Mälardalen University Press Licentiate Theses
No. 134

FORMAL APPROACHES TO SERVICE-ORIENTED DESIGN

FROM BEHAVIORAL MODELING TO SERVICE ANALYSIS

Aida Čaušević

2011

MÄLARDALEN UNIVERSITY
SWEDEN
School of Innovation, Design and Engineering

ISBN 978-91-7485-012-3
ISSN 1651-9256
Printed by Mälardalen University, Västerås, Sweden

To the Memory of Maja Đokić

Acknowledgments

I have never thought that the decision to attend "a some guy's" presentation about studies in Sweden, while all my friends went for a coffee break would impact my life this much. During the presentation I found out that "a some guy" is nothing less but well known professor in computer science, Ivica Crnković. Watching his presentation about Mälardalen University and existing research opportunities, with all those attractive photos (probably taken during the warmest and the sunniest day in summer), made me think about possibility to pursue PhD studies and move to Sweden. Few months later, I came to Sweden and started my journey. I cannot express how much I am grateful to him, for believing in me and giving me an opportunity to become a PhD student.

Of course this thesis would not be possible without my supervisors Paul Pettersson and Cristina Seceleanu who have not only served as my supervisors but also have encouraged and challenged me through my studies. I owe a great debt of gratitude for their guidance and for never accepting less than my best efforts.

I would also like to thank to present and some former members of my research group (working on Formal Modeling and Analysis of Embedded Systems) Andres Hessel, Aneta Vulgarakis, Cristina Seceleanu, Eun-Young Kang, Jagadish Suryadevara, Leo Hatvani, Paul Pettersson, and Stefan Björnader for all support, discussions, reviews and comments.

Outside of the thesis work I have been involved in teaching. Many thanks to people that I had pleasure to work with: Ivica Crnković, Frank Lüders, Jan Carlson, Aneta Vulgarakis, Séverine Sentilles, Adnan Čaušević and Andreas Johnsen.

During my studies I have attended a number of courses. I would like to thank to Hans Hansson, Ivica Crnković, Paul Pettersson, Sasikumar Punnekkat, Cristina Seceleanu, Frank Lüders, Gordana Dodig-Crnković,

Eun-Young Kang, Thomas Nolte, and Emma Nehrenheim for giving me knowledge and vision to become a better PhD student.

I would like to thank to the whole administrative staff at the department for making my life easier, in particular Hariet Ekwall, Monica Wasell, Carola Ryttersson, Gunnar Widforss, Susanne Fronnå, and Malin Rosqvist.

Spending time with people from the department made all coffee breaks, lunches, and travels more interesting and enjoyable. I would like to thank to Aleksandar, Ana^{+}, Andreas^{+}, Aneta, Anton, Antonio, Barbara, Batu, Bob, Branka, Cristina, Dag, Damir, Daniel, Etienne, Farhang, Federico, Frank, Fredrik, Giacomo, Hongyu, Hüseyin, Iva, Jagadish, Jan, Josip, Juraj, Lars, Leo, Luka, Luis, Mehrdad, Mikael, Moris, Nikola, Radu, Rafia, Saad, Svetlana, Thomas^{+}, Tibi, Tomas, Rikard, and Séverine. [1]

Furthermore, I thank to my Bosnian friend, Ajla Ćerimagić, for supporting and encouraging me during the last 10 years through thick and thin in life.

To my brother Adnan - thank you for being there for me despite the distance between us.

Veliko hvala mojim roditeljima, Edini i Mujagi. Hvala Vam što ste me naučili svemu što znam, za pruženu bezuvjetnu ljubav. Ja sam ono što jesam zahvaljujući Vama. Znajte da ovaj rad ne bi bio moguć bez svega što ste me naučili do sada. [2]

Finally, my deepest gratitude goes for my dear husband Adnan and daughter Alina. Thank you for bringing sunshine into my life, for being my inspiration, and motivation to continue during those moments when the things did not work well.

Aida Čaušević
Västerås, June, 2011

[1] The positive closure operator is used to express that one or more persons is acknowledged.

[2] I am grateful to my parents, Edina and Mujaga. Thank you for teaching me all I know, for the all unconditional love. You are the reason for which I am here today. This thesis would not be possible without everything you have though me throughout my entire life.

List of Publications

Publications Included in the Licentiate Thesis

Paper A: Aida Čaušević, Paul Pettersson, Cristina Seceleanu. *Analyzing Resource-Usage Impact on Component-Based Systems Performance and Reliability.* Proceedings of International Conference on Innovation in Software Engineering (ISE08), IEEE, Vienna, Austria, December, 2008.

Paper B: Aida Čaušević, Aneta Vulgarakis. *Towards a Unified Behavioral Model for Component-Based and Service-Oriented Systems.* Proceedings of 2nd IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS09), Seattle, USA, July, 2009.

Paper C: Aida Čaušević, Cristina Seceleanu, Paul Pettersson. *Modeling and Reasoning about Service Behaviors and their Compositions.* Proceedings of 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA10); Formal Methods in Model-Driven Development for Service-Oriented and Cloud Computing track, Heraklion, Crete, Greece October 2010.

Paper D: Aida Čaušević, Cristina Seceleanu, Paul Pettersson. *Checking Correctness of Services Modeled as Priced Timed Automata.* Submitted to conference.

Other publications, not included in the thesis

- Aneta Vulgarakis and Aida Čaušević. *Applying REMES behavioral modeling to PLC systems.* Mechatronic Systems, vol 1, nr 1, p40-49, Journal of Faculty Of Electrical Engineering, University Sarajevo, December, 2009.
- Aida Čaušević, Cristina Seceleanu, Paul Pettersson. *Formal reasoning of resource-aware services.* MRTC report ISSN 1404-3041 ISRN MDH-MRTC-245/2010-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, June, 2010

Contents

I

Thesis

Chapter 1

Introduction

It is a known fact that, during the last decade, the complexity of software systems has been continuously increasing. One of the reasons underlying such increased complexity is a new trend that aims to integrate and connect heterogeneous applications and available resources, in many cases on-the-fly. However, most of the existing systems and applications are not designed to offer smooth and easy integration and adaptation to new application scenarios. Additionally, to reduce development time of new systems and applications it became a requirement to facilitate software reusability and componentization. Most of these challenges have already been addressed by the component-based paradigm [1]. However, since component-based approaches offer component reusability and composition only at design time, while on-the-fly behavior is not tackled, it seems only natural that new paradigms and approaches that would deal with such challenges would emerge. The recently introduced paradigm of service-oriented systems (SOS) [2] accommodate the necessary conceptual foundations to cope with increased complexity and challenges related to integration, by advocating the development of autonomous and loosely coupled software entities, called services. Although the approach has brought many benefits, there are still issues to be addressed, such as: service modeling, service compatibility, interoperability between services implemented by different vendors and on different platforms, service composition via service orchestration and choreography, analyzing quality-of-service (QoS), etc. In this thesis, we focus on behavioral modeling of services, formal verification for functional, timing, and resource-

wise correctness, as well as hierarchical modeling through a "command-line" like language.

