and Trigger Connections ................................. 171
11.3.6 Component Hierarchy ........................................ 172
11.3.7 Linking Passive and Active Components ....................... 173
11.4 Discussion and Related Work .................................... 174
11.5 Conclusions ..................................................... 176
Bibliography .......................................................... 178
I

Thesis
Chapter 1
Introduction

An embedded system is a microprocessor-based system that is built (embedded) in a larger system that may or may not be a computer system. Embedded systems can be found in an enormous range of electrical items such as cell-phones and PDAs, instruments such as GPS automotive navigation systems, and also large engineering systems such as traffic control systems, or control systems of nuclear power plants. Virtually any electronic device designed and manufactured nowadays is an embedded system, and virtually all people are touched by this technology.

Embedded systems have tightly constrained heterogenous requirements [1, 2]. They must often have low cost, constantly react to changes in the system’s environment, must compute certain results in real time without delay and satisfy reaction constraints, such as deadlines and throughput, must be sized to fit on a single chip and consume minimum resources, and similar. Like all computing systems, embedded systems consist of hardware and software integrations, in which the software reacts to the environment. Nevertheless, in difference to other computing systems, most of the requirements of embedded systems are related to extra-functional properties (such as reliability and safety), and to limited resources. As such, design space exploration and verification at an early design stage are desirable.

During recent decades, the vast majority of functionality of embedded systems is realized with software. For example, up to 40 percent of the development time for an upper-class car is spent in car-IT (such as driver assistance) [3]. Nowadays, a car may hold up to 80 control-units that are cross-linked. The existing theories and methods for software development, when
Chapter 1

Introduction

An embedded system is a microprocessor-based system that is built (embedded) in a larger system that may or may not be a computer system. Embedded systems can be found in an enormous range of electrical items such as cell-phones and PDAs, instruments such as GPS automotive navigation systems, and also large engineering systems such as traffic control systems, or control systems of nuclear power plants. Virtually any electronic device designed and manufactured nowadays is an embedded system, and virtually all people are touched by this technology.

Embedded systems have tightly constrained heterogenous requirements [1, 2]. They must often have low cost, constantly react to changes in the system’s environment, must compute certain results in real time without delay and satisfy reaction constraints, such as deadlines and throughput, must be sized to fit on a single chip and consume minimum resources, and similar. Like all computing systems, embedded systems consist of hardware and software integrations, in which the software reacts to the environment. Nevertheless, in difference to other computing systems, most of the requirements of embedded systems are related to extra-functional properties (such as reliability and safety), and to limited resources. As such, design space exploration and verification at an early design stage are desirable.

During recent decades, the vast majority of functionality of embedded systems is realized with software. For example, up to 40 percent of the development time for an upper-class car is spent in car-IT (such as driver assistance) [3]. Nowadays, a car may hold up to 80 control-units that are cross-linked. The existing theories and methods for software development, when
applied to software design of embedded systems, reveal the two major challenges of embedded system design. The first challenge is to provide an artifact (an embedded computer system) that provides the specified services under given constraints. The second challenge is that relevant properties of this artifact need to be modeled at different levels of abstraction by models of adequate simplicity [4]. Accordingly, there is a need for improved software development techniques and processes that will let developers to tame software’s growing complexity, while reducing time to market and development costs. A promising approach to handle the complexity, reduce time to market, introduce structure and abstractions, lies in the adoption of the component-based software engineering (CBSE) paradigm. The central point of CBSE has been the reuse, but for embedded systems the structure and abstractions introduced by components are equally important as a basis for construction of abstract formal models. In that sense, the CBSE paradigm facilitates the use of formal methods, in modeling and analyzing the used components, to tackle the need for early stage verification.

The goal of this thesis is to propose solutions for modeling modern real-time embedded systems, in a component-based fashion, in an attempt to manage the associated extra-functional properties including resource constraints. Following the CBSE spirit, this thesis introduces an analyzable component model for development of distributed embedded systems, which tries to meet the designer’s needs for building vehicular embedded systems in particular. The component model is built in two layers, in order to address in same time loosely coupled subsystems (big parts) and control tasks (small parts) of a system. These parts differ from each other in terms of execution model, communication style, synchronization etc., but also in kind of analysis which are appropriate.

While a fully and semantically described interface of a component defines the intent of a component, that is, what the component does, the content of a component describes how the intent is realized [5]. Such information is hidden from the end user and becomes important only to those who intend to modify the component. Hence, in order to provide the designer with means for representing the internal behavior of a component, in a structured way, in this thesis we also introduce a model that describes both functional and certain class of extra-functional (such as timed behavior and resource consumption) behavior of components. Any modification of a component’s internal description, even if gives rise to a functionally equivalent model, might alter the component’s original properties wrt timing and resource usage. To prove that the desired properties are still exhibited by a modified component model, we formalize the
resource-wise properties of interest and show how to verify such behavioral models against them.

The work has been carried out within PROGRESS [6], a Swedish national centre for development of predictable embedded systems. The main aim with PROGRESS is to promote the development of embedded systems to a mature engineering discipline. Thus, PROGRESS should provide theories, methods and tools, which will increase quality, reduce costs and complexity in the development of embedded systems.

The following section provides the background for the basic concepts of CBSE, and formal analysis, as a foundation for reading the remainder of the thesis. In the end of this chapter the overview of the thesis is presented.

1.1 Preliminaries

1.1.1 Component Based Software Engineering

The basic rationale for the field of CBSE [7, 8] is the idea of constructing systems by reusing existing components, in much the same way as standard components are used in electronics or mechanics: integrated circuits, switches, etc. It is a promising approach for efficient software development, facilitating well defined software architectures and reuse.

With CBSE it is possible to divide large and complex software systems into smaller, less complex modules. These modules can be decoupled from each other and thus be implemented in parallel by different developers, independently of each others work. Therefore, development time is reduced. Virtually reliability is increased because components which have been tested thoroughly and worked good for one system may be reused in another system. The extra time and effort required for selecting, evaluating, adapting, and integrating components is mitigated by avoiding the much larger effort that would be required to develop such components from scratch. Another advantage is that software systems which consist of several modules are more flexible and maintainable than monolithic software systems.

Although CBSE has been widely used for software development of desktop and distributed enterprise applications, there is still a lack of broadly adopted component technology standards which are suitable for embedded systems. Due to the specific characteristics of embedded systems, a component architecture for embedded systems must have low overhead, be flexible to accommodate application unique requirements, and be able to address relevant extra-
functional issues (resource restrictions, timeliness, safety and dependability).

