Mälardalen University Licentiate Thesis
No.134

Formal Approaches to Service-oriented Design: From Behavioral Modeling to Service Analysis

Aida Čaušević

June 2011

Department of Computer Science and Engineering
Mälardalen University
Västerås, Sweden

Copyright © Aida Čaušević, 2011
ISSN 1651-9256
ISBN 978-91-7485-012-3
Printed by Mälardalen University, Västerås, Sweden

Abstract

Service-oriented systems (SOS) have recently emerged as context-independent component-based systems. In contrast to components, services can be created, invoked, composed and destroyed on-the-fly. Services are assumed to be platform independent and available for use within heterogeneous applications. One of the main assets in SOS is service composability. It allows the development of composite services with the main goal of reusable functionality provided by existing services in a low cost and rapid development process on-the-fly. However, in such distributed systems it becomes difficult to guarantee the quality-of-services (QoS), both in isolation, as well as of the newly created service compositions. Means of checking correctness of service composition can enable optimization w.r.t. the function and resource-usage of composed services, as well as provide a higher degree of QoS assurance of a service composition. To accomplish such goals, we employ model-checking technique for both single and composed services. The verification eventually provides necessary information about QoS, already at early development stage. This thesis presents the research that we have been carrying out, on developing of methods and tools for the specification, modeling, and formal analysis of services and service compositions in SOS. In this work, we first show how to formally check QoS in terms of performance and reliability for formally specified component-based systems (CBS). Next, we outline the commonalities and differences between SOS and CBS. Third, we develop constructs for the formal description of services using the resource-aware timed behavioral language called REMES, including development of language to support service compositions. At last, we show how to check service and service composition (functional, timing and resource-wise) correctness by employing the strongest postcondition semantics. For less complex services and service compositions we choose to prove correctness using Hoare triples and the guarded command language. In case of complex services described as priced timed automata (PTA), we prove correctness via algorithmic computation of strongest postcondition of PTA.

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 1

1 Introduction 3
1.1 Preliminaries 8
1.1.1 Service-oriented Systems 8
1.2 REMES: A Resource Model for embedded Systems 9
1.3 Formal Modeling and Analysis of Software Systems 11
1.3.1 Timed Automata 12
1.3.2 Priced Timed Automata 13
1.3.3 Model-checking technique 14
1.4 Thesis Overview 15

2 Research Summary 19
2.1 Problem Description 19
2.2 Research Questions 20
2.3 Research Methodology 22

3 Research contributions 25
3.1 Component-based vs. Service-Oriented Systems: System Modeling and Analysis 25
3.2 Formal Modeling of Resource-aware Service Behaviors in REMES 26
3.3 Checking the correctness of REMES services 28
3.4 Questions Revisited 29

4 Related Work 31
4.1 Services vs. Components 31
4.2 Service-oriented Frameworks 32

4.3 Checking Properties of Services and their Compositions . 33

5 Conclusions and Future Work 35

5.1 Summary of Thesis Contributions 35

5.2 Future Research Directions 37

Bibliography 39

II Included Papers 45

**6 Paper A:
Analyzing Resource-Usage Impact on Component-Based Systems Performance and Reliability 47**

6.1 Introduction 49

6.2 Working Example: A Real-time Multi-processor System . 50

6.3 Quality Prediction in Current CBFs 50

6.3.1 SOFA 51

6.3.2 KLAPER 52

6.3.3 Koala 53

6.3.4 ROBOCOP 53

6.3.5 BIP 54

6.4 Our approach 55

6.4.1 Example Revisited: Analyzing the Multiprocessor System's Performance and Reliability using UPPAAL 56

6.4.2 PTA Models 58

6.4.3 Analysis 59

6.5 Conclusions and Future Work 61

Bibliography 62

**7 Paper B:
Towards a Unified Behavioral Model for Component-Based and Service-Oriented Systems 67**

7.1 Introduction 69

7.2 Characteristics of CBSE and SOSE 70

7.3 Behavioral Modeling in CBS and SOS 73

7.3.1 Component-Based Modeling 74

7.3.2 Service-oriented Modeling 77

7.4 Discussion and Related Work 79

7.5 Conclusions and Future Work 80
Bibliography 83

8 Paper C:
Modeling and Reasoning about Service Behaviors and their Compositions **87**
8.1 Introduction 89
8.2 Preliminaries 90
8.2.1 REMES modeling language 90
8.2.2 Guarded command language 91
8.3 Behavioral Modeling of Services in REMES 92
8.4 Hierarchical Language for Dynamic Service Composition: Syntax and Semantics 97
8.5 Example: An Autonomous Shuttle System 101
8.5.1 Modeling the Shuttle System in REMES 102
8.5.2 Applying the Hierarchical Language 103
8.6 Discussion and Related Work 105
8.7 Conclusions 106
Bibliography 109

9 Paper D:
Checking Correctness of Services Modeled as Priced Timed Automata **113**
9.1 Introduction 115
9.2 Preliminaries 116
9.2.1 REMES modeling language 116
9.2.2 Priced Timed Automata 117
9.2.3 Symbolic Optimal Reachability 119
9.3 Algorithms for Service Strongest Postcondition Calculation 120
9.3.1 Strongest Postcondition 121
9.3.2 Strongest postcondition calculation and minimal cost reachability 121
9.3.3 Strongest postcondition calculation and maximal cost reachability 123
9.4 An Illustrative Example 124
9.5 Discussion and Related Work 127
9.6 Conclusions 128
Bibliography 131

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.

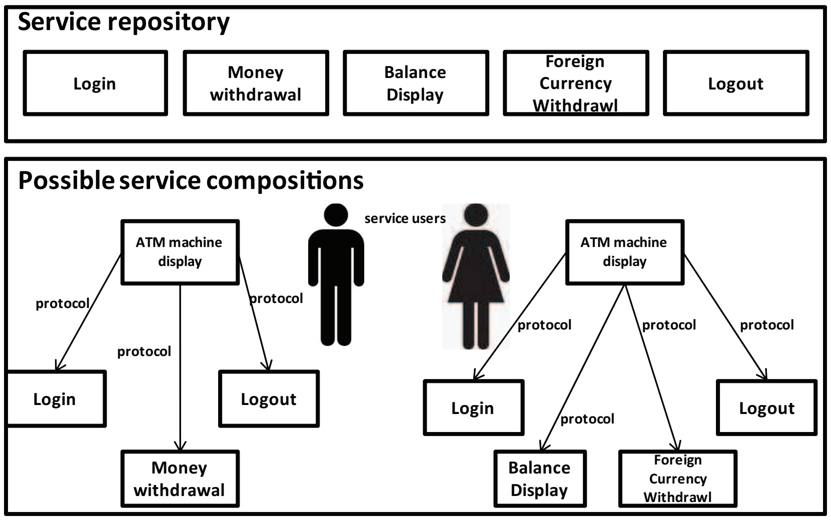

Figure 1.1: Service oriented ATM system

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

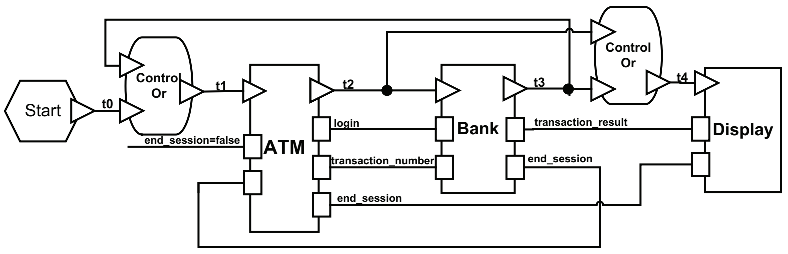

Figure 1.2: Component-based ATM system

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).

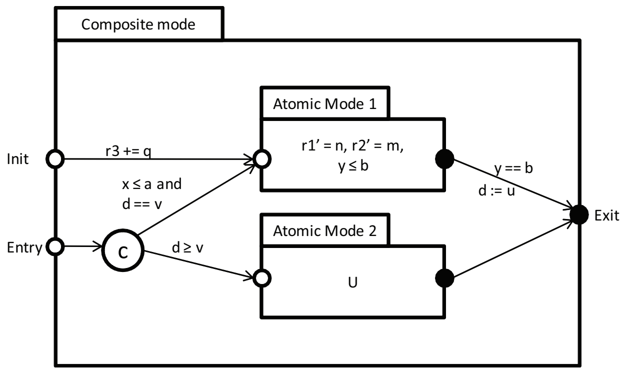

Figure 1.3: A REMES mode

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, A = (g, S) (e.g., (y == b, d := u) in the figure), is a statement S (in our case d := u), preceded by a boolean condition, the guard (y == 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, $x \leq a \wedge d == v$ or $d \geq v$ 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., $y \leq b in$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., r3 += 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`.