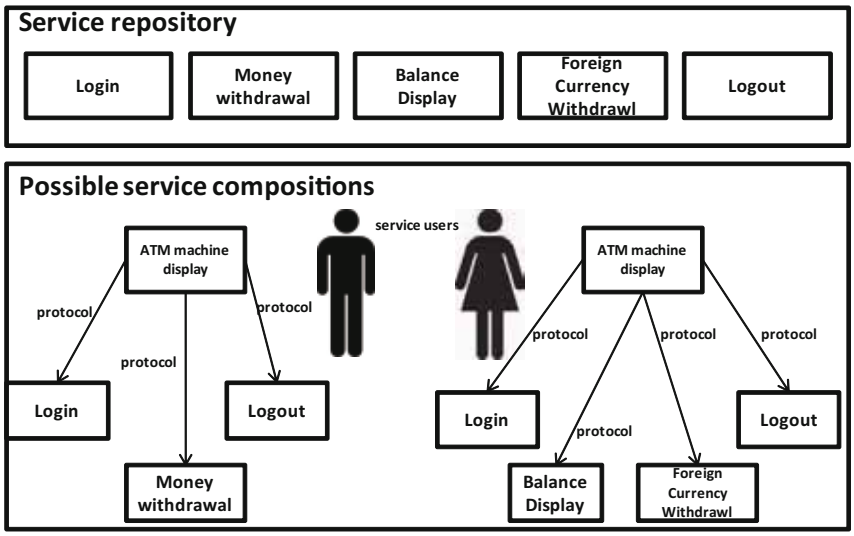

Figure 1.1: Service oriented ATM system

SOS assume services as their basic functional units, independent of any specific implementation platform, capable of being published, invoked, composed, and destroyed on-the-fly. One of the fundamental characteristics of services is separation of interfaces from the service behavioral description. Publicly available service interface information specifies service properties such as service type, capacity, time-to-serve, etc., visible to service users. The latter exploit interface information of available services, to find and invoke services most suitable for their needs. Figure 1.1 depicts a simplified overview of an ATM system. One can notice that the system consists of several services, available to the service user, which can be invoked and composed in different ways, based on the preferences of the user. Now, let us assume a component-based fixed architecture of the same ATM system, as depicted in Figure 1.2 - in such a version, all components are composed in advance, and all connections between components are implemented before the actual system became available to users.

On the other hand, details about service behavior description are

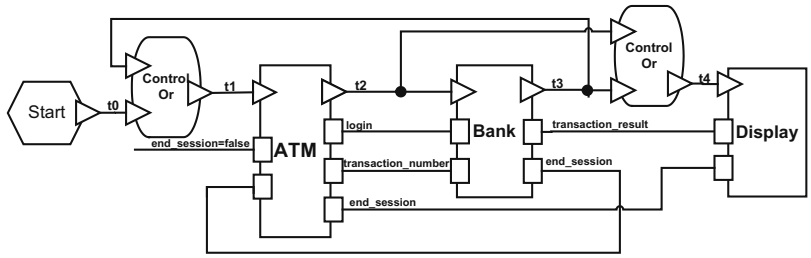

Figure 1.2: Component-based ATM system

normally hidden from service users, but available to service developers. Service behavior description gives a deeper insight into service functionality representation, enabled actions, resource annotations, and possible interactions with other services [3]. Such description may be useful to the service developer that needs to ensure that adding more function or improving a service with respect to some QoS attribute does not alter the correctness of the existing behavior. Also, it becomes important in cases when one has to differentiate between services that deliver the same functionality, but have, for instance different response time or resource usage. The service behavioral description not only enables a proper understanding of a service function/functionality, but also helps to connect services in the correct way, and provides means for rigorous reasoning about extra-functional properties, whose assurance is recognized to be insufficiently addressed.

If one considers the run-time service behavior, then ensuring the expected level of QoS becomes more difficult. QoS encompasses the extra-functional attributes of a service, such as performance, reliability, security, etc., as well as cost-related information. Being aware of QoS in advance, enables easier service composition, reduces the level of uncertainty, and gives a possibility to optimize the newly composed service whenever required. To guarantee the required level of QoS, some of the existing SOS frameworks provide formal analysis techniques for services [4–7]. In most cases, building the formal model to be analyzed is not a straightforward process.

One of the main principles of SOS is the idea of composing services by discovering and invoking them on demand, rather than building the whole application from scratch, at design time. The service composition can be achieved either through orchestration, or choreography. The

former assumes the existence of a central controller responsible with scheduling service execution, according to the user demands, while the latter assumes a mechanism of message exchange between participants in a composition, without requiring a central coordinator.

Because of the dynamic nature of services, it is compulsory that, besides ensuring service correctness in isolation, one checks the functional and extra-functional correctness of possibly composed services, as soon as they are formed. For example, let us assume that we have a service that is composed out of several navigation services, where some services return a route length in miles, and some in kilometers. If the developer has omitted to introduce a service that would convert length from one metrics to the other, one should be able to detect this, by formally checking the correctness of the actual composition, right after it is constructed.

The goal of this thesis is to provide methods and tools for the specification, modeling, and formal analysis of services and service compositions in SOS. Relying on the fact that SOS have similar characteristics with component-based systems (CBS) (e.g., componentization, reusability, composition, etc.), this thesis introduces an extension of the existing behavioral modeling language, called Remes, which has been designed to fit a component-based design (CBD) perspective [8, 9]. Our proposed extensions exploit such advantages of the model, and also introduce service-oriented features, aiming at making Remes suitable to behavioral modeling and analysis of SOS, too. As a first step, we identify commonalities and differences between CBD and SOS, in order to determine the set of extensions to be applied to Remes. Driven by our findings, we next show how services can be formally described by Remes, our resource-aware timed behavioral language, which we extend with service specific information, such as type, capacity, time-to-serve, etc., as well as boolean constraints on inputs, and output guarantees. By exploiting the pre-, and postcondition annotations, we show how to describe the service behavior in Dijkstra's guarded command language [10], and how to check the service correctness by employing Dijkstra's and Scholten's strongest postcondition semantics [11].

Since the original semantics of Remes is given in terms of priced timed automata (PTA), in this thesis we also present an algorithmic way to compute strongest postconditions of services modeled as PTA, which could be completely automated. We consider the service resource consumption in Remes as a cost variable in PTA and, alongside our

strongest postcondition calculation, we include, in our algorithms, well known approaches for computing the minimal and maximal reachability cost [12]. The two ways of computing the strongest postcondition of services modeled in REMES, needed for proving the correctness of service composition, stand complementary. The algorithmic technique can be applied for bounded-variable systems, whereas the deductive technique could be employed in those but also other cases, where the bounds of the variables are not specified, but they range over natural numbers, non-negative reals, etc.

Moreover, to address the on-the-fly aspects of services, we introduce a hierarchical language for dynamic service composition (HDCL) that allows creating new services, as well as adding and/or deleting services from lists. We also give the semantics of sequential, parallel, and parallel with synchronization service composition, respectively.

This work has been carried out within Q-ImPrESS project [13], funded under the European Union's Seventh Framework Programme (FP7), within the ICT Service and Software Architectures, Infrastructures and Engineering priority. The aim of the project is to bring service orientation to critical application domains, such as industrial production control, telecommunication and critical enterprise applications, where guaranteed end-to-end quality of service is particularly important.

To summarize, our main contributions are:

- showing how we can formally check QoS in terms of performance and reliability in formally specified CBS;
- an overview of commonalities and differences between SOS and CBS, which provides insight in the REMES modeling language, limitations, and possible extensions;
- adding constructs to REMES, such that it accommodates formal description of service behavior;
- developing a hierarchical composition language for REMES-based services and defining the semantics of possible service composition operators;
- algorithms for checking the correctness of services modeled in PTA.

The following section provides the background for SOS, and formal analysis, as a foundation for the remainder of the thesis. We close the chapter by giving the thesis overview.

1.1 Preliminaries

1.1.1 Service-oriented Systems

The rapid growth in complexity of the today's software systems is justified by the constant increase in functionality, by higher-level of quality requirements, increase in degree of distribution, mobility, etc. Service-oriented development is one of the most promising approaches that evolved from object-oriented and component-based software engineering concepts, as a solution for the above listed issues. The paradigm relies on two basic principles: (i) modularization, meaning that the overall functionality is split to obtain as smaller and separate as possible units of behavior, called services; and (ii) composition, that is, a way to efficiently, and possibly with lower costs obtain more complex systems out of existing units of behavior.

The literature provides many informal definitions for the term "software service", inspired mainly by the telecommunication domain. A popular definition is given by Broy et al. [14]:

> *A software service is a set of functions provided by a (server) software or system to a client software or system, usually accessible through an application programming interface.*

In SOS, services are the smallest functional units, independent of implementation platform, and equipped with constructs that allow them to be published, discovered, invoked, and if needed, destroyed on-the-fly. In each service, there exists a clear separation, at the model level, between its interface and its behavioral description. Publicly available interface information specifies service relevant information, such as time-to-serve, service capacity, service pre-, and postconditions, etc., such that an available service becomes visible to potential service users. On the other hand, internal behavior-related information, i.e., functionality representation, enabled actions, resource annotation, etc., is hidden from the service user, but available to service developers. In this way, upon request, a service may be easily changed and upgraded to fit with newly given user requirements.