In CBSE the smallest functional building unit is a *component*. The idea behind components originates from a paper published by M.D. McIlroy [9] at the NATO conference in Garmisch in 1968 about the idea of mass-produced software components. However, since McIlroy’s paper, component definitions and notions advanced in various, and in same time contradictory directions. Up until today there is no generally accepted definition of what a component is. A definition that is commonly cited in publications is the one from Szyperski [8], which focuses on the key characteristics of components:

*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*

This definition implies that in order a component to be deployed independently, a clear distinction between the environment and other components is required. A component must have clearly specified interfaces and the component’s implementation must be encapsulated in the component and not be directly reachable from the environment. The definition inclines that components should be delivered in binary form, and that deployment and composition should be performed at run-time. Regardless of its generality, it was shown that Szyperski’s definition does not fully cover a wide range of component-based technologies (e.g., those which do not support contractually specified interface or independent deployment). Further, embedded systems require optimal utilization of hardware (which in many cases has limited resources), and a predictable behavior, rather than flexibility at run time. A static compilation of components into an image is proven to be more efficient and more accurate than dynamic uploading of components. For this reason in embedded systems components are usually expressed as models or source code.

A component based system is a composition of components, where a component is an open system that communicates with the environment through its interfaces. The behavior of an embedded system should be predictable, both functionally and with respect to timeliness and resource usage. Ideally, the behavior of a component should be the same regardless of the environment in which it is deployed, i.e., the other components in the system, but this is not straightforward to achieve for properties such as timing, resource usage or reliability. Although the behavior modeling and analysis of an embedded system is very important it is often omitted in component models targeting embed-
ded system design. Thus, there is a need to include behavioral modeling in embedded systems.

Component models are used in the development of components to define standards for their interfaces, illustrate their dependencies, specify their properties and composition mechanisms [7]. In other words, the component model embraces a set of rules regulating how the components may or may not be used.

Nowadays a number of component models for embedded systems exist [10–15], however they seldom provide support for relevant extra-functional properties.

### 1.1.2 Formal Analysis

Component technologies for embedded systems should support system development with high degree of predictability. Predictability concerns the possibility to guarantee absence or presence of certain properties, or to predict/guaranty a value of a property. The employed predictability analysis should guide the design and selection of hardware and software system components.

Formal analysis is a process of rigorously exploring the correctness of system designs expressed as abstract mathematical models, most likely with the assistance of a computer. In this thesis, we consider two types of answers to formal analysis: “yes/no” answers as a result of verifying properties that can be either satisfied or not, but cannot be measured, and answers in form of numbers, in the sense that the formal analysis returns a computed number that might represent, in our case, the minimum/maximum value of the accumulated resource usage for reaching a given goal expressed as a reachability property for instance.

Today the best known formal analysis methods are model-checking and theorem-proving, both of which have sophisticated tool support and have been applied to non-trivial systems [16, 17]. Theorem-proving emphasizes highest assurance (theorems can only be created by a logical kernel, which implements the inference rules of the logic) and handling infinite-state systems, the main challenge being proof automation. Model-checking emphasizes automation, by relying on various efficient algorithms for deciding temporal logic formulas on finite state models, the main challenge being to reduce problems to a form in which they can be efficiently model checked. The advantage of model-checking of providing high level input languages that support the modeling and checking of complex computer systems, and the highest degree of automation, justify our choice for model-checking as the verification paradigm.

To perform model-checking, an automata model describing the possible
system behaviors is fed into a model-checking tool, together with a desired property (requirement) expressed in a temporal logic. The tool then automatically traverses the system’s state space in an exhaustive manner. If an invariant property is satisfied, the tool finishes the verification successfully, or if the invariant property is violated, it reports one of the traces that violates the property as a counter-example to the model. For reachability properties the opposite is true i.e., a trace is reported when the property is satisfied. Model-checking has achieved huge success in industry for verifying hardware designs. Companies, such as IBM, Intel, Motorola, Siemens are having in-house model-checking groups. Despite these successes, formal analysis has not been widely used in the development of embedded systems. One possible reason is the lack of expertise of design engineers for constructing and understanding abstract models in an interactive environment formal specifications.

Due to the real-time requirements of embedded systems and the need to verify the models against them, the designer should be equipped with methods and tools that support modeling of real-valued variables, and the combination of discrete and continuous behaviors. The framework of timed automata is an established formal framework to support such needs, and the UPPAAL [18] tool is one of the most popular and mature verification tools based on timed automata, and it is also used in this thesis. In the following, we recall the model of timed automata and the model of priced (or weighted) timed automata [19, 20], an extension of timed automata [21] with prices/costs on both locations and edges.

**Timed Automata**

The model of timed automaton (TA) [21] is a timed extension of the finite-state automaton. A notion of time is introduced by a set of non-negative real numbers, called clock variables, which are used in clock constraints to model time-dependent behavior. TA consists of a finite set of locations, connected by edges. One of the locations is marked as initial. All clocks in TA start at zero, evolve continuously at the same rate, and can be tested and reset to zero. Edges are labeled with guard expressions, an action, and a reset set i.e., set of clocks to be reset. We say that an edge is enabled if the guard evaluates to true and the source location is active. Locations are labeled with clock constraints called invariants, which enforce that the location is left before they are violated. The semantics of TA is defined in terms of a timed transition system. A state of TA depends on its current location and on the current values of its clocks. The transitions between states can be of two kinds: delay and discrete. Delay tran-
sitions are result of passage of time while staying at some location. Discrete transitions are result of following an enabled edge in a TA to its destination location with the clocks in the reset set, set to zero. Systems comprising multiple concurrent processes are modeled by networks of timed automata, which execute with interleaving semantics and synchronize on channels.

UPPAAL is a tool set for modeling, simulation, and verification of networks of timed automata. The UPPAAL model checker supports verification of temporal properties, including safety and liveness properties. The simulator can be used to visualize counter examples produced by the model checker. UPPAAL automata extend timed automata by introducing bounded integer variables, binary and broadcast channels, and urgent and committed location.

An example of a network of timed automata modeled in UPPAAL is shown in Figure 1.1. The timed automata consist of an automaton of a lamp and an automaton of a user. The behavior of the lamp depends on when the user presses the on/off switch. The automaton of the lamp consists of three locations Off, Dim and Bright, and one clock \( t \). The automaton starts at location Off. In case the user presses the switch the automaton of the lamp switches to location Dim and the clock \( t \) is reset, by the assignment \( t:=0 \). In location Dim the automaton can remain as long as the clock is smaller or equal to 10. However, if the user presses the switch of the lamp before 5 time units have elapsed then the automaton of the lamp switches to location Bright, in which it stays until the next pressing of the switch. Processes lamp and user synchronize by sending and receiving events through channels. Sending and receiving via a channel \texttt{press} is denoted by \texttt{press!} and \texttt{press?}, respectively.

![Figure 1.1: Timed automaton of a lamp and a user.](image-url)
Priced Timed Automata

Priced timed automata extend timed automata with prices/costs on both locations and edges. The cost labeling a location represents the price per time unit for staying in that location, whereas the cost labeling an edge represents the price for taking the transition. As such, every run in the priced timed automation has a global cost, which is the accumulated price along the run of every delay and discrete transition. Multi priced automata [22] are extension to priced timed automata in which a timed automation is augmented with more than one cost variable. In this thesis, the framework of priced timed automata is used for formally analyzing resource consumption in embedded systems.