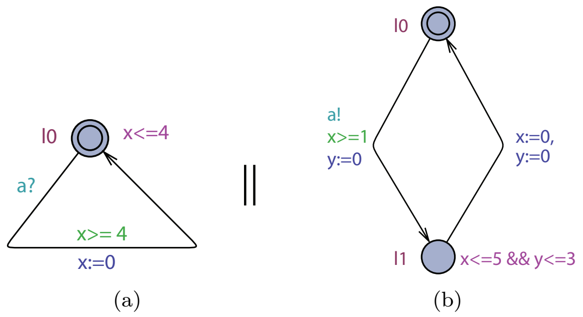

Figure 1.4: A timed automata

-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

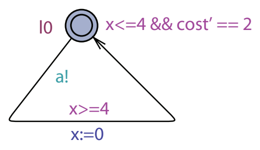

Figure 1.5: A priced timed automaton

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

1.3.3 Model-checking technique

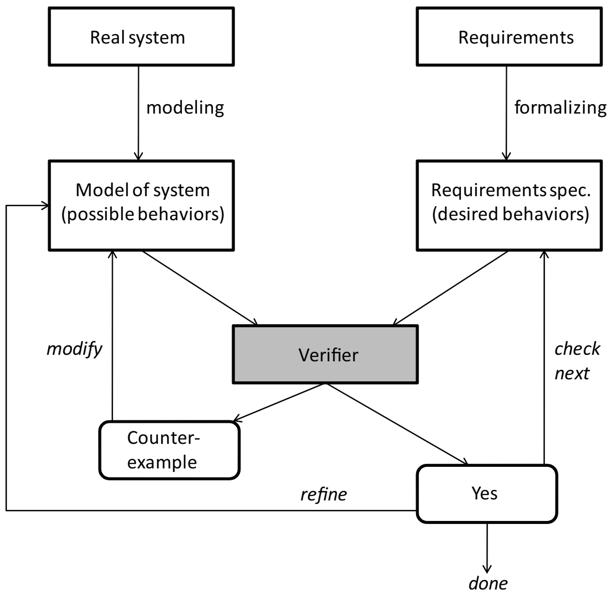

Figure 1.6: Verification methodology of model checking [25]

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 behaviors. 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. Although, 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.

II

Included Papers

Chapter 6

Paper A: Analyzing Resource-Usage Impact on Component-Based Systems Performance and Reliability

Aida Čaušević, Paul Pettersson, Cristina Seceleanu
In Proceedings of International Conference on Innovation in Software Engineering (ISE08), Vienna, Austria, December, 2008

Abstract

An early prediction of resource utilization and its impact on system performance and reliability can reduce the overall system cost, by allowing early correction of detected problems, or changes in development plans with minimized overhead. Nowadays, researchers are using both academic and commercial models to predict such attributes, by measuring them at earliest stages of system development. In this paper, we give a short overview of existing prediction models for performance and reliability, targeting popular component-based frameworks. Next, we describe our own approach for tackling such predictions, through an illustration on a small example that deals with estimations of energy consumption.

6.1 Introduction

In most component-based system development efforts, a great deal of time is spent on ensuring that the functional requirements are being properly implemented, while performance and reliability requirements are given a lower priority [1]. The premise of this paper is that performance and reliability, when added to functionality, constitute a necessary and complete set of metrics for reducing the development cost of complex component-based systems, be they service-oriented or embedded systems.

As the degree of using existing hardware and software resources affects quality attributes like performance and reliability in particular, there is a strong need of system models on which dedicated prediction methods can be applied, as early as possible in the development cycle. Such methods should be able to estimate how various changes in component-wise resource utilization impact on the respective response times (performance metric) and/or number of software faults and errors (reliability metric). The predictions would then later guide the designer towards potential redesigns, e.g., in case the system's resource utilization nears upper thresholds of performance criteria. Most of the component-based frameworks (CBFs) for system design rely on methods for estimating either performance or reliability changes under given resource utilization scenarios. However, few of the approaches can deliver predictions for any possible system behavior. Most of them cover subsets of behaviors, by using simulation or lightweight formal methods.

The goal of this paper is twofold: first, we briefly review some of the most significant component models and underlying approaches for analyzing the dependency between resource consumption, performance and/or reliability attributes; second, we show how can formal verification techniques, in particular model-checking, be used to predict the performance and reliability of a small real-time, distributed system, modeled as a *priced timed automata* [2]. The intention is to describe a way of carrying out a significant number of experiments, without increasing the system's development cost.

6.2 Working Example: A Real-time Multiprocessor System

Let us consider a simple distributed system made of 5 components, out of which C0, C1, C2 will be mapped, on a selected target platform, onto 3 real-time tasks that have to execute on 2 available processors, assumed to be components CPU0, CPU1. Moreover, a task component Ci can be executed on just one available processor at any point of time, and it cannot be split in more jobs and executed on both processors. Except for component C1 that can only be executed on processor CPU1, the other two task components can be executed on any available processor.

The tasks are characterized by attributes like cpu_type and deadline, and are assumed to be independent and non-preemptive. The attributes specify the type of CPU on which each component is allowed to execute and the relative deadline, respectively. The target system abstraction, that is the CPUs, have the attributes type and frequency, denoting the respective CPU's type and speed, respectively.

Assuming a component-based system model, our analysis goal is to select both the best mapping of the task components onto the available processors, and the optimal sequence of execution such that all tasks will meet their deadlines and the energy consumption for each execution is minimized. We recognize here both a *performance* as well as a *reliability* prediction problem of the composition, as follows:

- performance: minimize the energy consumption of each task, for each execution, given the fact that the consumed energy is directly proportional to the task's execution time;
- reliability: minimize the number of software errors, that is, number of times the tasks fail to meet their deadlines, respectively.

Here, the identified resources are energy and the computation resources, that is, the processors.

6.3 Quality Prediction in Current CBFs

In this section, we will give a brief overview of the current prediction techniques for performance and/or reliability, for some of the most popular state-of-the-art component models, if possible, in the context of the above example. Concretely, we will look at Palladio Component Model, Klaper, SOFA, Koala, Robocop, and BIP.

Palladio Component Model (PCM) represents a domain modeling language used for model-driven performance prediction [3]. The main purpose of introducing PCM is to perform early performance prediction of alternative software architectures. Therefore, the analysis methods are able to calculate metrics like the response time of provided services in a software system, with respect to parametric dependencies within components, and the actual usage profile of a software system. Simulation tools generate simulation source code and scenarios, based on instances of the PCM [4].

In our example's context, assuming that both tasks and processors are each described by a PCM, one could get the following:

- performance: calculate the response time of each task, for various task-processor allocation scenarios and energy-usage profiles;
- reliability: no support provided.

The main advantage of PCM-based prediction methods is that it reduces model complexity by providing models for different component-based software engineering (CBSE) developer roles, which are parameterized with the targeted platform attributes. The disadvantage of the approach is lack of support of reliability prediction techniques, of real-time analysis, and the lower degree of assurance provided by simulation, when compared to full formal verification.

6.3.1 SOFA

The Software Appliances (SOFA) component model, in its current version SOFA 2 [5], provides a modeling platform for software components, based on a hierarchical model with nested components that are able to communicate over defined interfaces. In terms of prediction, this model comes together with an infrastructure that allows for general component monitoring that gathers performance related information. The performance attributes are fed to the performance modes, and then the gathered information is feedbacked to the component model, whose resource usage level is adjusted accordingly. The process is iterative, stopping when a reasonable trade-off between resource-usage and performance is obtained.

The component behavior is captured by annotating invocations with lists of resource demands. The allocation of resources is described within the deployment model. With SOFA, one can predict how performance attributes change when the application scale changes.

If we assume the working example of section 6.2, by employing this approach, we can get the following results:

- performance: predict each task's completion time, based on monitoring, and a chosen energy-demand level;
- reliability: not supported.

6.3.2 KLAPER

KLAPER (Kernel LAnguage for PErformance and Reliability analysis) [6] is a kernel language intended to capture relevant information about non-functional attributes of component-based systems (CBS), focusing mainly on performance and reliability. To derive meaningful analysis models from design models, one can work within the so-called KLAPER-based transformation framework; the latter accepts as input software design models expressed via heterogeneous notations, and produces as output various performance and reliability models. Assuming that we virtually apply KLAPER-based transformations to our task and processor models, the following can be predicted:

- performance: calculate each task's execution time, as a function of the service speed attribute represented by the processor frequency;
- reliability: calculate each task's probability of failure to meet its respective deadline.

A remarkable feature of KLAPER is that it offers the possibility of a direct transformation from design-oriented into different analysis-oriented notations such as Petri Nets, Discrete Time Markov processes, and Extended Queueing Networks (EQN). Another advantage is that one can associate scheduling policies with a resource, in order to model access control policies. As such, the framework allows for a direct estimation of the resource-usage impact on quality attributes like performance and reliability. The lack of feedback between the analysis step and the design models could be considered as a disadvantage of the approach.

6.3.3 Koala

Koala [7] is a software component model, introduced by Philips Electronics, designed to build product families of consumer electronics. Resource information is exposed at the component's interface. The *provides* interface defines the operations offered by the component, whereas the *requires* interface defines the operations of other interfaces that the component needs to use. Since in a Koala model all the external functionality that is required by the component needs to go through the "requires" interface, it is somewhat straightforward to estimate the use of the system's resources, such as memory utilization. To estimate a Koala component's static memory consumption, one can assume that a special type of *reflection* provides the interface.