One may say that SOS offer cost-efficient software development by reusing functionality from available services. Also, a service becomes a single point of maintenance for a common functionality. Using discovery

mechanisms, developers can find and take advantage out of existing services, significantly reducing time to develop new systems. Also, in case the QoS of a service is guaranteed, the quality assurance of the new system also increases, and its verification requires a lower effort. Services can be seen as adaptable units, thanks to the clear separation between service interface and service behavior, making it possible to employ incremental deployment of services.

The price to pay for all the mentioned benefits brought by the service-oriented paradigm is a list of challenges in the design and analysis. It still remains a challenging task to predict QoS, since the system's QoS is not a function of the QoS of the services only. It also involves interdependencies between services, resource constraints of the environment, and network capabilities. Additionally, checking the correctness of service compositions lacks appropriate methods and tools especially for extra-functional properties like resource-wise behavior.
Nowdays a number of service-oriented approaches exist [4–6, 15–17]. All of them have the basic service-oriented concepts incorporated like discovery mechanisms, support for orchestration and choreography, some predictability for service performance, reliability, etc., but only few can deliver the whole process from creating single service to system development, including some means for analysis. It is obvious, that this paradigm of SOS still remains to be fully explored, developed, and utilized.

1.2 REMES: A Resource Model for embedded Systems

To address functional and extra-functional behavior such as timing and resource consumption, we use a dense-time state-based hierarchical modeling language called REMES [18].

The internal component behavior in REMES is depicted by REMES modes that can be either *atomic* (do not contain submode(s), see Atomic mode 1, Atomic mode 2 in Figure 1.3), or *composite* (contain submode(s)). The data transfer between modes is done through the *data interface*, while the control is passed via the *control interface* (i.e., entry and exit points). REMES assumes *local* or *global* variables that can be of types boolean, natural, integer, array, or clock (continuous variable evolving at rate 1).

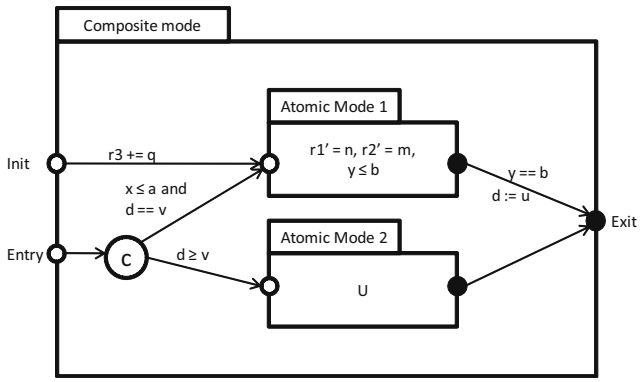

Figure 1.3: A REMES mode

A composite mode executes by performing a sequence of discrete steps, via actions that, once executed, pass the control from the current submode to a different submode. An action, $\mathsf{A} = (\mathsf{g}, \mathsf{S})$ (e.g., $(\mathsf{y} == \mathsf{b}, \mathsf{d} := \mathsf{u})$ in the figure), is a statement S (in our case d := u), preceded by a boolean condition, the guard $(\mathsf{y} == \mathsf{b})$, which must hold in order for the action to be executed and the corresponding outgoing edge taken. A REMES composite mode may contain conditional connectors (decorated with letter C) that allow a possibly nondeterministic selection of one discrete outgoing action to execute, out of many possible ones. In Figure 1.3, via C, one of the empty statement actions, $\mathsf{x} \leq \mathsf{a} \wedge \mathsf{d} == \mathsf{v}$ or $\mathsf{d} \geq \mathsf{v}$ can be chosen for execution.

In REMES one may model timed behavior and resource consumption. Timed behavior is modeled by global continuous variables of specialized type *clock* evolving at rate 1 (x, y in Figure 1.3). Modes may also be annotated with *invariants* (e.g., $\mathsf{y} \leq \mathsf{b}in\mathsf{Atomicmode1}$), which bound from above the current mode's delay/execution time. Once the invariant stops to hold, the current mode is exited. In case a mode is exited instantaneously after its activation, the mode is called *urgent* (decorated with letter U).

Each (sub)mode can be annotated with the corresponding continuous resource usage, if any, modeled by the first derivative of the real-valued variables that denote resources, and which evolve at positive integer rates (r1 and r2 in Figure 1.3). Discrete resources are allocated through updates, e.g., $\mathsf{r3} \mathrel{+}= \mathsf{q}$.

To enable formal analysis, REMES models can be semantically transformed into timed automata (TA) [19], or PTA [20], depending on the analysis goals.

The REMES language benefits from a set of tools[1] for modeling, simulation and transformation into TA and PTA, which could assist the designer during system development. For a more thorough description of the REMES model, we refer the reader to [18].

1.3 Formal Modeling and Analysis of Software Systems

Formal methods are mathematical techniques, often supported by tools, which enable rigorous analysis of systems design, described as well-formed statements in a mathematically precise way. Formal verification is a technique that provides means to prove or disprove the system model's correctness with respect to a formally specified property. This means that, by formally verifying a system model, one checks that the latter indeed behaves according to the specified property. As a result of formal analysis conducted using formal verification, one can get either qualitative answers (yes/no), of quantitative analysis results (numbers). The former, is a result of verification of properties that can be either satisfied, or not. The latter, in our case, represents the minimum/maximum value of the accumulated resource usage for reaching a given goal, but in a more general context, it could mean reliability estimates, performance estimates, etc.

Formal verification assumes the following steps:

- Formally model the system;
- Formalize the property to be checked;
- Prove that the model satisfies the property.

Since the services in SOS are assumed to be invoked, composed, and destroyed on-the-fly, and a designer of such systems is in need to have available methods and tools that support modeling and verification of of the system behavior, as soon as it is constructed, we have chosen the framework of TA and PTA as our modeling framework, and the UPPAAL

[1] The REMES tool-chain is available at `http://www.fer.hr/dices/remes-ide`.

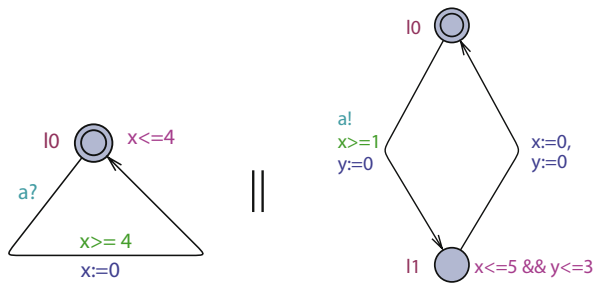

Figure 1.4: A timed automata

-based tools [2] as the model-checkers for verifying the system's property specified in Timed Computation Tree Logic (TCTL) [21], an extension of Computation Tree Logic(CTL) [22] with clocks.

In the following, we briefly describe the models of TA [19] and PTA [23, 24], an extension of TA with prices on both location and edges. Next, the reader is briefed on the model-checking analysis technique.

1.3.1 Timed Automata

A timed automaton [19] is a finite-state machine enriched with a set of clocks. All clocks in one system are synchronized and assumed to be real-valued variables, measuring the time elapsed between events. Consider the TA of Figure 1.4 b). It consists of 2 locations (l_0, l_1), where one of the locations is marked as initial (l_0). Control locations are connected via edges. Real-valued clocks x and y, initially set to zero, evolve continuously at the rate 1. A control node is labeled with a condition on the clock values (the invariant), which defines the maximum allowed time to be spent in a corresponding location. The TA in Figure 1.4 a) may stay in location l_0 as long as the invariant $x \leq 4$ is satisfied. The edges of TA may be decorated with boolean conditions (called guards) on the clock values, which must hold in order for an edge to be taken. (i.e. the edge from l_0 to l_1 will be enabled only if $x \geq 1$ holds). Additionally, edges may be labeled with simple assignments reseting clocks. For example,

[2] For more information about the UPPAAL tool, visit the web page www.uppaal.org.

when following the edge from l_1 to l_0 both clocks x and y are reset to 0.