Figure 1.2: Priced timed automaton of a lamp.

Switching on a lamp and letting it burn uses energy, therefore in Figure 1.2 is depicted a priced timed automaton of the lamp elaborated earlier. The energy consumption is modeled by using costs. A special variable cost can be increased explicitly on an edge by an update, or implicitly by specifying a rate. Guards and invariants are, however, not allowed to refer to the cost variable. The switch of the lamp from location Off to Dim is labeled with an update cost+=50, indicating that the cost is 50 for switching on the lamp. In locations Dim and Bright we have the cost rates cost’== 10 and cost’== 20, respectively, which indicate that the energy consumption is 10 and 20 units per time unit in the respective locations. When staying in these locations, cost is increasing linearly with time, with rate 10 and 20, respectively.
1.2 Thesis Overview

The thesis is divided into two distinct parts. The first part is a summary of the performed research. Chapter 1 describes the background and motivation of the research. Chapter 2 formulates the main research goal and introduces the research questions. Chapter 3 describes the research results and recapitulates the research questions. Chapter 4 presents the research method used. Chapter 5 surveys related work. Finally, Chapter 6 concludes the thesis, summarizes the contributions and outlines future work that formulates guidelines for further PhD studies.

The second part of the thesis presents a collection of peer-reviewed journal, conference and, workshop papers that contain details of the answers of the research questions, methods and, results presented in the first part of the thesis. The following five papers are included in the second part of the thesis:


*Summary:* This paper presents a survey of a number of component models, described and classified with respect to a four dimensional classification framework, which groups different aspects of the development process of component models. As such, this classification framework identifies common characteristics as well as differences between selected component models. The results of the comparison have led to some observations which are discussed in this paper.

*Contribution:* This paper was mostly written with equal contribution of the first three authors. All the coauthors have contributed with ideas, discussions, and reviews. I was responsible mainly for the lifecycle section and shared the responsibility with Séverine Sentilles for collecting, analyzing and classifying in tables the included component models. The classification framework was developed in several iteration steps including observations and analysis. It was discussed with several CBSE and empirical software engineering researchers and experts from different engineering domains.

Component-Based Software Engineering (CBSE2008), Karlsruhe, Germany, October, 2008.

**Summary:** In this paper, the two-layered ProCom component model for design and development of control-intensive distributed embedded systems is introduced. ProCom takes into account the most important characteristics of these systems and employs the concept of reusable components throughout the whole development process, from early design to deployment. The two-layered model is developed to efficiently cope with different design paradigms that exist at different abstraction levels of embedded systems (high level view of loosely coupled subsystems and a low-level view of control loops controlling a particular piece of hardware). Additionally it provides ground for analysis and predicting properties (e.g., timed behavior and resource consumptions) in such systems.

**Contribution:** This paper was written with equal contribution from all the authors. I took part in the discussions and contributed with writing and improving parts of the paper, particularly in the discussions about the semantics of the component model, analysis and predicting properties and the related work section. The ProCom component model that we describe in this paper was developed in several iteration steps resulting from the conducted discussions between the authors.


**Summary:** In this paper, we discuss several representative frameworks that model and estimate resource usage of embedded systems, identifying their advantages and limitations. As such, we divide the variety of approaches existing in the literature into three distinctive categories: code-level resource modeling and analysis of component assemblies, UML-based description of embedded resources and higher-level formal approaches based on temporal logics and process algebras. In the end, we present the resource-aware development view that we are adopting throughout the rest of the thesis.

**Contribution:** This paper was written with equal contribution from all the
1.2 Thesis Overview

authors. I was specifically working on the code-level and UML-based resource modeling and analysis.


*Summary:* This paper introduces the model REMES for formal modeling and analysis of both functional and extra-functional behavior of interacting embedded components. REMES is a state-based behavioral language with support for hierarchical modeling, resource description, continuous time, and notions of explicit entry and exit points that make it suitable as a semantic basis for component-based modeling of embedded systems. The analysis of REMES-based systems is placed around a weighted sum in which the variables capture the accumulated consumption of resources, respectively.

*Contribution:* This paper was written with equal contribution from all the authors. I particularly worked on the classification of the resources and specified, modeled in REMES, and analyzed in UPPAAL CORA [23] the TCS system presented as a case study in the paper.


*Summary:* In this paper, we define the formal semantics of the ProCom component model in a small but powerful finite state-machine based formalism, with notions of urgency, timing, and priorities. As such, the formalism provides an unambiguous description of the modeling elements of ProCom, sets the ground for formal analysis using other formalisms, and provides and intuitive and useful description for both practitioners and researchers.

*Contribution:* I was the main driver and principal author of this paper and contributed with defining a formal semantics of the ProCom component model and exemplifying it on the modeling elements of ProCom. All the coauthors
have contributed with valuable discussions and reviews. The paper proceeded from a technical report that was written together with the second author of this paper.
Chapter 2

Research Problems

This chapter presents the scope of our work by formulating the research goal, and introducing the research questions that address the goal.

2.1 Problem Description

Our research is in the area of component based development for embedded systems, and was driven by the problems coming from the embedded systems domain, such as managing complexity, distribution, resource limitations, analysis, managing the strong coupling between the components, the system and the target platform. In this thesis, we address the problem of modeling modern real-time embedded systems, in a component-based fashion, in an attempt to tackle the system complexity and manage the associated resource constraints. Concretely, the overall goal of the thesis is

\textit{design of an analyzable component model for real-time embedded systems}

This goal is broad and admits various answers. We approach the goal by answering to three research questions that address component models for embedded systems and behavioral modeling of embedded systems, which we formulate in the next section.
2.2 Research Questions

Research question 1.

Component models are indispensable to CBSE, as they define rules for constructing individual components and for assembling them into systems. There are various component models proposed in the literature as suitable for the development of real-time embedded systems. Some characteristics are shared among the component models, yet each of the latter has distinct characteristics too. Therefore, it is important to analyze and compare the existing component models in order to identify the most interesting for the development of embedded systems. Such motivation justifies our first research question:

What are the common characteristics and differences between existing component models?

(Q1)

Research question 2.

One of the main characteristics of embedded systems is the restriction of available resources. The diversity of approaches on resource modeling and analysis existing in the literature [24–31] indicate the difficulty of handling all relevant embedded resources within the same formal model. This calls for an innovative look on resource-aware design methods, based on the experience gathered from the existing modeling approaches. In order to properly specify and analyze embedded systems, the designer should use a modeling language that incorporates resources as primitive types, that is, built-in the model. Ideally, the same language should provide support for modeling and analyzing functional and timing behavior too, besides the resource-wise behavior of the embedded system. This would allow both separation of concerns as well as simple model-to-model transformations, for analysis purposes. Accordingly, the second research question can be formulated:

How can we model and formally analyze functional, timing and resource-wise behavior in a unified manner?