In the context of our example, one could analyze the following:

- performance: assuming that all the components (tasks) of the example require the same amount of memory, and that the latter is specified, yet some of the components need to queue to get memory access, one could analyze how various component configurations can affect task execution and system performance, under different memory availability scenarios. In addition, if one tags the task that uses the largest amount of memory as "slow", one could estimate the number of failures that affect the system budget, and compare the new budget with the actual execution cost, hence capturing performance changes;

- reliability: estimate the number of failures that might occur during the execution of "slow" tasks.

The above technique supports budgeting, that is, the expected values of the resource consumption of non-implemented components can also be accounted for. On the other hand, the approach can be used to estimate the total system performance, in a compositional fashion, only on specific, reduced-size scenarios for which the set of components instantiated in a composition is known before run-time.

6.3.4 ROBOCOP

The Robust Open Component Based Software architecture for Configurable Devices Project (ROBOCOP) is inspired by the Koala component model. It consists of several models that provide parts of the component information, respectively.

To solve the static memory estimation problem, a scenario-based simulation approach has been introduced [8, 9]. This approach delivers resource estimations for a set of scenarios that represent critical usages/executions of the system. The proposed resource model specifies the predicted resource consumption for all the operations implemented by the services of an executable component. As such, the model contains a number of cost functions that give the operations' costs. There can be multiple cost functions, for each resource. To increase the faithfulness of the prediction, the resources that are claimed and released are specified per operation.

Let us assume that we have the resource-wise and functional behavioral model (written for example in binary code) for the components in our example, and the accompanying application scenario that describes service instances and bindings between the respective components. In such case, we would be able to proceed with the analysis described below:

- performance: for components `C0`, `C1`, `C2` their worst-case response times could be checked against the respective deadlines;
- reliability: compute the number of missed deadlines for `C0`, `C1`, `C2`; this result can reflect the reliability of the modeled system.

Muskens and Chaudron are describing a method for run-time resource consumption in multi-task CBS, via a formal approach that allows prediction of dynamic resource consumption [10].

6.3.5 BIP

The acronym BIP stands for **B**ehavior, **I**nteraction, **P**riority, and is a framework for modeling heterogeneous real-time CBS. Each component is obtained as a superposition of three layers. The lower layer describes the component behavior, the intermediate layer includes connectors that describe component interactions, and the upper layer is a set of priority rules that model scheduling policies for interactions. BIP does not make an explicit distinction between inputs and outputs, such that the global variables can be treated either as inputs or as outputs. Basu et al. [11] give an example of performance evaluation of timed tasks that process events from a bursty event generator, all modeled and executed in the BIP framework.

If we employed BIP to tackle our example, we would get the following results:

- performance: worst-case execution time of `C0`, `C1`, `C2`, for any valid task-processor allocation scenario, and scheduling analysis via formal verification.

The advantage of the approach is threefold: (a) it accounts for possible heterogeneity of components; (b) it provides a rigorous, correct-by-construction basis for the study of architectural transformations, and (c) it is supported by formal verification tools for the compositional analysis of performance, or important properties such as deadlock-freedom.

6.4 Our approach

Our approach is based on the SaveComp component technology and its component modeling language SaveComp Component Model (SaveCCM), which have been developed within the SAVE project[1] [12]. The semantics of the core part of the language is given as models of timed automata. Having semantics defined in terms of timed automata, we are able to analyze SaveCCM models within different model-checking tools (e.g. UPPAAL[2]). Recent research on SaveCCM has been performed in the area of embedded control applications of vehicular systems [13, 14]. The electronics in vehicles represents a class of systems where quality attributes, such as reliability and resource usage, have impact throughout the development process. The analysis that has been done with SaveCCM within case studies mainly address these topics.

In this section, we will present our model, which is based on Priced Timed Automata (PTA) theory [2], an extension of Timed Automata (TA) with costs on locations and edges. In PTA, costs are associated with edges, to define the cost of executing a corresponding action transition, and location, to define the cost per time unit of delaying there. The PTA framework provides modeling and analysis of continuous, monotonically increasing consumption of resources, e.g. energy consumption. Since the PTA framework does not provide combined reasoning about monotonic (e.g. energy) and non-monotonic resources (e.g. memory), we

[1]SAVE project is supported by Swedish Foundation for Strategic Research.

[2]The UPPAAL tool is developed in collaboration between Uppsala University, Sweden and Aalborg University in Denmark. More information is available at http://www.uppaal.com/

will treat the whole amount of required memory as static, allocate the total memory amount at the beginning of each task execution, and model it as a discrete value. This problem can be solved with multi-priced TA [15], which are PTA with multiple cost variables evolving according to given rates for each location. Due to space limitation, we will not describe the model of priced timed automata here, but refer the reader to [2] for a thorough description of the framework.

6.4.1 Example Revisited: Analyzing the Multiprocessor System's Performance and Reliability using UPPAAL

To exemplify our approach, we recall the component-based system presented as our working example in Section 6.2. The system model is depicted as a SaveCCM-based description in Figure 6.1. We model the example system as the composition of three real-time tasks T0, T1, and T2, corresponding, in the Save-CCM representation, to C0, C1, and C2, respectively, and two processors P_0 and P_1 describing components CPU0 and CPU1. Note that tasks are assumed to be independent, that is, their execution do not depend on the state, results, or side effects of the other tasks.

Our first goal is to model non-preemptive multiprocessor task scheduling. Tasks T_i can be executed in parallel if there are available processing resources P_j, enabling multiple requests to be served simultaneously. Each task T_i is defined by its deadline (D_i). Processors (P_j) are characterized by their period (P_j) and consumed energy (E_j). We introduce the notion of *task execution time* (ET_i), since the consumed energy is directly proportional to the latter. As such, in a quite simplified form, ET_i can be equated to:

$$ET_i = NoCyc * Per_j$$

where $i \in \{0, 1, 2\}$ is the task identifier, $j \in \{0, 1\}$ is the processor identifier, and NoCyc represents the total number of CPU cycles per task, which we assume known.

The consumed energy, per task, is given by the following equation, also in an abstracted form:

$$E_i = ET_i * w_j * PW_j,$$

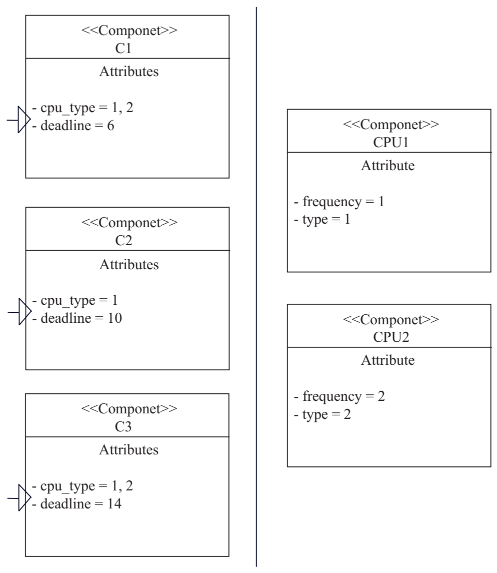

Figure 6.1: SaveCCM component model

where PW_j models CPU power dissipation, which we assume fixed and known. Note that E_i is a weighted function of ET_i and PW_j, where the given weight w_j expresses the relative importance of E_i, which in turn influences the final cost. The accumulated energy consumption is then given as the following cost:

$$c = \sum_{i=0}^{2} E_i$$

Choosing the values of the weights is subjective, depending on both the application and the analysis goals.

When a task execution completes by meeting its deadline, it sends an acknowledgment to some processor to inform that the execution is

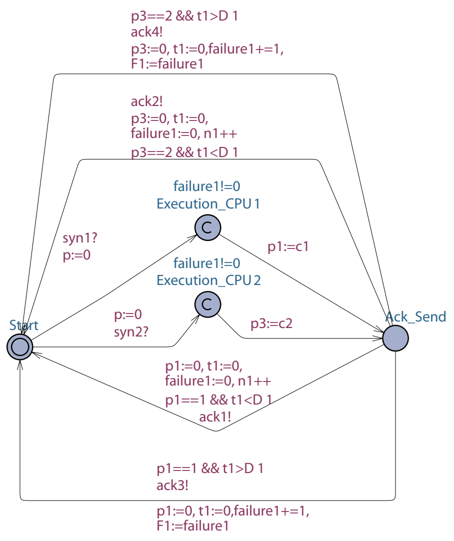

Figure 6.2: The model of a task

finished. Our complementary goal is to model a system that would let us predict the total resource usage and its impact on performance and reliability.

6.4.2 PTA Models

We model our example as a collection of five non-deterministic PTA. The PTA (also called processes) communicate using synchronization channels and shared global variables (i.e. variables that can be read and written by all of the processes). The model consists of three automata representing the tasks (T0, T1, T2) that are competing for two available automata representing the processors (CPU0 and CPU1).