The semantics of TA is defined as a timed transition system, where each state consists of the current location and the current values of the clocks. The transitions between states may be either delay transitions that model the passage of time, or a discrete transitions that correspond to following an enabled edge in the TA syntactic representation, and result in changing the current TA location.

Systems modeled as a finite set of automata executed in parallel for a given synchronization function represent networks of TA. Automata in Figure 1.4 synchronize on complementary actions via channel a (i.e., $a?$ is complementary to $a!$).

Uppaal is a tool-set for validation and verification of TA models, which serve as the tool input. The Uppaal model checker supports verification of temporal properties, including safety and liveness properties, specified in a decidable sub-set of TCTL. The tool is equipped with a simulator, useful to visualize counter examples produced by the model checker, but also to spot out possible model errors before embarking upon full formal verification. The Uppaal TA extend original TA with the notions of bounded integer variables, binary, and broadcast channels, and urgent and committed locations.

1.3.2 Priced Timed Automata

Priced timed automata are timed automata decorated with costs on both locations and edges. The cost that annotates an active location represents the cost of a delay transition and it is the product of the duration of the delay and the cost rate of the active location. On the other hand, the cost that annotates an edge represents the cost of the discrete transition and it is given by the cost of the edge. Each run in PTA has a global cost, which is the accumulated price along the run of every delay and discrete transition. In this thesis, we use the framework of PTA for the formal analysis of resource usage in services and service compositions.

Let us assume that the PTA in Figure 1.5 is a clock that periodically synchronizes (every 4 time units, which represents the clock period) with another PTA via channel a. Moreover, we assume that the periodic synchronization uses a certain amount of energy, modeled here as the cost variable *cost*, which evolves at rate 2. The special variable *cost* is increased by the price per time unit for staying in the location l_0

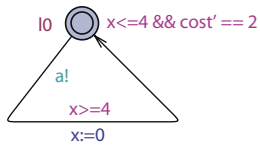

Figure 1.5: A priced timed automaton

($cost' == 2$ indicates that the energy consumption is 2 units per time unit in location l_0).

1.3.3 Model-checking technique

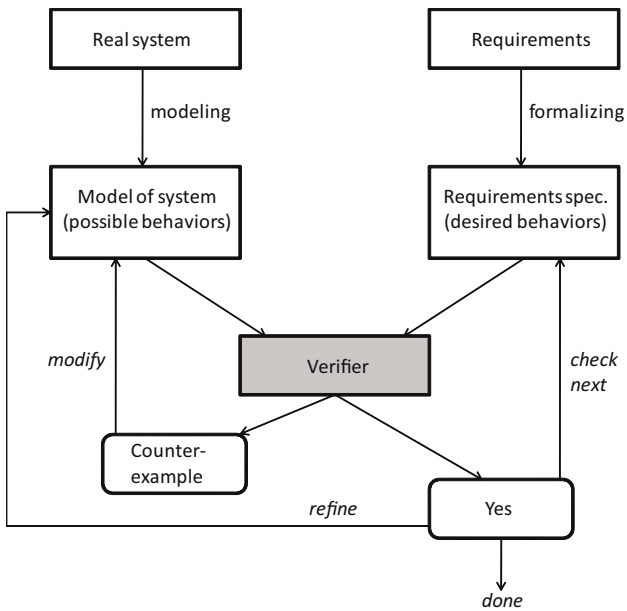

Figure 1.6: Verification methodology of model checking [25]

Nowdays, one of the most used and best known formal techniques is model-checking. The crux of model-checking is its ability to automatically verify finite-state system properties for all possible system behav-

iors. The properties to be examined have to be precisely and unambiguously defined. Being completely automatic and capable to detect counterexamples, model-checking is also suited to uncover and correct errors, in case a given model fails to satisfy the specified requirement. The benefit of model-checking is the possibility to modify the system model, in case that counterexample is detected. On the other hand, even if the system's desired behavior is satisfied, one can refine the model and reapply model checking. Figure 1.6 depicts a generic example of model-checking and includes all steps that the technique follows.

The properties to be examined can be specified using CTL [22]. CTL is a specification language for finite state systems that enable reasoning about sequences of events. The model-checking problem reduces to checking that for a given model M, initial state $\mathsf{s} \in \mathsf{S}$, where S is the set of all model states, and CTL-formula ϕ, $M, s \models \phi$ is satisfied.

1.4 Thesis Overview

This thesis is organized in two distinctive parts. The first part gives a summary of the performed research. Chapter 1 describes the background and motivation of the research. Chapter 2 formulates the main research goal, introduces the research questions, and the research method that we use. Chapter 3 describes the research results and recapitulates the research questions. Chapter 4 surveys related work. Finally, Chapter 5 concludes the thesis, summarizes the contributions and outlines future work that that can be seen as guidelines for future PhD studies.

The second part consists of a collection of peer-reviewed conference, and workshop papers, presented below, contributing to the research results.

Paper A. "Analyzing Resource-Usage Impact on Component-Based Systems Performance and Reliability". Aida Čaušević, Paul Pettersson, Cristina Seceleanu. Proceedings of International Conference on Innovation in Software Engineering - ISE08, IEEE, Vienna, Austria, December, 2008.

Summary: In this paper, we briefly review several popular component models and underlying approaches for analyzing the dependency between resource consumption, performance and reliability attributes,

and discuss their potential to support performance and reliability analysis. We have also showed how formal verification techniques, in our case model-checking, can efficiently be used to predict the performance and reliability of a small real-time, distributed system, modeled as a network of priced timed automata.

Contribution: This paper was written with equal contribution from all three authors. I have been responsible to collect relevant information about chosen component models and underlying approaches for analyzing resource consumption, performance and reliability attributes, and to model a real-time multiprocessor system in UPPAAL CORA, which acts as the example in the paper.

Paper B. "Towards a Unified Behavioral Model for Component-Based and Service-Oriented Systems". Aida Čaušević, Aneta Vulgarakis. Proceedings of 2nd IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS), Seattle, USA, July, 2009.

Summary: This paper overviews the general characteristics of both SOS and CBS, pointing out the similarities and differences between them. We show how an existing component framework could be effectively used to model and analyze SOS constituent services. We assume the existing model REMES as being the underlying model of modeling of functional and extra-functional behavior of services, as well as their interface assumptions and guarantees. For this to become applicable, we first identify the ceratin specific constructs that we need to equip REMES with, such that our goal is achieved. The benefit of REMES language is that it is abstract enough and ready to use even when no detailed system description exists. The modeling part includes also resource annotations on corresponding transitions and modes. Via transformation to PTA, one can conduct rigorous, formal analysis on REMES models . It also benefits from a recently implemented tool-chain for simulation and automatic transformation into PTA. The paper's small case-study is used to illustrate the modeling process within REMES.

Contribution: This paper was written with equal contribution from all the authors. My responsibility has been related to the description of SOS, identifying their characteristics, and the necessary concepts that

would be needed for SOS modeling in behavioral language called Remes . With Aneta Vulgarakis I have shared responsibility for modeling an illustrative example of ATM machine in Remes.

Paper C. "Modeling and Reasoning about Service Behaviors and their Compositions". Aida Čaušević, Cristina Seceleanu, Paul Pettersson. 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA); Formal Methods in Model-Driven Development for Service-Oriented and Cloud Computing track.