(Q2)
2.2 Research Questions

Research question 1.
Component models are indispensable to CBSE, as they define rules for constructing individual components and for assembling them into systems. There are various component models proposed in the literature as suitable for the development of real-time embedded systems. Some characteristics are shared among the component models, yet each of the latter has distinct characteristics too. Therefore, it is important to analyze and compare the existing component models in order to identify the most interesting for the development of embedded systems. Such motivation justifies our first research question:

What are the common characteristics and differences between existing component models?

(Q1)

Research question 2.
One of the main characteristics of embedded systems is the restriction of available resources. The diversity of approaches on resource modeling and analysis existing in the literature [24–31] indicate the difficulty of handling all relevant embedded resources within the same formal model. This calls for an innovative look on resource-aware design methods, based on the experience gathered from the existing modeling approaches. In order to properly specify and analyze embedded systems, the designer should use a modeling language that incorporates resources as primitive types, that is, built-in the model. Ideally, the same language should provide support for modeling and analyzing functional and timing behavior too, besides the resource-wise behavior of the embedded system. This would allow both separation of concerns as well as simple model-to-model transformations, for analysis purposes. Accordingly, the second research question can be formulated:

How can we model and formally analyze functional, timing and resource-wise behavior in a unified manner?

(Q2)

Research question 3.
The potential benefits of CBSE are as attractive in the domain of embedded systems as they are in other areas of the software industry. Beside component models, component technologies form another central concept of CBSE. They make use of component models in practice, that is, a particular component technology provides tools that enable development and deployment of systems that adhere to a corresponding component model. Although there exist several component models and technologies for the development of embedded systems (e.g., Koala [10], Robocop [14], BlueArX [15], AUTOSAR [32], COMDES-II [33], Pecos [12], Rubus [13], and SaveCCM [11]), CBSE is still not broadly used in the embedded systems industry. An important reason for such limited success is the difficulty of providing solutions that meet typical embedded system requirements. Wolf [34] discusses about which domain specific requirements a component technology targeting embedded system development should be aware of. In the embedded systems domain, designing for predictability requires architectures that meet both the corresponding functional requirements (e.g., expected services, functionality and features), as well as extra-functional ones (resource-feasibility, timing and/or reliability). Hardware and software models annotated with performance, resource consumption or size information can be beneficial to embedded system designers. In order to simplify analysis and help the intuition behind the embedded system’s functioning, one could create a hierarchy of models that will allow them to reason about timed behavior, resource consumption and so on, without going down to the instruction level. For instance, architectural models may be used for modeling basic functionality, and behavioral models for modeling functional and extra-functional behavior. Also, embedded system developers must verify that applications meet their functional and extra-functional specification. All these requirements should be reflected in the component model. However, the specifications of many component models are defined informally and component models suffer from incomplete and imprecisely defined syntax and semantics. Formalization of component models using formal methods can provide precise definitions. The formalization should be designed to unambiguously describe the elements of the component model. Thus the third question is as follows:

What is an appropriate component model for real-time embedded systems and how can we describe its elements in an unambiguous way?

(Q3)
Chapter 3

Research Results

The current chapter presents the main lines of our contribution and research results, starting from the research questions proposed in Chapter 2. The research does not provide complete answers to the questions, but only partially answers them and gives directions for further research. The following sections describe each research topic.

3.1 Classification of Component Models

Goal:

The large number of existing component models having particularities, different aims, sometimes unclear concepts, but also many similarities, calls for a systematic analysis of such models. The goal of this research is to provide a classification framework that will identify and discuss the basic principles of component models. Later, according to this classification framework, existing component models may be classified and compared. This framework could also help in the design of new component models, since one of its goals is to identify the elements of a component model that would be important when designing a (new) component model.

Research process:

The research was performed in several iterations of observations, analysis, classification, and validation. We have started with a large number of component models, by first studying the state of the art on general principles of CBSE, and the existing literature on classification of architecture description languages, quality attributes and component models. From this we...
Chapter 3

Research Results

The current chapter presents the main lines of our contribution and research results, starting from the research questions proposed in Chapter 2. The research does not provide complete answers to the questions, but only partially answers them and gives directions for further research. The following sections describe each research topic.

3.1 Classification of Component Models

**Goal:** The large number of existing component models having particularities, different aims, sometimes unclear concepts, but also many similarities, calls for a systematic analysis of such models. The goal of this research is to provide a classification framework that will identify and discuss the basic principles of component models. Later, according to this classification framework, existing component models may be classified and compared. This framework could also help in the design of new component models, since one of its goals is to identify the elements of a component model that would be important when designing a (new) component model.

**Research process:** The research was performed in several iterations of observations, analysis, classification, and validation. We have started with a large number of component models, by first studying the state of the art on general principles of CBSE, and the existing literature on classification of architecture description languages, quality attributes and component models. From this we
have gained knowledge and made the first version of the classification framework. Then, we discussed this classification framework with several CBSE and empirical domain software engineering researchers and experts from different engineering domains. The discussions have led to a refinement of the framework. In the next iterations the refined framework was mapped on the studied and new component models and discussed with other researchers and practitioners. The research process has been completed when all studied component models complied well with the classification framework.

**Results:** The result of this research is a classification and comparison framework for component models. The classification framework includes four dimensions (lifecycle, constructs, extra-functional properties and domains) in which the basic characteristics and principles of component models are distinguished: (i) The lifecycle dimension identifies the support provided (explicitly or implicitly) by the component model during components’ lifecycle, such as modeling of components and component based systems, implementation, packaging and distribution, and deployment of components into an executable system or some target environment. (ii) The constructs dimension identifies (i) the component interface used for the interaction with other components and external environment, and (ii) the means of component binding and communication. (iii) The extra-functional properties dimension identifies specifications of different property values, and means for management and their composition. (iv) The domain dimension classifies component models according to their usage domain: general-purpose, specialized or generative. Details about the classification framework can be found in the included paper A.

**Limitations and future work:** The proposed framework can always be extended since it does not comprise all the elements of all component models. Some component models have specific solutions related to particular models or technologies. Furthermore, we have not characterized the components themselves (e.g., internal behavior, whether components are active or passive, etc.) and the list of component models that we have studied can always be extended. This is subject to future work. However, to our knowledge the proposed framework identifies the minimal criteria for considering a model to be a component model and it groups the basic characteristics of the models.
3.2 The REMES Behavioral Model

**Goal:** The competing or inconsistent requirements of real-time embedded systems, such as minimizing memory consumption while still ensuring all deadlines are met at run-time, call for rigorous analysis of the system’s resource consumption already at early design stages. Our goal is to propose a model for formal modeling and analysis of embedded resources. The envisioned outcome has been a behavioral model with support for reasoning about functional, timing and resource-wise behavior in a unified way.