The model of a task is shown as a PTA in Figure 6.2. It has two locations: Start and AckSend. The synchronization with an available processor is modeled by using two channels, syn1 (models synchronization with processor P_0), and syn2 (models synchronization with processor P_1). The execution start of a task is controlled by the failure1 variable — a counter (bounded integer) that indicates whether the task failed to meet its deadline or not. The counter is initially set to one, and increased if

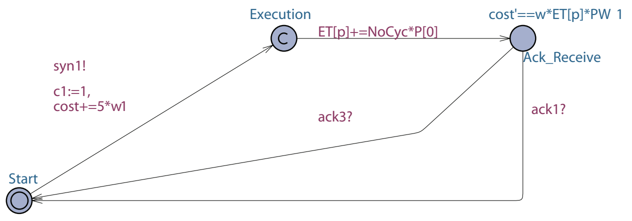

Figure 6.3: The model of the processing unit

a failure occurs. If the execution is successful, that is, the deadline is met, the variable is reset to zero. This also indicates that the task is no longer in the ready queue. For each task, the variable pi is assigned the processor number (cj) on which it is currently executing. Depending on the execution result, one of two types of acknowledgment can be sent; in case the task completes successfully, ack1 or ack2 are sent, depending on which CPU the task is synchronizing with; in case the task fails to meet its deadline, ack3 or ack4 are sent.

The PTA model of a processor consists of two locations: Start and AckReceive. In Figure 6.3, a synchronization channel syn1 is used for synchronization with the tasks present in the ready queue. Variable cj stores the processor number used by task variable pi to identify the task that is being executed on the respective processor. If the execution is successful, acknowledgment ack1 is received by the processor, or ack3 otherwise. The cost of energy consumption is influenced by the assigned weights, execution times, and CPU power dissipation as described previously in this section. The minimum cost of energy consumption is the infimum of the costs of all finite executions from the first to the last state.

6.4.3 Analysis

The best performance analysis could include finding the best mapping of tasks onto available processors, such that all task deadlines are met, but also the execution order for which the power consumption is minimal. The results are presented as cost values of the computed optimal execution traces. Recall that in our example task T_1 can be only executed on processor P_0.

Usually, the reliability of a system reflects its ability to perform a given function under present conditions, in a specified period of time.

Scenario	Order of execution	Cost
1	(T_0, P_0)-(T_1, P_0)-(T_2, P_0)	15
2	(T_0, P_1)-(T_2, P_0)-(T_1, P_0)	20
3	(T_0, P_0)-(T_2, P_1)-(T_1, P_0)	30

Table 6.1: Best task mapping with minimum cost.

Sequence of task execution	Ratio
(T_0, P_1)-(T_2, P_0)-(T_1, P_1)	23/10
(T_2, P_0)-(T_0, P_1)-(T_1, P_1)	47/10
(T_2, P_0)-(T_1, P_1)-(T_0, P_0)	20/10
(T_1, P_1)-(T_2, P_0)-(T_0, P_0)	37/10

Table 6.2: Ratio between number of failures occurred and system executions

Our assumption is that during normal system execution, failures can occur, and this affects directly the overall system reliability. In order to account for failures in our PTA-based model, we analyze the reliability via a ratio between the number of failures occurred during all system invocations, and the number of system invocations. The results are given in Table 6.2.

We note that the cost is minimum in case when all tasks T_0, T_1 , and T_2 are competing for the same processor. The cost value presented in Table 6.1 shows that the cost is minimal if all tasks are being executed on processor P_0, which is assumed to be "less expensive", than the other one. Of course, cost could be higher if we assigned additional cost for waiting in ready queue. Table 6.1 presents the cost results assuming all tasks complete successfully. Beside the minimum cost, we also present in Table 6.1 costs for scenarios in which T_1 has to wait additional time for tasks T_0 and T_2 to complete. Tasks T_0 and T_2 arrive before T_1 to the ready queue and they are allowed to compete for all available CPU resources. In these scenarios, task T_1 is forced to wait in the ready queue, despite the fact that early execution of this task would result in lower cost for the whole system. Clearly, if failures occur during execution, such that the tasks need to be executed more than once in order to complete, the final cost is much higher.

We have noticed that most of the failures occur in situations when two tasks with the greatest and the smallest execution time and deadline T_2 and T_0, respectively) are competing for the same free processor, and T_2 gains its CPU time (see Table 6.2). In that case, T_0 has to wait an additional time to start its execution. This problem can be easily solved by including some additional scheduling policy, however this is out of the scope of this paper.

6.5 Conclusions and Future Work

In this paper, we have briefly reviewed the performance/reliability analysis techniques available in the state-of-the-art component-based frameworks, and their possibility of estimating the impact of changing resource usage on the above mentioned quality attributes. Although extensive work has tackled such problems, the real-time systems area is left less researched. This has motivated us to propose a priced timed automata model-checking approach for component-based systems described in the SaveCCM modeling language that is designed for a real-time and embedded systems. As demonstrated in a small accompanying example, our approach allows for rigorous predictions of performance and/or reliability, depending on the prices of using various resources, such as CPU, memory etc.

In the future, we plan to investigate the possibility of carrying out probabilistic quantitative predictions, by expressing properties to be verified in a probabilistic temporal logic (e.g., PCTL) [16].

Acknowledgments: The authors are grateful to Petr Tuma, Mikael Sjödin, and Tiberiu Seceleanu for their valuable comments on the example used in this paper. This work was partially funded in the context of the Q-ImPrESS research project (FP7-215013) by the European Union under the Information and Communication Technologies priority of the Seventh Research Framework Programme. We also want to express our gratitude to the Swedish national strategic research center PROGRESS, supported by the Swedish Foundation for Strategic Research (SSF) and Mälardalen University, for co-funding this work.

Bibliography

[1] Paul R. Work and Jr. H. E. (John) Johnson. Risk Management in Computer-Based Systems Development by Use of Performance and Reliability Metrics. In *Proceedings of the 1995 International Symposium and Workshop on Systems Engineering of Computer Based Systems*, pages 367–373. IEEE, 1995.

[2] 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.

[3] Steffen Becker, Heiko Koziolek, and Ralf Reussner. Model-based performance prediction with the palladio component model. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, pages 54–65, New York, NY, USA, 2007. ACM.

[4] Klaus Krogmann. Reengineering of Software Component Models to Enable Architectural Quality of Service Predictions. In Ralf H. Reussner, Clemens Szyperski, and Wolfgang Weck, editors, *Proceedings of the 12th International Workshop on Component Oriented Programming (WCOP 2007)*, volume 2007-13 of *Interne Berichte*, pages 23–29, Berlin, July31 2007. Universität Karlsruhe (TH).

[5] Tomas Bures, Petr Hnetynka, and Frantisek Plasil. Sofa 2.0: Balancing advanced features in a hierarchical component model. In

SERA '06: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications, pages 40–48, Washington, DC, USA, 2006. IEEE Computer Society.

[6] Vincenzo Grassi, Raffaela Mirandola, and Antonino Sabetta. From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 25–36, New York, NY, USA, 2005. ACM.

[7] Rob van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The koala component model for consumer electronics software. *Computer*, 33(3):78–85, 2000.

[8] Egor Bondarev, Michel R. V. Chaudron, and Peter H. N. de With. Compositional performance analysis of component-based systems on heterogeneous multiprocessor platforms. In *EUROMICRO-SEAA*, pages 81–91, 2006.

[9] Egor Bondarev, Peter de With, Michel Chaudron, and Johan Muskens. Modelling of input-parameter dependency for performance predictions of component-based embedded systems. In *EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 36–43, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Johan Muskens and Michel R. V. Chaudron. Prediction of run-time resource consumption in multi-task component-based software systems. In *CBSE*, pages 162–177, 2004.

[11] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in bip. In *SEFM '06: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[12] Jan Carlson, John Haakansson, and Paul Pettersson. SaveCCM: An analysable component model for real-time systems. In Z. Liu and L. Barbosa, editors, *Proceedings of the 2nd Workshop on Formal Aspects of Components Software (FACS 2005)*, volume 160 of *Electronic Notes in Theoretical Computer Science*, pages 127–140. Elsevier, 2006.

[13] Mikael Åkerholm, Johan Fredriksson, Kristian Sandström, and Ivica Crnkovic. Quality attribute support in a component technology for vehicular software. In *Fourth Conference on Software Engineering Research and Practice in Sweden*, October 2004.

[14] Hans Hansson, Mikael Åkerholm, Ivica Crnkovic, and Martin Torngren. Saveccm - a component model for safety-critical real-time systems. In *EUROMICRO '04: Proceedings of the 30th EUROMICRO Conference*, pages 627–635, Washington, DC, USA, 2004. IEEE Computer Society.

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

[16] Marta Kwiatkowska, Gethin Norman, Jeremy Sproston, and Fuzhi Wang. Symbolic model checking for probabilistic timed automata. *Inf. Comput.*, 205(7):1027–1077, 2007.