Summary: In this paper, we have first extended Remes with specific service attributes deemed useful for service discovery, and we have also semantically defined the composition of Remes services. In Remes, the smallest unit used to represent a single service, is a mode. The notion of mode is extended with attributes such as: service type, service capacity, time-to-serve, service status, service pre-, and postcondition. When all these attributes are published, a service becomes visible and ready to be composed with other services to achieve the given user requirement. To provide means for service composition, and decomposition, the paper proposes a hierarchical dynamic service composition language. The language facilitates modeling of sequential, parallel or synchronized services. It takes into account the services to be composed, type of binding between them and requirement given by the service user. For a small case study described in this paper, we show the service composition correctness checking by manually calculating the strongest postcondition for a program expressed in terms of guarded commands language.

Contribution: This paper was written as equal contribution of all the authors. I have particulary worked on the development of the hierarchical language for dynamic service composition and specified, modeled in Remes, and analyzed the correctness of service compositions for an autonomous shuttle system presented as the example in the paper.

Paper D. "Checking Correctness and Refinement of Services Modeled as Priced Timed Automata". Aida Čaušević, Paul Pettersson, Cristina Seceleanu. Submitted to conference.

Summary: In this paper, we introduce an algorithmic way to check the correctness of services formally defined as PTA by employing forward

analysis technique that assumes computation of the strongest postcondition of automata, with respect to a given precondition. Our algorithms are inspired by already existing approaches for computing the minimal and maximal reachability cost [12]. We show that proving the correctness of a services reduces to showing that the calculated strongest postcondition and minimum/maximum cost of resource consumption implies a requirement defined by a user. The approach is demonstrated in a small accompanying example. Also, we illustrate resource consumption calculation using priced zones for a service modeled in the example.

Contribution: I was the main driver and principal author of this paper. I have contributed with developing algorithms for checking the correctness of services. All the coauthors have contributed with valuable discussions and reviews.

Chapter 2

Research Summary

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

The research presented in the thesis is conducted in the area of service-oriented development, and it has been driven by problems coming from the domain of SOS. The list includes issues such as increase in complexity, composition, resource limitations, and formal analysis of such systems.

An important challenge is thus to develop appropriate methods and languages to model, compose, and formally analyze behavior of services in SOS. Motivated by the need for solutions, the main goal that this thesis aims at addressing is the following:

> *Provide methods for specification, modeling, and formal analysis of services and service compositions in SOS.*

The goal is broad and admits various answers. We approach the goal by answering to five research questions and two subquestions, as formulated in the next section.

2.2 Research Questions

Research question 1.

The clear distinction between SOS and CBS is not completely established. Based on several similar characteristics, one could consider that SOS evolved from CBS. However, despite numerous similarities between SOS and CBS, in order to understand SOS in a proper way, one needs to be aware of the differences between the two, as well. Due to many similar concepts that SOS and CBS rely on [26, 27], we have assumed that it could be beneficial to use a unified behavioral model for both paradigms. Under such assumption, rather than embarking upon the development of a new service-oriented modeling environment, we have chosen to extend an already existing CBD-fit model towards making it suitable for SOS, too.

For this purpose, we have identified the behavioral language Remes [8] as a possible candidate for describing, modeling, and analyzing SOS, for three main reasons: i) it is already developed for CBS modeling, ii) it is suitable for describing both functional and resource-wise behavior of components, iii) and has precise semantics. Since resource-aware timed behavioral language Remes is aimed at distributed embedded systems for which the architecture is usually fixed at design-time, the detailed investigation of its suitability in SOS is needed. In terms of analysis, our focus has been on extra-functional behavior, especially optimal resource-usage of various types of resources, such as, memory, energy, etc. In the light of our exposed overall research goal, and of the motivation outlined above, we have first tried to answer the following research questions:

> *What are the characteristics, advantages and limitations of existing component-based frameworks with respect to analysis of extra-functional behavior like system's resource-usage?*
>
> *(Q1A)*

> *How do such models differ from the service-oriented ones?*
>
> *(Q1B)*

Research questions 2. and 3.

To understand the ways in which services behave and provide meaningful analysis of SOS, we have to be able to access a detailed behavioral description of each service. Most approaches that are dealing with SOS usually end up at service interfaces level, not describing the underlying service behavior [28, 29]. Our aim has been to provide service behavior description in REMES, where by service behavior we mean internal state change for each specific entity of the service architecture, needed for properly understanding of the whole SOS. To meet the target that we have just described, we need to answer the questions below:

> *What are the relevant features of SOS that need to be supported by* REMES *and its analysis methods?*
>
> *(Q2)*

> *How to model services such that they could be discovered and reasoned about?*
>
> *(Q3)*

Research question 4.

One of the growing trends of software engineering is building platform independent software services. Unlike components in CBS that are composed at design-time, in SOS services are assumed to be published, invoked, composed, and destroyed on-the-fly. They are more loosely coupled and more independent of implementation specific attributes than components are. Furthermore, there is a need to enable complex application creation based on given requirements. This means that the user, or developer can create new systems out of existing services, on the spot, and this in turn requires the newly composed system to comply to a desired QoS. If this is not the case, then one should be able to replace services that contribute to the violation of required QoS with ones that could ensure the system quality. When the user ceases to need it, the corresponding service composition should be destroyed, and unnecessary services shut off. Accordingly, the next questions need to be answered:

> *How to compose services on-the-fly and formally analyze the resulting composition in terms of functional and extra-functional correctness?*
>
> *(Q4A)*

How to model hierarchically built services, and represent the main operations on services in a programming-like language?

(Q4B)

Research question 5.
Since services can be composed on-the-fly besides verifying the correctness of the constituent services in isolation, we need to perform verification of the composition as soon as it is built. We are interested in proving that the given composition provides the intended/required functionality, while possibly using as efficiently as possible the involved computing resources. We need to answer the following question, in order to solve the problem that we have just described:

How to ensure the correctness of services?

(Q5)

2.3 Research Methodology

In order to adequately answer the research questions, it is important to adopt an appropriate research methodology, suitable for a given setting. The methodology used in our research is based on the research steps proposed/described by Shaw [30]. It includes the following:

1. Identification of the research problem based on current trends and demands from the SOS community.
2. Transferring the problem to a research setting and defining the research questions.
3. Analysis of the current state-of-the-art based on the defined research questions.
4. Answering the research questions by presenting the achieved research results.
5. Research results illustration. The goal is to show that the defined research questions have been properly answered. It can be achieved by performing case studies, giving a formal proof, or by prototype implementation.

6. Validating whether the research results can be applied in the real-world applications.

Based on these steps, in our research we have first defined the initial problem, as stated in Chapter 2. The problem definition has been followed by identification of research questions as also presented in Chapter 2. In the next step, we have conducted a state-of-the-art investigation, which has resulted in writing paper A. Further, in papers B, C, and D we have presented our research results which are summarized in Chapter 3.

The thorough validation of the presented results is missing, and is the subject of the future work to be done through the rest of PhD studies. However, all research results have been exemplified as shown in papers A, B, C, and D. In paper A, we have shown how formal verification techniques can be used to predict the performance and reliability of a small real-time, distributed system. Further, in paper B, we have illustrated the modeling process within REMES on a simple ATM system. The approach presented in paper C, has been demonstrated on an adapted version of an intelligent shuttle system, for which we have computed resource consumptions, and showed energy-time trade-off analysis. Paper D includes an illustrative example of our proposed approach, presented in the paper.

Chapter 3

Research contributions

This chapter provides a brief overview of our contributions and research results with respect to the research questions proposed in Chapter 2. The details are presented in the appended papers, to be found in the second part of this thesis.

3.1 Component-based vs. Service-Oriented Systems: System Modeling and Analysis

Goal: Based on a considerable number of similarities between SOS and CBS, it is assumed that SOS have evolved from CBS. These two paradigms share many of the main concepts and principles, both are focused on modularization and composition, and both proclaim software reusability and rapid system development. However, one has to also be aware of differences that exist in mechanisms, approaches, and implementations, of the two paradigms. The goal of this research is to conduct an investigation on characteristics, advantages, and limitations of existing component-based frameworks. The result of such investigation can help to better understand the background of such frameworks, and be able to distinguish them from the service-oriented ones. Furthermore, based on the comparison, one can extract a list of relevant features that need to be supported by component-based frameworks.