**Research process:** The research included several iterations. First we have studied and grouped several representative frameworks that model and estimate resource consumptions of embedded systems. The studied frameworks indicate the possible difficulty of reasoning about all resource types within the same theoretical framework. While studying them, we developed our own view on how to model and, carry out analysis of embedded resources. In developing our behavioral model we were inspired by the CHARON [35] modeling language, used for specifying embedded systems as communicating agents, while relying on hybrid automata for the semantic translation. Our main contribution is the introduction of resource as a built-in data type, and the addition of other constructs (like the conditional connector) that facilitate the application of REMES to modeling both functional and extra-functional behavior of component-based embedded systems.

**Results:** The result of this research is the behavioral model REMES (Resource Model for Embedded Systems) and associated analysis techniques for performing resource-wise behavioral analysis, such as feasibility analysis, optimal/worst-case resource consumption, and trade-off analysis. In our studies we consider resources as quantities of finite size, and we classify them by their discrete or continuous nature, the way they are consumed and/or allocated and released, and whether they can be referred to, or not. The classification of resources is not tied to any particular formal semantic representation. Consequently, REMES can model number of generic resources (e.g., memory, CPU, energy, bandwidth, etc.). REMES is a dense time state-based hierarchical behavioral language with a notion of explicit entry- and exit points, continuous variables, flows and progress invariants. For formal analysis purposes, REMES can be semantically translated into timed automata or (multi) priced timed automata depending on the analysis goals (i.e., timing analysis, resource consumption, etc.). The analysis of REMES is based on a weighted sum in which
the variables capture the accumulated consumption of resources, respectively.

Details about the variety of approaches on resource modeling and analysis existing in the literature and about REMES can be found in the included papers C and D, respectively.

**Limitations and future work:** REMES is tailored for embedded systems, but it is also suitable for modeling behavior of reactive systems. We have performed analysis in UPPAAL CORA [23], which can currently only handle priced timed automata models where the weighted sum is monotonically increasing. As future work, we plan to integrate REMES in the PROGRESS IDE, by first connecting REMES with ProCom, and second, by implementing the kinds of analysis that we are interested in, in UPPAAL CORA. The scalability and appropriateness of REMES for real-world industrial applications is unfortunately not exercised in this work. We plan to apply REMES on a series of complex systems, in order to better identify its weaknesses and limitations. The proposed cost analysis model for REMES is platform-aware. Hence, as future work, it could benefit from including abstractions of platform specific tools, such as the associated compiler, linker etc. We do believe that in order to derive the costs, one could apply static analysis techniques on already implemented components. We underline the fact that the values of the weights are a subjective matter; the way they are chosen depends mostly on the designer’s experience, application domain and on the analysis goals.

### 3.3 The ProCom Component Model

**Goal:** The goal of this research is to propose a component model suitable for development of real-time embedded systems, in particular vehicular- and telecommunication systems. The type of embedded systems found in the targeted domains typically have specific characteristics when considered at different levels of granularity. The loosely coupled subsystems differ from the control loops controlling a certain piece of hardware, with respect to execution model, communication style, synchronization, etc. Also, there are differences in the kind of information that must be available and the type of analysis that is appropriate. Our goal has been the design of a component model that supports both a high-level view of loosely coupled subsystems encapsulating complex functionality, as well as low-level view of control loops having dedicated, restricted functionality, simpler communication, which control a certain piece of hardware.
To enable formal analysis, the component model should be given a formal semantics. The formalization should not only describe the modeling elements of the component model in a rigorous way, but also provide support for reasoning about functional and extra-functional behavior of the modeling elements.

**Research process:** The research process included studying existing component models (in particular SaveCCM [11]) for the development of embedded systems and discussions with domain experts from the targeted domains. From this we have identified the following requirements that a suitable component model should fulfill:

- manage complexity;
- manage the strong coupling between the system and the targeted platform;
- deal with different types of components with respect to granularity, functionality and semantics;
- utilize resources efficiently;
- provide support for different kinds of analysis (functional behavior, timed behavior, resource usage, etc.).

Following these requirements, the ProCom component model has been developed in several iterations.

The formalization of ProCom’s architectural elements included several iterative steps as well. We first started with studying the formal semantics of the SaveCCM [36] component model (ProCom’s predecessor). The formal semantics of SaveCCM is based on timed automata (TA). While carrying out the formalization work, we have managed to comprise the necessary semantic descriptions in a simpler and more compact form than TA provides: a high-level FSM-like model that abstracts away some aspects present in the corresponding TA.

**Results:** The result from this research is the ProCom component model and an associated architectural semantics based on an FSM-like representation. In comparison to other component models targeting embedded systems, ProCom addresses quality attributes, resource consumption and distribution more systematically. In order to address the different concerns at different levels of granularity, ProCom is structured in two distinct, but related, layers (ProSys
and ProSave). The two layers differ in terms of granularity, architectural style and communication paradigm. The upper layer, called ProSys, is intended for modeling the embedded system as a collection of complex active and concurrent subsystems, communicating via asynchronous message passing. The lower layer, ProSave, serves for modeling the internal design of a subsystem down to primitive functional components implemented by code. ProSave components are passive units, which communicate based on a pipe-and-filter architectural style with an explicit separation between data and control flow. In both layers, information about a component is stored in a repository, including requirements, textual documentation, formal semantics and REMES behavioral models.

The FSM language, used for the ProCom formalization, has notions of urgency, implicit timing and priorities. Its formal semantics, hence of the architectural elements of ProCom, is expressed in terms of TA with priorities [37] and urgent transitions [38]. The FSM language has graphic simplicity, making it simpler than the corresponding TA model, as it abstracts from real-valued variables and synchronization channels. The FSM models of ProCom systems can be analyzed both in a dense-time underlying framework, as well as in a discrete-time one, since TA has been recently given a sampled semantics [39]. Hence, tools such as UPPAAL [18] can be employed for early-stage verification of ProCom models, whereas discrete-time model-checkers, such as DT-Spin [40], could be used for later-stage analysis, as sampled time semantics is closer to the actual software or hardware system with a fixed granularity of time.

Details about the design and formal semantics of the ProCom component model can be found in the included papers B and E.

**Limitations and future work:** The current version of the ProCom component model is developed for the vehicular domain and focuses on the design of a class of distributed embedded systems that primarily perform real-time controlling tasks. In the future, ProCom may be extended for instance to the telecommunication domain. Additionally, ProCom has not yet been industrially verified on a real-world example case study but we plan to do this as future work. Although the FSM formalism sets the grounds for formal analysis, the semantic descriptions focus only on formalizing the correct behavior of ProCom architectural elements, without consideration for efficiency in formal analysis of the resulted models. As future work, we plan to integrate REMES with the formal semantics of ProCom.
3.4 Questions Revisited

In this section, we show how the research results and included papers give answers to the research questions.