Chapter 7

Paper B: Towards a Unified Behavioral Model for Component-Based and Service-Oriented Systems

Aida Čaušević, Aneta Vulgarakis

In Proceedings of 2nd IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS09), Seattle, USA, July, 2009

Abstract

An early prediction of resource utilization and its impact on system performance and reliability can reduce the overall system cost, by allowing early correction of detected problems, or changes in development plans with minimized overhead. Nowadays, researchers are using both academic and commercial models to predict such attributes, by measuring them at earliest stages of system development. In this paper, we give a short overview of existing prediction models for performance and reliability, targeting popular component-based frameworks. Next, we describe our own approach for tackling such predictions, through an illustration on a small example that deals with estimations of energy consumption.

7.1 Introduction

As the complexity of software systems grows, be it in size, functional or extra-functional requirements, requests of mobility, or number of users, applying formal verification techniques to achieve predictability becomes increasingly necessary. In order to accomplish this, one should be able to model and analyze the system behavior throughout the whole system lifecycle.

Nowadays, the two most promising approaches that enable efficient, error-free software development are component-based software engineering (CBSE) [1], and service-oriented software engineering (SOSE) [2]. Since SOSE evolved from CBSE [3], it is easy to assume that they are similar and tightly connected. While in CBSE the smallest functional unit is a component, in SOSE that role is given to a service. Underlying concepts for both CBSE and SOSE are component/service modularization (i.e., system functionality is split up in order to achieve separate modules of behavior) and component/service composition (i.e., the separate modules are combined in order to get the overall system behavior). However, services enjoy looser coupling, higher reusability, and larger independence from implementation specific attributes. Moreover, SOSE subsumes features like dynamic service discovery, system scalability, and service availability.

In the service-oriented systems (SOS), it is still a challenge to predict Quality of Service (QoS) [4, 5]. This is the case because in SOS, QoS is not just a function of the quality of a provided service, but of the interdependencies between the services, the resource constraints of the runtime environment, and network capabilities. These make it difficult to predict how such factors might influence the system behavior. Since CBSE and SOSE are built around similar concepts [3, 6], it would be beneficial to use a unified behavior model for both paradigms.

This paper studies the CBSE and SOSE paradigms, and points out their distinctive features. In most CBSE and SOSE frameworks, the design description usually ends up at the architectural level, without support for details regarding the inner structure or behavior of the latter. SOS frameworks treat service behavioral modeling either at lower level of abstraction, tightly coupled with the underlying programming language [7, 8], or behavior is described at higher levels of abstraction than programming languages, but they lack a formal description [9, 10]. In component-based approaches authors usually give detailed description of behavior in terms of interfaces, connectors, and protocols [11–13], but internal behavioral description is assumed to be known, hence, it is often not described in detail.

In the SOSE community some recent research has been devoted to specification and prediction of QoS [4, 5], yet detailed service behavioral modeling with support for formal analysis is still needed. Seceleanu et al. introduce a behavioral modeling language called REMES [14] for formal modeling and analysis of component-based systems (CBS) with provided formal analysis support for component behavior. REMES may serve as an intermediate language between abstract architectural modeling of systems (e.g., architecture description languages (ADLs) [15]) and formal analysis models (e.g., timed automata [16]). Given the similarities between CBSE and SOSE, in this paper we show how REMES can be extended for behavior modeling of services in a service-oriented paradigm.

In brief, in this paper we contribute by :

- making a clear distinction between *architectural view* of CBS and SOS and *behavioral (internal) view* of components and services;
- comparing behavioral modeling in CBS and SOS;
- identifying ways in which REMES can become a unified behavioral model for both CBS and SOS;

The remainder of the paper is organized as follows. Section 7.2 draws a parallel between basic characteristics of CBSE and SOSE. Section 7.3 discusses the behavior modeling of both CBS and SOS, and shows how REMES can be extended towards modeling service behavior. In Section 7.4, we present the related work, and in Section 7.5 we conclude the paper.

7.2 Characteristics of CBSE and SOSE

CBSE and SOSE both aim at supporting component reusability, error-free and efficient software development. CBSE is more concerned with maintainability and component substitutability, whereas SOSE focuses on dynamic service discovery, low-cost application composition, reconfiguration and interoperability. Since both CBSE and SOSE can coexist in large scale systems, it is desirable to have common ways of modeling CBS and SOS architecture and behavior.

More specifically, CBSE aims at fulfilling the following goals [1]:

- support the development of systems as component assemblies;
- support component reusability;
- support maintenance and upgrade of systems while customizing and/or replacing components;
- guarantee a given level of quality attributes in given CBS.

Related to the above listed goals, the main focus in CBSE nowadays is on the specification of QoS, prediction of extra-functional attributes, component interfaces and processes, and component reusability issues. Despite being proven as successful for software reuse and maintainability, CBSE still lacks appropriate relations between abstract models and the underlying platform. The basic blocks of CBSE are component models. Component models are used in the development of components to describe their interfaces, illustrate their dependencies, specify their properties and composition mechanisms. A component framework should be able to express both structural requirements for interconnection and composition of components, as well as behavior-oriented requirements. For example, a system development can be seen as an interplay between design and deployment (see PROGRESS[1]). The interplay is ensured by the notion of virtual architecture that provides an abstraction of the targeted platform on which components can be (virtually) mapped to [17]. The virtual architecture can be gradually refined, at the same time with increasing the details of the system representation. Such a framework allows an early prediction of extra-functional properties, since the designer has access to models of the platform.

A typical CBSE development process is described in [18]. The development process of CBS is separated from the development process of its components, but is influenced by it. As shown in Figure 7.1, once components are implemented, they are released into a component repository. The system developer can select any component from the repository, provided that the respective component is functionally adequate, and interface compliant [18].

In comparison, the dynamic nature of services (that is, the fact that they can be created and destroyed at run-time), as well as their availability, and greater degree of distribution, might let us consider SOS as an evolution from CBS. On the other hand, their hard-to-ensure dependability and maintainability quality attributes could be considered as a

[1]http://www.mrtc.mdh.se/progress/

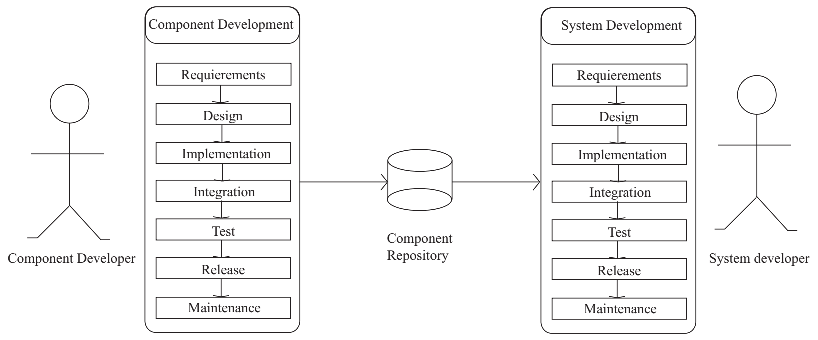

Figure 7.1: CBSE development process

weakness of the SOS paradigm. The SOS' low dependability (reliability and security) is a consequence of the flexible nature of the service oriented architecture (SOA), more concretely, of the fact that services can be composed at run-time. The SOSE approach and its corresponding SOA support a collection of loosely coupled services that can be invoked, published, and discovered. They communicate with each other via well defined standard interfaces and message exchanging protocols. Services are autonomous and platform independent entities, which can be composed at runtime in order to achieve the desired functionality. The discovery of the services is done through the service provider that searches the existing service descriptions and provides the suitable ones to the service user. SOSE has some specific features that include:

- standard-based interoperability - allows services that are developed on different platforms and by different vendors to be integrated with each other based on their service specification only;
- dynamic composition - new services can be formed and destroyed at runtime;
- dynamic orchestration - assumes the existence of a central controller (i.e., service provider) that has the responsibility to schedule service execution, according to the demands received from a service user;
- service interrupt - each service can be interrupted if any of the given constraints is violated, and QoS can be renegotiated.

In Figure 7.2 we give an abstract overview of the SOSE. Observe that in order to get the requested service, the user has to communicate with the service provider.

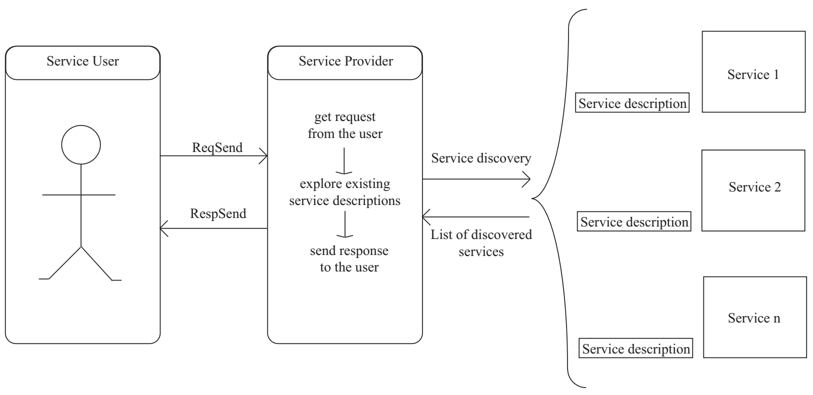

Figure 7.2: SOSE overview

In order to get the best out of both CBSE and SOSE, these approaches need to be combined, as shown by Collet et al. [19], through unified modeling languages.