Results: The result of this research is an analysis conducted on several popular component-based frameworks including: Klaper, Palladio, SOFA, and BIP, in terms of identifying their capability of modeling extra-functional properties, with a focus on performance and reliability. Here, by performance, we mean performance metrics such as response time, throughput, completion time, etc., and by reliability the ability of a system or component to perform its required functions under stated conditions for a specified period of time. We have noticed that some of them are specialized on analyzing specific extra-functional properties depending on the area in which these frameworks are used. We have carried out comparisons between such approaches, and a recently introduced framework for component-based design, called ProCom, and its behavioral language REMES, on which we rely our subsequent research. The comparison highlights similarities and differences between our and the assumed frameworks, paving the way towards extending REMES with the necessary constructs, needed for the language to become fit for service-oriented development. Detailed results can be found in papers A and B.

Limitations and future work: The conducted investigation selects and compares only several popular approaches and it can be always extended to other component models. Moreover, the provided analysis is limited to only performance and reliability as extra-functional properties of interest. In the future it might be of interest to expand the analysis to more frameworks focusing on other extra-functional properties, too.

3.2 Formal Modeling of Resource-aware Service Behaviors in REMES

Goal: Relying on the fact that, in most cases the development of SOS uses platform-independent services, there is a need for rigorous analysis of such systems already at design time. Additionally, some systems have limited available resources that makes the development process more strict and demanding. Also, since services are platform independent and loosely coupled it is possible to compose them in more than one way, usually on-the-fly. In these cases, even if the service behavioral description is available, becomes beneficial to reduce service composition analysis to checking that could be performed based on the information supplied

in the service pre-, and postcondition. Due to many similar characteristics between CBS and SOS, we have decided to extend the recently introduced resource-aware timed behavioral language REMES, initially developed for CBS, with necessary constructs to support SOS. Our goal is to propose a model that relies on precise semantics, to be used as a basis for the formal modeling and that comprises both formal modeling and analysis of SOS.

Results: The result of this research is a service-oriented extension of the resource-aware behavioral language REMES. In our work, we have defined the service interface, such that a service could be published and visible to service users. This extension relies on the work described previously, in which we have identified such necessary SOS features. Our service interface is modeled to include information about the service type, time-to-serve, service status, service pre-, and postcondition. The latter specify the set of initial conditions to be fulfilled by the service in order to be executed, as the precondition, and the guaranteed result of operation, possibly including extra-functional information like timing and resource-usage, as the service post-condition. A REMES service can be atomic, composite, but also employed in various types of compositions, resulting in new, more complex, services. There are cases in which these subservices need to be composed sequentially, in parallel, or need to be synchronized. In order to model the synchronized behavior of services we have introduced a special kind of REMES mode (the smallest functional unit in REMES), called AND/OR mode. By the semantics of the mode, in an AND or an OR mode, the services finish their execution simultaneously, from an external observers point of view. However, if the mode is employed as an AND mode, the subservices are entered at the same time, and their incoming edges do not contain guard, while an OR mode assumes that one or all subservices are entered based the guards annotated on the incoming edges. In order to support on-the-fly service manipulation, we have enriched REMES with interface operations such as: create service, delete service, replace service, etc. Alongside the above operations, we have defined a hierarchical language that supports dynamic REMES service composition (HDCL), and facilitates modeling of nested sequential, parallel or synchronized services. Originally, REMES can be semantically translated to TA or PTA, depending on the expected outcome of the analysis (i.e., results w.r.t. timing properties, resource consumption, etc.), for formal analysis purposes. However, in

this work, we have relied on a guarded-command language description of a REMES service, and on the associated strongest postcondition semantics [11]. All details regarding this contributions are available in paper C.

Limitations and future work: As a result of our extension, REMES language supports modeling and analysis of SOS. In our work, we do not take into consideration dynamic resource usage (i.e., dynamic memory allocation). It is sometimes the case that a particular service needs to be replaced by a service that delivers better QoS but similar functionality. It is desirable, therefore, that, before the actual replacement, one verifies a refinement relation between services, to ensure that the previous service properties are preserved. Our plan is to investigate possibilities for proving refinement relation between services modeled as REMES modes. Regarding tool support, there exists a stabile eclipse-based implementation of REMES tool chain. However, we plan to provide a stand-alone implementation of modeling services in REMES, paired with means for automated analysis.

3.3 Checking the correctness of REMES services

Goal: Developing systems on-the-fly, by using services equipped with constructs that support online behavior, raises some concerns regarding the quality and correctness of the employed services. As the case with most of the CBS, it is not sufficient to check the correctness of single services, but also be able to verify the functional and extra-functional correctness of service compositions. Considering the fact that some SOS could be embedded into larger systems that need to run on limited resources, it becomes an essential demand to ensure that the system's resource-usage is kept within existing bounds. To address such requests already at early design stages, one needs powerful analysis techniques that encompass both functional but also extra-functional service behavior.

Results: In our approach, we have decided to use, as our verification approach, the forward analysis technique that assumes computation of the

strongest postcondition of a REMES service with respect to a given precondition. To prove the correctness of a REMES service in isolation, we check the boolean implication between the calculated strongest postcondition and the given requirement, reducing verification to a simple proof. We have proposed two techniques for strongest postcondition calculation for services: a deductive one, starting from the guarded command language (GCL) [10] description of a REMES service [31], and an algorithmic one, starting from the PTA description of a service. The latter includes also the minimum/maximum resource-usage trace computation, while performing strongest postcondition analysis. To accomplish the service composition correctness check, we have introduced a hierarchical language for on-the-fly service composition (HDCL) that allows creating new services, by composing existing services via binary operators, as well as adding and/or deleting services from lists. We also give the semantics of sequential, parallel, and parallel with synchronization service composition, respectively. The benefit of this language is that, after each composition, we require that one checks whether the given requirement is satisfied, by forward analysis, e.g., by calculating the strongest postcondition of a given composition w.r.t. a given precondition. The details about this research are incorporated in papers C and D.

Limitations and future work: Both presented approaches would be limited to less complex systems and service, unless a postcondition calculator would be provided in REMES IDE [32] or algorithms would be implemented in UPPAAL CORA [20]. Also, both approaches are illustrated on simple examples. It would be beneficial for our research to model more complex examples, connected to real-world applications. Our intention is to extend the REMES tool-chain with a postcondition calculator and to apply our approach on a series of complex systems, in order to get better knowledge about its weaknesses and limitations.

3.4 Questions Revisited

In this section, we show how the research results and included papers answer the research questions.

Question *Q1A: What are the characteristics, advantages and limitations of existing component-based frameworks with respect to analysis of extra-functional behavior like system's resource-usage?*

Question *Q1B: How do such models differ from the service-oriented ones?*

From the research summary, we can see that these questions are answered by the first research topic and papers A and B. These papers provide a comparison between several selected component-based approaches in terms of analysis of extra-functional properties and highlight differences between component-based and service-oriented frameworks.

Question *Q2: What are the relevant features of SOS that need to be supported by* Remes *and its analysis methods?*

Question *Q3: How to model services such that they could easily be discovered and reasoned about?*

The second research topic and included papers B and C contribute with answers to these questions. It is our intention to provide constructs for modeling and reasoning about services in Remes and we have fulfilled this goal as presented in the mentioned papers

Question *Q4A: How to compose services on-the-fly and formally analyze the resulting composition in terms of functional and extra-functional correctness?*

Question *Q4B: How to model hierarchically built services, and represent the main operations on services in a programming-like language?*

The second research topic gives answers to these questions. In paper C we present mechanisms that enable a service composition out of existing ones and formal analysis w.r.t. the function and resource-usage of composed services.

Question *Q5: How to ensure the correctness of services?*

The third research topic and papers C and D address this question. While in paper C we show how to describe service behavior in Dijkstra's guarded command language, and how to check the service correctness by employing Dijkstra's and Scholten's strongest postcondition semantics, in paper D we present algorithmic computation of strongest postcondition for a service formally described as PTA.