**Question Q1: What are the common characteristics and differences between existing component models?**

From the research summary, we can see that this question is answered by the first research topic and by paper A in which a classification framework for component models is introduced. Among other things, the four-dimensional framework identifies the common characteristics and differences between existing component models.

**Question Q2: How can we model and formally analyze functional, timing and resource-wise behavior in a unified manner?**

The second research topic and included papers B and E contribute with answers to this question. It is our intention that REMES is used to model both functional and extra-functional behavior of interacting embedded components, while relying on the solid verification framework of priced timed automata.

**Question Q3: What is an appropriate component model for real-time embedded systems and how can we describe its elements in an unambiguous way?**

The second and third research topics give answers to this question. ProCom is targeting the development of embedded systems and the semantics of the ProCom architectural elements can be presented with the FSM language introduced in paper E. REMES can be used for internal behavioral modeling of ProCom-based systems, whereas the underlying PTA framework in which REMES is translated can be used for formal verification of functional and extra-functional requirements/properties.

Hence, all three questions have been at least partly answered. Needless to say, we have provided one possible answer to each question, out of a possibly large pool of valid answers.
Chapter 4

Research Method

Different research methods are suitable for different settings, and similarly different validation techniques are suitable for different types of results. The methodology that has been used in this research is based on the research steps presented in [41]. The main activities are:

1. Identification of the research problem from real-world software engineering issues.
2. Transferring the problem to a research setting, and defining the research questions. In this stage the research problem is often refined and narrowed down.
3. Analysis of the current state of the art addressing the research questions.
4. Answering the research questions and presenting the research results. This stage includes several iteration steps, such as observations, discussions, analysis and improvement of the research results.
5. Checking whether the research results adequately answer the research questions. This can be performed in several different ways, e.g., by formal proofs, by performing case-studies, by implementation of a prototype, by describing experiences etc.
6. Validating the research results in the sense of checking whether they are feasible for the real-world software engineering problem.
Chapter 4

Research Method

Different research methods are suitable for different settings, and similarly different validation techniques are suitable for different types of results. The methodology that has been used in this research is based on the research steps presented in [41]. The main activities are:

1. Identification of the research problem from real-world software engineering issues.

2. Transferring the problem to a research setting, and defining the research questions. In this stage the research problem is often refined and narrowed down.

3. Analysis of the current state of the art addressing the research questions.

4. Answering the research questions and presenting the research results. This stage includes several iteration steps, such as observations, discussions, analysis and improvement of the research results.

5. Checking whether the research results adequately answer the research questions. This can be performed in several different ways, e.g., by formal proofs, by performing case-studies, by implementation of a prototype, by describing experiences etc.

6. Validating the research results in the sense of checking whether they are feasible for the real-world software engineering problem.
Following the abovementioned activities to a great extent, we have initially defined the research problem, as stated in Chapter 2. Second, from the research problem we have identified the research questions and presented them in Chapter 2. Later we have conducted a thorough investigation of the current state of the art addressing the research questions. This investigation has resulted in two papers: paper A and paper C. Further in papers B, D and E we have presented our research results on designing a component model suitable for development of embedded systems and a behavioral model for modeling and analyzing functional, timing and resource-wise behavior in a unified manner, which are summarized and discussed in Chapter 3. Since the research done so far does not yet offer a complete solution to the research problem (a thorough validation of the results is missing), the research results have been applied on simple yet relevant “research examples” presented in papers A, B and D. Accordingly, in paper A the classification framework was demonstrated on a considerable number of component models. In paper B we have exemplified the ProCom component model on an electronic stability control system of a car, but a deeper analysis of a real case is needed. Further in paper D, we have performed a small case study demonstrating the principles of our resource modeling and analysis approach. The case study has been conducted on an abstracted version of the internal design of a temperature control system for a heat producing reactor. Again, a more detailed case study for a complete evaluation is needed. Consequently, in the aforementioned research methodology, we have completed the first 3 activities. The methodology awaits the validation of the research results, which might entail improving and extending the research results towards applicability on real-world engineering problems.
Chapter 5

Related Work

This chapter relates the contributions presented in this thesis to relevant research and practice areas, subdivided into two sections. Paper A and Paper C contain extensive related work and state of the art so here we give only a short summary.

5.1 Component Models for Embedded Systems

Nowadays many component models exist, either general purpose or dedicated to specific domain. Still, only few component models target the development of embedded systems and most of them are dedicated to specific subdomains only. In these component models, component implementations are mostly given in C programming language and components are composed before compilation. Often the component models are intended for applications of an algorithmic nature and these applications are commonly modeled as data- or signal-driven block diagrams. Another name for this is pipe-and-filter architecture. Most component models targeting embedded systems focus primarily on “small” granularity components. Although they provide techniques for handling extra-functional properties there is still need for further research to improve the theories of specifying, modeling and analyzing extra-functional properties of components and composed systems, and to develop tool support. In this section we survey some component models that have been developed specifically for application in the embedded system domain and compare them with the ProCom component model proposed in this thesis. Special attention
is dedicated to the capability of these component models to model and analyze extra-functional properties, in particular resource-related properties.

AUTOSAR (AUTomotive Open System ARchitecture) [32] component model has resulted from the cooperative research of a number of automotive manufactures and suppliers. The goal of AUTOSAR is to define a standardized platform for automotive systems facilitating the exchange of “elements” between different vehicle platforms and subsystem manufacturers. Although some similarities with ProCom exist, such as the transparent communication between subsystems and components with the use of standardized interfaces, distribution of the functionalities provided by each subsystem across several nodes, some essential differences can also be noticed. In AUTOSAR, components are runtime entities whereas in ProCom they are considered at design time. Moreover, in AUTOSAR subsystems are unaware of the characteristics of the underlying platform and not so much emphasis is put on analysis of the developed elements. The upcoming AUTOSAR 4.0 release should contain a meta-model extension for specifying timing properties and constraints of software components and it is expected that TIMMO [42] project results will strongly influence future AUTOSAR releases with respect to timing modeling.

BlueArX [15, 43] is a component model developed and used by Bosch for automotive systems, such as engine control systems or chassis systems. Each component consists of specification, documentation and implementation and has an analytic interface which is used to store components’s extra-functional properties (such as worst-case execution times, code memory, stack memory, and data memory). Input to the analytic interface is the current context such as hardware dependencies, tool chains and the setting of constants and/or calibration parameters in which the component should be applied. Properties are specified in the service level of each component and the context information is specified for each property. Semantic context information is also specified by referring to the modes (such as initialization mode, cyclic executive mode or shut-down mode). Bosch uses static analysis tool aiT [44] to analyze object code and to extract the worst-case execution time of a component, and SymTA/S [45] tool as a reasoning framework that aids analysis and prediction of timing properties. The BlueArX concepts are close to to the ProSave layer of the ProCom component model, however ProCom uses a management framework to associate extra-functional properties to components and others entities of the component model (component services, message ports, communication channels and component instances).