7.3 Behavioral Modeling in CBS and SOS

In this section, we present the architectural views and the internal behavioral representations, of both CBS and SOS. The main goal is to show how can we reuse the behavioral modeling framework of CBS, in a service-oriented context. This endeavor targets a unified modeling environment for both paradigms. The architectural view depicts the structure of the system, which includes among others: the software components, the relationships and connections between them, and the externally visible properties of those components [20]. The behavioral view gives a description of the internal state change for each specific entity of that architecture. The architectural view is not meant to illustrate the concrete runtime state change or communication mechanism, but to explain how different parts of a system, be it CBS or SOS, work together to deliver certain properties, regardless of whether they are functional or extra-functional.

7.3.1 Component-Based Modeling

In the following, we exemplify the differences between the architectural views and the behavioral views by using for instance ProCom component model (A Component Model for Distributed Embedded Systems) [21]. The architectural view of the ProCom component model defines CBS as a collection of hierarchical structured and interconnected ProCom components with well defined *input-* and *output interfaces*. The component model models a *data-* and a *control flow*, where data can be read and written through the *data ports*. The component activation is controlled by *trigger ports*. The type of the components, *active* or *passive*, as well as the way of the interaction with other components or composition of components is given by the architectural view. The behavioral modeling of ProCom components is introduced in the REMES (REsource Model for Embedded Systems) [14] language that is briefly recalled in the following, via an example.

Component-based behavioral modeling in REMES. REMES is intended to provide a basis for modeling and analysis of embedded resources (e.g., energy, computation, communication). The model supports both continuous (like energy) and discrete (e.g., memory) resources. Basically, it is a state-machine behavioral language that supports hierarchical modeling, continuous time, and notions of explicit entry- and exit points that make it suitable for component-based system modeling.

In order to provide support for formal analysis, REMES can be easily transformed into Timed Automata (TA) [16] or Priced Timed Automata (PTA) [22] depending on the analysis goals (i.e., timing analysis, resource consumption, etc.). It is important to point out that the behavior of CBS with respect to resource usage can be formally analyzed (e.g., compute the most "expensive" trace w.r.t. utilized resources).

Simple ATM scenario in REMES. Here, we exemplify the internal component behavioral modeling, as well as the system architectural modeling of a simple ATM system. The ATM machine has a GUI through which the user communicates with a bank. Each user is required first to login and depending on the success/failure of the login the user can proceed with the transaction request or not. If the login is successful, the user can choose between four types of transactions: withdrawal transaction, deposition transaction, transfer transaction, and inquiry transaction; if the login is not successful, an information is then sent to the GUI, informing that the session has ended. The outcome of the transaction is displayed on the GUI, as well.

We model an abstracted version of the ATM system architecture in the ProCom language, used to describe the three constituent components: ATM, Bank, and Display as shown in Figure 7.3. The Display component performs simple output writing (transaction status or transaction result), so we chose not to model its behavior.

The ATM component is activated when a user tries to login by pressing the Start button on the GUI (then trigger signal t0 is sent). Trigger port t2 initiates an interaction with the Bank component. The Bank component uses login data and type of transaction request through login and transaction_number data ports, respectively. The Bank component triggers ATM via trigger port t3 if the login has been successful. Afterwards, the user can chose between four existing types of transactions. When the transaction has been chosen request is sent to Bank to be processed. Finally, transaction_result is sent to Display. If the login has not been successful then ATM informs the user through the Display component that the session has ended. The *control or* connector is a ProCom construct used to join control flows of two or more alternatives paths.

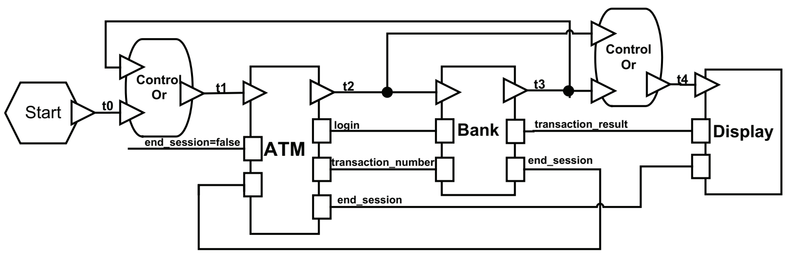

Figure 7.3: Component based ATM system as a ProCom-based description

We model the internal behavior (with respect to resource usage and time) of the ATM system components as modes in REMES. The modes of the ATM and Bank component are *composite*, as depicted in Figure 7.4. They are made up of *atomic* modes (i.e. Send Request, Check Login, Login, and Transaction, respectively), *conditional connectors* (C), and *discrete actions* (e.g. mem += 40).

For each mode, the discrete control is implemented using the *con-*

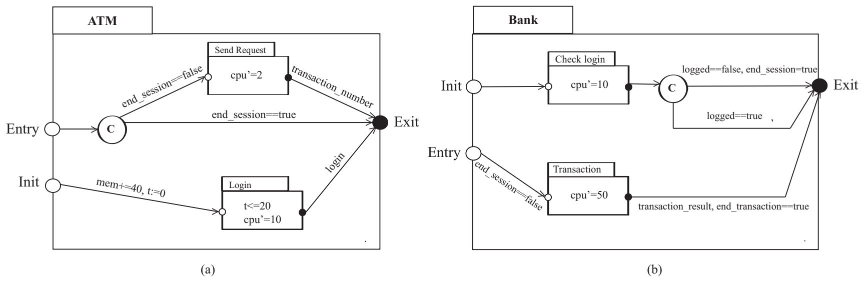

Figure 7.4: REMES modes for ATM and Bank

trol interface that consists of *entry-* and *exit* points. The *data interface*, constituted by global variables, enables the data transfer between modes. The ATM and Bank modes use the global variables login, transaction_number, and end_session. In order to distinguish the initialization of a mode from other entry actions, a composite mode may have an *Init* entry point, used to start the execution of the mode for the first time. Composite mode execution involves executing a sequence of actions that can be *delay/timed actions* or *discrete actions* depending on whether the control stays in the same mode or it is transferred to another mode or submode. A discrete action is a statement or list of statements preceded by a guard (boolean expression) that needs to hold in order for the corresponding edge to be taken and action executed. One REMES composite mode can have *conditional connectors* that allow the selection of edges from the outgoing ones, based on the values of the corresponding actions guards, as seen in Figure 7.4 (b).

REMES supports modeling of timed behavior and resource usage. The timed behavior is captured by a global variable of specialized type, clock, which specifies continuous variables evolving at rate 1. In this context, an *invariant* may be used to define for how long the execution continues in the same mode. When the invariant is violated the mode must be exited through one of the outgoing edges. For example, in the atomic mode Login an invariant $t \leq 20$ is used. In the ATM system, we make use of two resources: memory and CPU.

For analysis purposes, this REMES-based ATM system can be translated to a network of two PTA models of ATM and Bank. For details we refer the reader to [14].

7.3.2 Service-oriented Modeling

The SOS architecture defines the internal communication between services, the ways in which services can be published and discovered, orchestrated, or how can they make a choreography in order to be composed into more complex systems. On the other hand, the service behavioral view gives the detailed overview of the interfaces, actions, functionality, resources involved, and the possible interactions within the service. After a thorough investigation on related work, [7,8,10–13] one can notice the sparingly used service description at the behavioral level. We consider the detailed behavioral description as a crucial element for a proper understanding of a service-oriented model, and especially its analysis. Knowledge about the service behavior will not only help us to connect services in a correct manner, but will also help to provide rigorous reasoning about extra-functional requirements whose assurance is recognized to be insufficient, especially with respect to performance and security. Regarding performance, for instance, response times are difficult to calculate and ensure, usually because the concrete time for a user to be served is hard to predict. If one applies common security policies from CBS in SOS, one will notice that they will be violated, since services are supposed to be stateless, without a precise model of the environment (e.g., no user context forwarding). In order to address some of the identified issues, we propose ways of service behavioral modeling in Remes.

Service-Based Behavioral Modeling in Remes**.** Most of the research on SOS is devoted to describing service interfaces, discovery, interactions, service assembly, often leaving the service internal behavior unspecified. In this section, we investigate possibilities to describe and model the internal service behavior by exemplifying Remes.

In Remes one can already do:

- *Service modeling*: Remes supports simple and/or composed entity modeling in terms of modes and submodes that can provide service behavior description (i.e., internal workflow process). The additional logic based on well defined inputs, outputs, conditions, and guards is rich enough to provide both, simple and complex service functionality modeling.
- *Service provider modeling*: Remes provides possibility not only to model services, but also the service provider as an element of

service-user interaction.

- *QoS prediction*: for predicting the quality attributes (i.e., performance, reliability, etc.), it is possible to annotate the resource-wise behavior and to semantically transform REMES into TA or PTA; the transformation gives the opportunity to predict resource usage in SOS, for critical resources. Early resource prediction can provide insights in how to minimize consumption of highly critical resources, such as energy or memory.