Chapter 4

Related Work

This chapter relates the work in this thesis to relevant research areas. It is subdivided into a number of sections in which we provide comparisons with work of fellow researchers, for each area, respectively.

4.1 Services vs. Components

Broy et al. [14] view a service as a way of orchestrating interactions among a subset of components in order to obtain some required functionality. They assume services as coordinators of component interplay that leads to accomplishing a given task. Masek et al. emphasize that the main difference between services and components is the way the composition is established [33]. While services are composed at run-time, components depend on pre run-time composition. However, their strong similarity in basic concepts and principles makes services and components highly interoperable. Rychlý describes a service as a system that consists of components whose external interfaces match the provided interfaces of the service [16]. In our approach, we regard the notion of a service as an extension of an an already existing component notion. Although it is possible to still ensure the service-component interoperability, our main concern is however, on how to establish service functional and extra-functional guarantees described through pre-, and postconditions at service interface.

4.2 Service-oriented Frameworks

The behavioral side of service engineering is lagging behind the architectural one, a great deal. However, based on the level of details that are provided through the existing behavioral description, all approaches related to services and SOS can be in principle divided into three groups.

The first group is made of code-level behavioral description approaches, in most cases relying on the XML language (e.g., BPEL, BPEL4WS, WS-CDL, etc.). BPEL [4] is an orchestration language whose behavioral description includes a sequence of project activities, correlation of messages and process instances, and recovery behavior in case of failures and exceptional conditions. Approaches like BPEL are useful when services are intended to serve a particular model, or when the access to the service implementation exists. The drawback of such approaches is the lack of formal analysis support, which forces the designer/developer to master not only the specification and modeling processes, but also the techniques for translating models into a suitable analysis environment.

When compared to the above group, BPMN [6] can be seen as a higher-level language. It relies on a process-oriented approach, and supports a graphical representation to be used by both designers and analysts. The lack of a formal behavioral description does not provide means for detailed analysis, as the one supported by Remes. SRML [34] is a service modeling framework that relies on UML state machines to model service behavior, which could help to spread its use among researchers. The benefit of the approach comes with the mechanism that supports the formal analysis formal analysis of functional and timing properties via model-checking; however, the analysis of extra-functional properties, other than timing, is not addressed.

The third group includes approaches with strong formal basis. Rychlý describes the service behavior as a component-based system for dynamic architectures [16]. The specification of services, their behavior, and hierarchical composition are formalized within the π-calculus. Similar to our approach, this work emphasizes the behavior in terms of interfaces, (sub)service communication, and bindings, yet we can also cater for service descriptions including timing and resource annotations [35]. Broy et al. present a theoretical setting of mathematical model of a component model and service mathematical model [14]. The authors provide details on a service behavior in terms of a partial behavior, in comparison to components that are assumed to be described by total behaviors. Al-

though, the work provides a rich theoretical and formal foundation, the approach lacks corresponding automated analysis techniques.

4.3 Checking Properties of Services and their Compositions

A comprehensive survey on several approaches that are accommodating service composition [4–7] is given by Beek et al. [36]. Regarding service modeling, all these approaches are solid; however, w.r.t. service composition [37–39] (usually by employing formal methods), such approaches show limited capabilities to automatically support these processes. Compositions of REMES models can be mechanically reasoned about (although, as for now, we still miss the interface correctness tool support), or can be automatically translated to TA [19] or PTA [20], and analyzed with UPPAAL , or UPPAAL CORA tools, for functional but also extra-functional behaviors (timing and resource-wise behaviors). Foster et al. present an approach for modeling and analysis of web service compositions [17]. The approach takes BPEL4WS service specification and translates it into Finite State Processes (FSP), and Labeled Transition Systems (LTS), for analysis purposes. The drawback of the approach might be too tedious transformation process while acquiring the analysis model, especially in cases when the user is not familiar with different notations and approaches required in this process.

Díaz et al. describe a process of automatic translation of BPEL and WS-CDL service models to timed automata in order to provide means for analysis via UPPAAL model checker [37]. However, the described approach is limited to checking only service timing properties. Narayanan et al. show how semantics of OWL-S, described using first-order logic, can be translated to Petri-nets and then analyzed as such [38]. The analysis includes reachability and liveness properties and checking if the given service or service compositions are deadlock free. Compared to our approach, REMES services can be both mechanically [31] and algorithmically reasoned about. Moreover, REMES services described as TA or PTA can be analyzed with UPPAAL , or UPPAAL CORA tools, for functional but also extra-functional behaviors.

Chapter 5

Conclusions and Future Work

The goal of the research presented in this thesis is to develop methods and tools for the specification, modeling, and formal analysis of services and service compositions in SOS. Mostly, we have focused on the behavioral aspects of services and the challenges associated with analyzing such models. Consequently, we have extended the resource-wise timing behavioral language, called Remes, and have provided associated analysis techniques. We have also introduced a language for composing and verifying services, on demand. We have illustrated our approach on several small examples, yet the comprehensive analysis of the achieved research results needs to be performed in more realistic case-studies and it is subject of future work.

5.1 Summary of Thesis Contributions

In this work, we have presented our work aiming at answering the formulated research questions of Chapter 2, which can be summarized in the following concrete lines of contribution:

Analysis-wise comparison between component-based frameworks. In this thesis, we present the comparison-driven results of several popular component-based frameworks in term of analysis of extra-functional

properties, i.e., performance and reliability. Foremost, the comparison has set our work in the appropriate context, while showing how our favorite framework handles performance and reliability analysis, through a small real-time system example.

SOS vs. CBS. Due to the similarity of the fundamental principles on which SOS and CBS are built, we have carried out a deeper comparison between the two paradigms, in order to identify the differences, but also to emphasize what is required from a component-based approach to become fit for service-orientation, too. The results has shown that the level of similarities is significant enough to allow us to use an unified behavioral model for both, service-oriented and component-based systems.

REMES **behavioral language for service-oriented setup.** REMES is a resource-wise timed behavioral language that enables modeling of services as modes that have a notion of explicit entry- and exit points. We have enriched the original modes with service attributes and service pre-, and postconditions, in order to expose the service interface for potential service discovery, and set the ground for formal analysis of services. The language supports modeling both single and composed service, via a hierarchical dynamic composition language that allows to create new services, using binary operators, as well as adding and/or deleting services from lists. In addition, it allows serial, parallel and parallel with synchronization service composition.

Checking the correctness of REMES **services.** We present two approaches to check the correctness of REMES services that rely on the forward analysis technique. First approach is defined using Hoare triples and Dijkstra and Sholten's strongest postcondition predicate transformer. It allows calculation of the strongest postcondition for a REMES service by hand and it is more suitable for less complex services. Since the original semantics of REMES is given in terms of PTA, in second approach, we show algorithmic calculation of the strongest postcondition for services denoted as PTA. The approach makes checking the correctness of more complex services feasible, and awaits implementation in the UPPAAL CORA tool.

5.2 Future Research Directions

We have identified several possible directions that our research could follow in the future. The current approach has not been validated yet. Our intention is to apply the proposed modeling and analysis techniques presented in this thesis on real-world case-studies/systems. This could add our understanding of how to extend our work such that it becomes more complete and adequate for real systems, but also uncover some of the limitations of our approach.

In SOS, it is sometimes the case that the service user needs to replace a particular service with one of better QoS but similar functionality. In order to ensure that the two services are behaviorally similar, one needs to verify a refinement relation between services. As known, the existence of a timed simulation relation is a sufficient condition for proving language inclusion, hence refinement. Future work includes a detailed investigation on how the simulation relation between two REMES services can be proved, as there is no decidability result regarding computing a simulation relation between two PTA.