COMDES-II (COMponent-based design of software for Distributed Embedded Systems) [33] is a two-layered component model similar to ProCom,
developed at University of Southern Denmark. At the system (first) layer, a distributed system is modeled as a network of communicating actors, and at the second level the functionality of individual actors is further specified by interconnected function blocks. COMDES-II supports modeling architectural and behavioral aspects of systems with a goal to analyze and verify system behavior at high abstraction level and to enable automatic code generation. In difference to ProCom, the timing behavior in COMDES-II is separated from the functional behavior. The timing behavior is verified by schedulability analysis, whereas functional properties are formally verified. Ke et al. [46] show how a COMDES-II system can be equivalently transformed into UPPAAL timed automata, and verified with preservation of system operational semantics.

IEC 61499 [47] is developed by the International Electrotechnical Commission (IEC) to support the development of automation and control systems. It has evolved from IEC 61131-3 [48] standard that is widely used in the development of software for PLCs. IEC 61499 components are called function blocks. Similar to ProSave, the data between the blocks is transferred using pipe-and-filter paradigm and the execution of the function blocks is event driven. In comparison to ProCom, there is no support for specifying or reasoning about extra-functional properties.

Koala component model [10] is designed and used by Philips for the development of software in consumer electronics (such as TVs, VCRs, and DVDs). Components are connected via provided and required interfaces that depict a small set of semantically related functions. The Koala component model is hierarchical, so, compound components may be defined. Since Koala components are delivered as source code, it is possible to statically analyze components and systems built by composing them. To some extent, Koala allows calculating and predicting resource consumption (e.g., static memory), but it lacks support for managing other extra-functional properties. Compared to ProCom, Koala is geared towards less safety-critical applications.

PECOS [12] is a component model developed conjointly by ABB Corporate Research and academia for development of small reactive embedded systems in automation applications (such as industrial field devices). The PECOS component model supports hierarchical component composition. Similarly to ProSave level, components interact via data ports, and the communication between them is based on the pipe-and-filter paradigm. A PECOS component can be active, passive or an event. Active and event components have their own thread of execution, and passive components cannot control their execution and are used as part of the behavior of another component being executed synchronously. Besides data ports, PECOS components have also interfaces to
express extra-functional properties and constraints. In PECOS, as in ProCom, a strong importance is given to extra-functional properties, and there is possibility to specify component’s meta-data such as worst-case execution times and memory usage, but the techniques differ. The behavior of the components can be modeled with Petri nets.

Pin [49] component model is developed at Carnegie-Mellon University. Its purpose is to be used as a basis for PECTs (Prediction-Enabled Component Technologies), which are concerned with providing predictability principles for the run-time behavior of assemblies of software components, such as performance, safety and security. In order to attain predictability of a given property PECT offers a reasoning framework that includes a component technology powered by analytical interface and analysis theory. Analytical interfaces are used for specification of the properties, which are n-tuples consisting of a name, value and additional property-specific information (e.g., confidence interval of the property value). Analysis theories are used to predict properties of component compositions. At this time PECT supports three reasoning frameworks: \( \lambda_{ABA} \) - for predicting average latency in assemblies with periodic tasks, \( \lambda_{ss} \) - for predicting average latency in stochastic tasks managed by a sporadic server and ComFoRT - for formal verification of temporal safety and liveness. Contrary to ProCom, Pin is not distributed, does not support hierarchical component nesting and does not have support for high-level design.

Robocop [14] component model is a successor of the Koala component model, and is developed out the collaboration between Philips and Eindhoven Technical University. Similar to ProCom, a component is considered as “a whole”, i.e., a collection of models gathering all the information needed and/or specified at different points of time of the development process (e.g., documentation, source code, functional model, resource model, simulation model and execution model). Models may be used as well for depicting extra-functional properties of Robocop components. These extra-functional models can include timeliness, resource consumption, reliability, safety and security. The resource model is based on resource predictions, which can not provide 100% guarantees if compared to formal methods. Therefore, it is not suitable for safety-critical systems. The functionality offered by a component is logically modeled as a set of “services”. Similar to Koala, Robocop is dealing only with static resource consumption, since it is assumed that consumption of resources stays constant per operation of a service.

Rubus [13] is a component model developed in collaboration between Arcticus Systems AB and Mälardalen University, and is intended for development of distributed, resource-constrained, embedded control systems, with a mix of
5.2 Resource Modeling and Analysis

Although, one may think of numerous extra-functional properties crucial for embedded systems, in practice, they often reduce to timing, memory, performance or throughput, and dependability/reliability-related aspects. These aspects may be addressed differently depending on the context or the application domain (e.g., timing aspects have to be more precise for safety-critical systems than for home-appliances). Thus, depending on the context, extra-functional properties can be modeled or built-in at different levels of formality, such as: informal level, which describes extra-functional aspects in natural language; semi-formal, which uses notations such as the UML or even more formal, which describes extra-functional aspects by using much more formal notations such as temporal logics or process algebras. Using to a great extent the research detailed in paper C, this section summarizes the related work on modeling and analyzing resources in component-based real-time embedded systems and compares them with the REMES behavioral model proposed in this thesis. The related work may be grouped into three categories.

First, research has been devoted to predicting code-level resource consumption of component assemblies. In Koala [27] component model, compositional
ways of estimating static memory consumption have been performed for applications in which the instantiated components of a composition are known prior to run-time. The resource information is exposed through a spacial type of component’s interface, called IResource. The interface contains information about different types of memory and a formula for estimating the memory size of each type of memory is added to the IResource implementation. The technique supports budgeting i.e., the expected values of the resource consumption of non-implemented components can be also accounted for. In Robocop [28] component model is presented a scenario-based prediction of run-time resource consumption. Robocop resource model specifies the predicted resource consumption for all operations implemented by the services of an executable component. The resource consumption is given as a number of cost functions. The resources that are claimed and released are specified per operation. Similar to Koala, this method is also dealing with static resource consumption, since it is considered that the consumption of resources is constant per operation. Both of the aforementioned approaches deal with low-level code-driven resource estimates, which can only be used in cases when one has access to the components implementations. However, more abstract descriptions of expected resource usage may be needed for not-yet implemented components, or for guiding the selection of components from the repository. In such cases, the designer could first employ REMES for early resource usage analysis, and then apply the approaches mentioned above.