The modeling concepts that differentiate services from components, which should be added to REMES in order to support service behavioral description are as follows:

- *Service failure report*: should be introduced for those services that do not meet given conditions. In REMES, it is currently assumed that a component runs until completing the execution, without interruption;
- *Service discovery mechanism*: The biggest issue here is publishing and presenting all capabilities that a service can offer, in order to be recognized by service users through the service provider;
- *Support for negotiation*: should be provided to model the cases when some hard constraints given by a user on requested services can not be provided by a service provider. The mechanism should enable the user to chose weaker constraints, in order to get an acceptably qualitative service (i.e., contract-based design of services [23]).
- *Reliability prediction*: to provide support for reliability modeling, REMES should be extended with probabilistic behavior, e.g., by adding probabilities on edges. The resulting model could then be transformed into a Discrete Markov Chain for analysis purposes.

We consider that the REMES modeling language is rich enough to be used for service orientation, provided that the above mentioned additional modeling capabilities are added to the current version of the language. Services, as basic entities of SOS, can be described in terms of modes, actions, guards, and invariants, which are basic elements that support behavioral description in REMES. The ATM example is a relevant example in which the component-based architectural description (depicted in Figure 7.4) is combined with the service-oriented modeling, as shown above. To summarize, the notions of service failure report, service discovery mechanism, support for negotiation, etc., should be added to REMES, in order to provide the designer with a framework suitable for service-orientation, too. Moreover, the REMES support for predicting quality attributes is a valuable tool, especially for SOS, when optimizing the usage of possibly critical resources could be of extreme importance.

7.4 Discussion and Related Work

The approaches that deal with component and service behavioral description can be broadly divided into three groups, based on the level of details exposed through the description.

The first group constitutes code-level (low level) behavioral description approaches. One of this kind in CBSE is Koala [24]. All behavioral information is exposed through interfaces, where *provides* interface defines operations offered by a component, and *requires* interface defines operations required by a component. WS-CDL [8] is a XML based language that exposes information on specific function provided by a service by the behavioral description. There are no details given on inner service behavior. BPEL [7] defines a notation for specifying business process behavior based on Web Services. The scope of the behavioral description includes a sequence of project activities, correlation of messages and process instances, and recovery behavior in case of failures and exceptional conditions. These low-level, code-driven approaches are valuable only when one has access to the implementation of the components/services, and especially when the components/services conform to a particular model. Nevertheless, REMES is more abstract and may be used already at early stages of system development, even when no detailed design description exists.

The second group is made of approaches that model behavior at a higher-level of abstraction than the previous. Two representative examples are UML-activity diagrams [25] and BPMN [9]. UML takes an object-oriented approach to model applications while BPMN takes a process-oriented approach. They both support graphical representation and they are aimed to be suitable for designers and analysts. Lack of formal behavioral description for both, components and services make them not completely suitable for throughout analysis as one provided by REMES. To this group also belong languages that represent semantic web services, such as OWL-S [9] and WSMO [10] that use logical statements in order to capture and manipulate service related information. They

assume that a service is a set of facts and rules related to service capabilities, extra-functional properties, and interfaces. While semantic web service descriptions are suitable in view of applying automated reasoning techniques (e.g., automated planning) over service descriptions, their suitability for use at the level of domain analysis and systems design is questionable. Domain analysts do not typically describe services down to the level of details required for non-trivial automated reasoning.
The third group involves approaches that have formal background and give opportunity not only to specify behavior, but also to analyze given compositions, regardless whether they are component-based or service-oriented. In [26] Rychlý formally describes service behavior and structure in SOA as a CBS systems with features of dynamic and mobile architectures. He uses π-calculus formalism for service specification, definition of individual component behavior, and their composition into a hierarchically structured CBS that implements service behavior. In comparison with our work introduced in this paper [26] is more concerned for the behavior in terms of interfaces and inner subcomponent/subservice communication and bindings, while we give more detailed description involving not only this type of behavioral characteristics, but also actions and resources.

7.5 Conclusions and Future Work

In this paper, we present the commonalities and differences between CBSE and SOSE, by pointing out the relation between the architectural view of CBS and SOS and component/service (internal) behavioral view. The language Remes has already been used for CBS behavioral modeling and analysis, by translating Remes models to the PTA framework, and analyzing them within the Uppaal [2] and Cora[3] tools. To provide support for reliability analysis too, Remes should be added with probabilistic constructs, such as edges annotated with probabilities. Moreover, we have enumerated some ways of extending Remes to support service-oriented contexts. As future work, we plan to provide concrete representation of: service failure report, service discovery mechanisms, support for negotiation, and reliability prediction in the existing modeling language Remes.

[2]http://uppaal.com/
[3]http://www.cs.aau.dk/ behrmann/cora/

Acknowledgements This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre Progress, and by the European Union under the ICT priority of the 7th Research Framework Programme in the context of the Q-ImPrESS research project (www.q-impress.eu).

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] 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.

[4] Vincenzo Grassi, Raffaela Mirandola, and Antonino Sabetta. From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 25–36, New York, NY, USA, 2005. ACM.

[5] Steffen Becker, Heiko Koziolek, and Ralf Reussner. Model-based performance prediction with the palladio component model. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, pages 54–65, New York, NY, USA, 2007. ACM.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] E. Bruneton, T. Coupaye, and J. Stefani. Recursive and dynamic software composition with sharing, 2002.

[12] Frantisek Plasil and Stanislav Visnovsky. Behavior protocols for software components. *IEEE Trans. Softw. Eng.*, 28(11):1056–1076, 2002.

[13] Gary T. Leavens and Krishna Kishore Dhara. Concepts of behavioral subtyping and a sketch of their extension to component-based systems. pages 113–135, 2000.

[14] 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.

[15] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

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

[17] H. Hansson, I. Crnkovic, and T. Nolte. The world according to progress. *Draft paper*, 2008.

[18] Ivica Crnkovic, Michel Chaudron, and Stig Larsson. Component-based development process and component lifecycle, pages. *Journal of Computing and Information Technology*, 13(4):321–327, November 2005.

[19] Phillipe Collet, Thiery Coupaye, Herve Chang, Lionel Seinturier, and Guillaume Dufrene. Components and services: A marriage of reason. Technical Report ISRN I3S/RR-2007-17-FR, CNRS, May 2007. Project RAINBOW.

[20] David Garlan and Mary Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, pages 1–39, Singapore, 1993. World Scientific Publishing Company.

[21] Séverine Sentilles, Aneta Vulgarakis, Tomáš Bureš, Jan Carlson, and Ivica Crnkovic. A component model for control-intensive distributed embedded systems. In *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*, pages 310–317. Springer Berlin, October 2008.

[22] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. *Theor. Comput. Sci.*, 318(3):297–322, 2004.

[23] Tom Henzinger, Shaz Qadeer, and Sriram K. Rajamani. Decomposing refinement proofs using assume-guarantee reasoning. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*, pages 245–252, January 2000.

[24] Rob C. van Ommering. Koala, a component model for consumer electronics product software. In *Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families*, pages 76–86, London, UK, 1998. Springer-Verlag.

[25] UML 2.0 Superstructure Specification. Technical report, Object Management Group (OMG), August 2005.

[26] 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.

Chapter 8

Paper C: Modeling and Reasoning about Service Behaviors and their Compositions

Aida Čaušević, Cristina Seceleanu, Paul Pettersson
In 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

Abstract

Service-oriented systems have recently emerged as context independent component-based systems. Unlike components, services can be created, invoked, composed, and destroyed at run-time. Consequently, all services need a way of advertising their capabilities to the entities that will use them, and service-oriented modeling should cater for various kinds of service composition. In this paper, we show how services can be formally described by the resource-aware timed behavioral language REMES, 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. Assuming a Hoare-triple model of service correctness, we show how to check it by using the strongest postcondition semantics. To provide means for connecting REMES services, we propose a hierarchical language for service composition, which allows for verifying the latter's correctness. The approach is applied on an abstracted version of an intelligent shuttle system.

8.1 Introduction

Service-oriented systems (SOS) assume *services* as their basic functional units, with capabilities of being published, invoked, composed and destroyed at runtime. Services are loosely coupled and enjoy a higher level of independence from implementation specific attributes than components do.

An important problem is to ensure the *quality-of-service* (QoS) that can be expected when deciding which service to select out of a number of available services delivering similar functionality. Some of the existing SOS standards support formal analysis [1–4] to ensure QoS, but usually it is not straightforward to work out the exact formal analysis model.

In order to fully understand the ways in which services evolve and impact on QoS attributes, a *service behavioral description* is required [5]. Such behavior is assumed to be internal to the service, and hidden from the user. It should include the representation of a service functionality, enabled actions, resource annotations, and possible interactions with other services.

To meet the above demands, in this paper, concretely in Section 8.3, we extend the existing resource-aware, timed hierarchical language Remes [6], recalled in Section 8.2, to become fit for service behavioral modeling. In Remes, a service is modeled by an atomic or composite *mode*, which we enrich with attributes such as service type, capacity, time-to-serve etc., pre- and postconditions, which are exposed at the mode's interface. Still in Section 8.3, we introduce a synchronization mechanism for Remes modes, which enables modeling and verification of synchronized services.