Moreover, we have found interesting to be able to manipulate different types of resources within the same service model, and carry out various types of analysis. In the future, we plan to investigate possibilities for trade-off analysis of QoS attributes. To tackle such problems, the current research needs to be extended to dual-priced timed automata (DPTA) [40], as the modeling framework, in which separate costs model various QoS. In addition, algorithms that provide the strongest postcondition calculation need to be implemented in UPPAAL CORA. At the moment, we rely on tools that enable modeling of REMES services in an Eclipse-based environment and their transformation to UPPAAL for the analysis purposes [32]. However, our plan is to provide a stand alone tool suitable for modeling, correctness check, and resource-wise analysis via UPPAAL CORA of both single and composed services.

Bibliography

[1] Ivica Crnkovic and Magnus Larsson. *Building Reliable Component-Based Software Systems.* Artech House publisher, 2002.

[2] Manfred Broy, Norbert Diernhofer, Johannes Grünbauer, Michael Meisinger, Martin Rappl, Sabine Rittmann, Bernhard Schätz, Maurice Schoenmakers, and Bernd Spanfelner. Service-Oriented Development - Whitepaper. Whitepaper, Technische Universität München, 2006.

[3] Aida Causevic and Aneta Vulgarakis. Towards a unified behavioral model for component-based and service-oriented systems. In *2nd IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS 2009).* IEEE Computer Society Press, July 2009.

[4] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1.* IBM, 2003.

[5] Nickolas Kavantzas, David Burdett, Greg Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web services choreography description language version 1.0. World Wide Web Consortium, Candidate Recommendation CR-ws-cdl-10-20051109, November 2005.

[6] Object Management Group (OMG). *Business Process Modeling Notation (BPMN) version 1.1.*, January 2008.

[7] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.

[8] Cristina Seceleanu, Aneta Vulgarakis, and Paul Pettersson. Remes: A resource model for embedded systems. In *In Proc. of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009)*. IEEE Computer Society, June 2009.

[9] Aneta Vulgarakis, Cristina Seceleanu, Paul Pettersson, Ivan Skuliber, and Darko Huljenic. Validation of embedded systems behavioral models on a component-based ericsson nikola tesla demonstrator. In *11th InternationalConference on Quality Software (QSIC 2011)*. IEEE, July 2011.

[10] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457, 1975.

[11] Edsger W. Dijkstra and Carel S. Scholten. *Predicate calculus and program semantics*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[12] Kim Guldstrand Larsen and Jacob Illum Rasmussen. Optimal reachability for multi-priced timed automata. *Theor. Comput. Sci.*, 390:197–213, January 2008.

[13] Q-ImPrESS Project. `http://www.q-impress.eu/wordpress/`.

[14] Manfred Broy, Ingolf Krüger, and Michael Meisinger. A formal model of services. *ACM Transactions on Software Engineering Methodology (TOSEM)*, 16(1), 2007. available at http://doi.acm.org/10.1145/1189748.1189753.

[15] Johannes Maria Zaha, Alistair P. Barros, Marlon Dumas, and Arthur H. M. ter Hofstede. Let's dance: A language for service behavior modeling. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 4275 of *Lecture Notes in Computer Science*, pages 145–162. Springer, 2006.

[16] Marek Rychlý. Behavioural modeling of services: from service-oriented architecture to component-based system. In *Software Engineering Techniques in Progress*, pages 13–27. Wroclaw University of Technology, 2008.

[17] Howard Foster, Wolfgang Emmerich, Jeff Kramer, Jeff Magee, David Rosenblum, and Sebastian Uchitel. Model checking service compositions under resource constraints. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 225–234, New York, NY, USA, 2007. ACM.

[18] Cristina Seceleanu, Aneta Vulgarakis, and Paul Pettersson. Remes: A resource model for embedded systems. In *In Proc. of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009)*. IEEE Computer Society, June 2009.

[19] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[20] Rajeev Alur. Optimal paths in weighted timed automata. In *In HSCCć01: Hybrid Systems: Computation and Control*, pages 49–62. Springer, 2001.

[21] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Inf. Comput.*, 104:2–34, May 1993.

[22] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 414 –425, jun 1990.

[23] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-Cost Reachability for Priced Timed Automata. In Maria Domenica Di Benedetto and Alberto Sangiovanni-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybris Systems: Computation and Control*, number 2034 in Lecture Notes in Computer Sciences, pages 147–161. Springer–Verlag, 2001.

[24] Kim Guldstrand Larsen and Jacob Illum Rasmussen. Optimal reachability for multi-priced timed automata. *Theor. Comput. Sci.*, 390(2-3):197–213, 2008.

[25] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[26] Hongyu Pei-Breivold and Magnus Larsson. Component-based and service-oriented software engineering: Key concepts and principles. In *33rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Component Based Software Engineering (CBSE) Track, IEEE*, August 2007.

[27] W. T. Tsai. Service-oriented system engineering: A new paradigm. In *SOSE '05: Proceedings of the IEEE International Workshop*, pages 3–8, Washington, DC, USA, 2005. IEEE Computer Society.

[28] Jim Amsden. Modeling SOA, parts I-V. October 2007.

[29] Marek Rychlý and Petr Weiss. Modeling of service oriented architecture: From business process to service realisation. In *ENASE 2008 Third International Conference on Evaluation of Novel Approaches to Software Engineering Proceedings*, pages 140–146. Institute for Systems and Technologies of Information, Control and Communication, 2008.

[30] Mary Shaw. The coming-of-age of software architecture research. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, page 656, Washington, DC, USA, 2001. IEEE Computer Society.

[31] Aida Causevic, Cristina Seceleanu, and Paul Pettersson. Modeling and reasoning about service behaviors and their compositions. In *Proceedings of 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA 2010), Formal Methods in Model-Driven Development for Service-Oriented and Cloud Computing track*. Springer LNCS, October 2010.

[32] Dinko Ivanov, Marin Orlic, Cristina Seceleanu, and Aneta Vulgarakis. Remes tool-chain - a set of integrated tools for behavioral modeling and analysis of embedded systems. In *Proceedings of the*

25th IEEE/ACM International Conference on Automated Software Engineering (ASE 2010), September 2010.

[33] Karel Masek, Petr Hnetynka, and Tomás Bures. Bridging the component-based and service-oriented worlds. In *EUROMICRO-SEAA*, pages 47–54, 2009.

[34] João Abreu, Franco Mazzanti, José Luiz Fiadeiro, and Stefania Gnesi. A model-checking approach for service component architectures. In *Proceedings of the Joint 11th IFIP WG 6.1 International Conference FMOODS '09 and 29th IFIP WG 6.1 International Conference FORTE '09 on Formal Techniques for Distributed Systems*, FMOODS '09/FORTE '09, pages 219–224, Berlin, Heidelberg, 2009. Springer-Verlag.

[35] Aida Causevic, Cristina Seceleanu, and Paul Pettersson. Formal reasoning of resource-aware services. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-245/2010-1-SE, Mälardalen University, June 2010.

[36] Maurice H. Ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Formal methods for service composition. *Annals of Mathematics, Computing & Teleinformatics*, 1(5):1 – 10, 2007. In: Annals of Mathematics, Computing & Teleinformatics, vol. 1 (5) pp. 1 - 10. Technological Education Institute of Larissa (TEIL), Greece, 2007.

[37] Gregorio Díaz, Juan José Pardo, María-Emilia Cambronero, Valentin Valero, and Fernando Cuartero. Automatic translation of ws-cdl choreographies to timed automata. In Mario Bravetti, Leïla Kloul, and Gianluigi Zavattaro, editors, *EPEW/WS-FM*, volume 3670 of *Lecture Notes in Computer Science*, pages 230–242. Springer, 2005.

[38] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 77–88, New York, NY, USA, 2002. ACM.

[39] Gwen Salaün, Lucas Bordeaux, and Marco Schaerf. Describing and reasoning on web services using process algebra. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 43, Washington, DC, USA, 2004. IEEE Computer Society.

[40] Kim Larsen and Jacob Rasmussen. Optimal conditional reachability for multi-priced timed automata. In Vladimiro Sassone, editor, *Foundations of Software Science and Computational Structures*, volume 3441 of *Lecture Notes in Computer Science*, pages 234–249. Springer Berlin / Heidelberg, 2005.