The second category is represented by the attempts of software modeling languages and profiles (e.g., UML [50], UML/SPT [29] and MARTE [51]) to tackle the modeling and analysis of embedded resources. Amar et al. [30] model resources in UML-based simulative environment. They extend the UML notation with new stereotypes for resources types. In one capsule diagram are gathered the software architecture and the resources that the software components require. As such, the capsule diagram is split in two parts: the software side and the resource side. The resource side is composed by a Main Dispatcher, which is in charge of receiving resource requests from the software side and a set of resource types. Internally every resource type capsule contains an Internal Dispatcher and a set of actual resource instances. The UML profile for Schedulability, Performance and Time (UML/SPT) [29] is a framework for modeling concurrency, resources and timing concepts, which eventually produces models for schedulability and performance analysis. The core of the profile represents the General Resource Modeling framework, which describes resource types (hardware or software) and their management. The UML/SPT profile provides set of stereotypes and tag values that can be used
for annotation of the model elements and for performing analysis. The new profile, MARTE (Modeling and Analysis of Real-Time and Embedded systems) [51], which emerged from the UML/SPT profile is dedicated to complement UML with the required extensions for supporting modeling and analysis of embedded real-time systems. The new profile should address specification of not only real-time constraints but also other embedded extra-functional characteristics such as memory and power consumption and modeling and analysis of component-based architectures. It provides a basic framework for platform-based modeling, the General Resource Modeling (GRM). It is based on a clear design pattern considering platforms as a set of resources containing possible sub-resources in hierarchical manner and offering at least one service. GRM is refined in Software Resource Model and Hardware Resource Model dedicated to describe software and hardware computing platforms, respectively. Although graphical and intuitive, these UML-based approaches are not precise and rigorous, and lack formally founded semantics. They can not entirely guarantee the feasibility of the architecture, but rather give partial answer. In contrast, REMES provides both a graphical behavioral notation, as well as a rigorous underlying framework for formal analysis.

The third category is mainly represented by the higher-level formal approaches [25, 26], proposed by Lee et al. They propose a family of process-algebraic formalisms, developed to unify formal modeling and analysis of embedded systems resources. Their formalisms can theoretically account for various resource types and a resource is considered as a generic, first-class modeling entity. A resource may be characterized by a set of attributes, such as timing parameters, probability of failure, priority, power consumption, etc., which capture the resource’s behavior. The authors take into account sets of resource classes important for embedded real-time systems: serially reusable shared resources, used to model processor units, communication resources, used to model synchronous and asynchronous communication channels, and multi-capacity resources that naturally correspond to memory modules. The framework is theoretically rich, however it is not intuitive and the tool support is not equally mature. Ouimet et al. [31] use timed abstract state machines as a unified formalism to specify functional and extra-functional properties of embedded systems. The resources are described as simple annotations, in the form of real-valued variable assignments. Consequently, the framework can not support trade-off analysis of possibly conflicting resource requirements, which is supported by REMES.
Chapter 6
Conclusions and Future Work

In this thesis we have addressed the design, behavioral modeling and analysis of resource-aware component model for development of distributed embedded systems, vehicular embedded systems in particular. We have exemplified the applicability of the results presented on two small case studies. However, full validation of the research results in more realistic case studies is subject to future work.

6.1 Contributions

The main contributions of the presented research are summarized as follows:

A four dimensional classification framework. In this thesis we present a classification and comparison framework for component models. The classification framework consists of four dimensions (lifecycle, constructs, extra-functional properties and domains) in which the basic characteristics and principles of component models are distinguished.

A two-layered ProCom component model for embedded systems. Comparing with other component models targeting embedded systems, the ProCom component model addresses quality attributes, resource consumption and distribution more systematically.
Chapter 6

Conclusions and Future Work

In this thesis we have addressed the design, behavioral modeling and analysis of resource-aware component model for development of distributed embedded systems, vehicular embedded systems in particular. We have exemplified the applicability of the results presented on two small case studies. However, full validation of the research results in more realistic case studies is subject to future work.

6.1 Contributions

The main contributions of the presented research are summarized as follows:

A four dimensional classification framework. In this thesis we present a classification and comparison framework for component models. The classification framework consists of four dimensions (lifecycle, constructs, extra-functional properties and domains) in which the basic characteristics and principles of component models are distinguished.

A two-layered ProCom component model for embedded systems. Comparing with other component models targeting embedded systems, the ProCom component model addresses quality attributes, resource consumption and distribution more systematically.
An unambiguous and compact description of the modeling elements of ProCom. The description is based on an extension of finite-state machines and sets the ground for formal analysis of systems built out of ProCom elements. The proposed finite-state machine language has graphical appeal, making it simpler than the corresponding timed automata model, and it abstracts from real-valued variables and synchronization channels.

REMES behavioral language. REMES is a dense time state-based hierarchical behavioral language that has a notion of explicit entry- and exit points, continuous variables, flows and progress invariants. It is our intention REMES to be used for unified modeling and formal analysis of functional, timing and resource-wise behavior of embedded systems.

Performing resource-wise analysis. We present a method for encoding the resource-wise analysis problem as a weighted sum in which the variables capture the accumulated consumption of resources, respectively. Thus, we perform three types of analysis: feasibility analysis, optimal or worst-case resource consumption analysis, and trade-off analysis. Feasibility analysis checks whether the accumulated values of the resources consumed/used during all possible system behaviors are within the available resource amounts provided by the implementation platform. Optimal or worst-case resource consumption analysis returns the cost of the “cheapest”, and/or most “expensive” trace that will eventually reach some goal. This analysis may help in resolving the possible non-determinism in a component implementation. Trade-off analysis is a systemic approach to balancing trade-offs between conflicting resource requirements: memory vs. execution time, energy vs. memory, etc. The result of this analysis is the best alternative between the conflicting requirements.

6.2 Future Research Directions

There are many possible future extensions of the work presented in this thesis. The current version of the ProCom component model is primarily targeting vehicular systems and focuses on design of a class of distributed embedded systems that execute real-time controlling tasks. As future work, ProCom component model may be extended for instance to the automation domain. Additionally, ProCom has not yet been industrially verified on an industry case study and we plan to do this as future work.
We have already started with studying the applicability of REMES to other domains related to embedded systems, such as service-oriented systems and programmable logic controllers. In future, we plan to integrate REMES and its notion of resources in the ProCom component model. Here mapping between ProCom ports and REMES entry/exit points should be formally defined. Further, we plan to perform automatization of the process of predicting the resource usage of components and systems. As such, REMES GUI should become part of the ProCom development IDE, that is currently in phase of development. We have performed analysis in UPPAAL CORA, which can currently only handle priced timed automata models where the weighted sum is monotonically increasing. Therefore, it is left for future work to conduct a case-study which tackles feasibility analysis problem for a system in which some of the edge prices are negative so that the global cost function is non-monotonically increasing. In addition, all of the resource-wise verification algorithms presented in this thesis need to be implemented in UPPAAL CORA and the scalability of the approaches should be checked. As future work, we also plan to integrate REMES with the formal semantics of ProCom. Another opportunity for future work is to investigate the compositional reasoning i.e., analyzing each component of the system in isolation and allowing global properties (such as resource consumption of the whole system) to be inferred about the entire system. This of course will leave us the obligation of proving that the component specifications in turn imply the specification of the entire system.


Bibliography