By exploiting the pre-, postcondition annotations, we show how to describe the service behavior in Dijkstra's guarded command language [7], and how to check the service correctness by employing Dijkstra's and Scholten's strongest postcondition semantics [8].

Since services can be composed at run-time, analyzing the correctness of a service in isolation does not suffice. To exemplify, let us consider a service that is composed of several navigation services, out of which some return the route length in miles, whereas others in kilometers. If the developer has omitted to introduce a service that converts length from one metric to the other, it is desirable to uncover such an error right away, by formally checking the correctness of the actual service composition, at run-time.

To address the dynamic aspects of services, in Section 8.4, we propose a hierarchical language for dynamic service composition (HDCL) that allows creating new services, via binary operators, as well as adding and/or deleting services from lists. In the same section, we also give the semantics of sequential, parallel, and parallel with synchronization service composition, respectively. Next, we apply the approach on an abstracted version of an intelligent shuttle system, for which we show the use of REMES language to model the system and apply HDCL language to check the correctness of service compositions. In Section 8.6, we compare to some of the relevant related work, before concluding the paper in Section 8.7.

8.2 Preliminaries

8.2.1 REMES modeling language

The REsource Model for Embedded Systems REMES [6] is intended as a meaningful basis for modeling and analysis of resource-constrained behavior of embedded systems. REMES provides means for modeling of both continuous (i.e., power) and discrete resources (i.e., memory access to external devices). REMES is a state-machine behavioral language that supports hierarchical modeling, continuous time, and a notion of explicit entry and exit points, making it fit for component-based system modeling.

To enable formal analysis, REMES models can be transformed into timed automata (TA) [9], or priced timed automata (PTA) [10], depending on the analysis type.

The internal component behavior in REMES is given in terms of modes that can be either *atomic* (do not contain submode(s)), 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). 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.

The control flow is given by the set of directed lines (i.e., *edges*) that connect the control points of (sub)modes. Modes may also be annotated with *invariants*, which bound from above the current mode's delay/execution time. For a more thorough description of the REMES model, we refer the reader to [6].

The REMES language benefits from a set of tools[1] for modeling, simulation and transformation into PTA, which could assist the designer during system development.

8.2.2 Guarded command language

The Guarded Command Language (GCL) was introduced and defined by Dijkstra for predicate transformers semantics [7]. The basic element of the language is the guarded command, a statement list prefixed by a boolean expression, which can be executed only when the boolean expression is initially true.

The syntax of the GCL is given in Backus-Naur Form (BNF) extended with braces "{..}", where the braces mean: "followed by zero or more instances of the enclosed".

< guarded command >	::=	< guard > − > < guarded list >
< guard >	::=	< boolean expression >
< guarded list >	::=	< statement > {; < statement >}
< guarded command set >	::=	< guarded command > { ▯ < guarded command >}
< alternative construct >	::=	**if** < guarded command set > **fi**
< statement >	::=	< alternative construct > | "other statements"
< repetitive construct >	::=	**do** < guarded command set > **od**

The semicolons in the guarded list denote that whenever the guarded list is selected for execution, its statements will be executed successively in the order from the left to the right. A guarded command is not a statement but a component of a guarded command set from which statements can be constructed. The separator " ▯ " is used for mutual separation of guarded commands in guarded command set.

The alternative construct is written using special bracket pair: "**if ... fi**". The program aborts if none of the guards is true, otherwise an arbitrary guarded list with a true guard will be executed. Similarly, the repetitive construct "**do ... od**" means that the program runs as long as one of the guards is true, and terminates if none of the guards is true.

Semantics and Correctness of Guarded Commands. Let us assume the Hoare triple, {p} S {q}, where p, q are predicates, denoting the partial correctness of guarded command S with respect to precondition p and postcondition q. Introduced by Dijkstra and Sholten [8],

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

the *strongest postcondition predicate transformer* (a function that maps predicates to predicates), denoted by $\mathsf{sp.S.p}$, holds in those final states for which there exists a computation controlled by S, which belongs to class “initially p”. Proving the Hoare triple, that is, the correctness of a guarded command, reduces to showing that ($\mathsf{sp.S.p} \Rightarrow \mathsf{q}$) holds. The strongest postcondition rules for the assignment statement, for sequential composition, and for the non-deterministic conditional are as follows:

$$\mathsf{sp.(x := e).p(x)} \equiv \mathsf{x = e} \wedge (\exists \mathsf{x} \cdot \mathsf{p(x)}) \tag{8.1}$$

$$\mathsf{sp.(S_1;\ S_2).p} \equiv \mathsf{sp.S_2.(sp.S_1.p)}, \forall \mathsf{p} \tag{8.2}$$

$$\mathsf{sp.(if\ g_1 \rightarrow S_1\ [\!]\ \ldots\ [\!]\ g_n \rightarrow S_n\ fi).p} \equiv \mathsf{sp.S_1.(g_1 \wedge p)} \vee \ldots \vee \mathsf{sp.S_n.(g_n \wedge p)}, \forall \mathsf{p} \tag{8.3}$$

8.3 Behavioral Modeling of Services in REMES

In REMES, a service is represented by a mode (be it atomic or composite). The service may have a special Init entry point, visited when the service first executes, and where all variables are initialized. In order for a service to be published and later discovered, a list of attributes should be exposed at the interface of a REMES mode/service (see Fig.8.1).

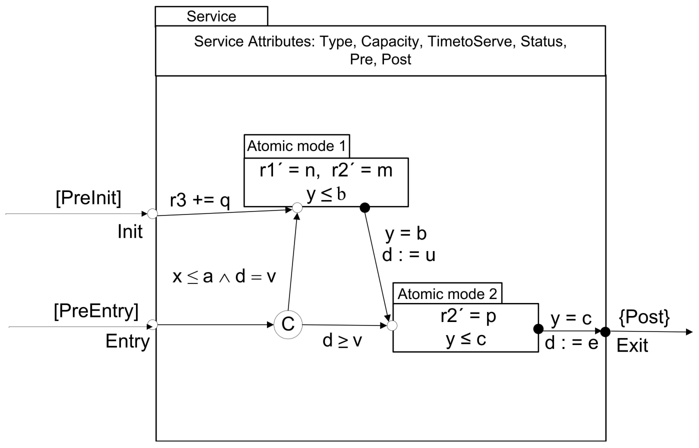

Figure 8.1: A service modeled in REMES

The attributes depicted in Fig.8.1 have the following meaning:

- service type - specifies whether the given service is a web service (i.e., weather report), a database service (i.e., ATM services), a network service, etc.;
- service capacity - specifies the service's maximum ability to handle a given number of messages per time unit (i.e., the maximum service frequency)($\in \mathsf{N}$);
- time-to-serve - specifies the worst-case time needed for a service to respond and serve a given request ($\in \mathsf{N}$);
- service status - describes the current service status (that is, passive (not invoked), idle, active);
- service precondition - is a predicate ($\mathsf{Pre} : \sum \rightarrow \mathsf{Bool}$, $\mathsf{Pre} \equiv (\mathsf{PreInit} \vee \mathsf{PreEntry})$) that conditions the start of service execution, and must be true at the time a REMES service is invoked. In this expression $\sum$ is the polymorphic type of the state that includes both local and global variables, and predicates PreInit, PreEntry are the initial, and the entry precondition of the service, respectively;
- service postcondition - is a predicate (Post) that must hold at the end of a REMES service execution.

The attributes are used to discover Service; they are specified by an interested party and, based on the specification, the service is either retrieved or not.

The formal specification of a service, modeled as the composite mode of Fig. 8.1, is the Hoare triple $\{p\}\mathsf{Service}\{q\}$, where Service is described in terms of the guarded command language, and the mode's precondition p, and postcondition (requirement) q are as follows:

$$p$$
$$\equiv$$
$$\mathsf{y \leq c \wedge c > b \wedge (d = 0 \vee v \leq d \leq e) \wedge r1 = r2 = r3 = 0 \wedge (h = 0 \vee h = 1)}$$
$$q$$
$$\equiv$$
$$\mathsf{y \leq c \wedge d \leq e \wedge (\forall i, 1 \leq i \leq 3 \cdot ri \leq val_i)}$$

where val_i are the given upper bounds on each resource usage, respectively.

Below, we give the GCL description of the REMES composite mode Service:

```
Service ::=
IF
$\neg u1 \wedge h = 0 \wedge y \leq b$                                    Init $\rightarrow$ Atomic mode 1
   $\rightarrow r3 := r3 + q$;
      $sm :=$ Atomic mode 1; $u1 :=$ true;
      Update(now)
[] $\neg u2 \wedge h = 1 \wedge (x \leq a \wedge d = v) \wedge y \leq b$             Entry $\rightarrow$ Atomic mode 1
   $\rightarrow sm :=$ Atomic mode 2; $u2 :=$ true;
      Update(now)
[] $(\neg u3 \wedge (h = 1 \wedge d \geq v) \vee d = u) \wedge y \leq c$    (Entry or Atomic mode 1) $\rightarrow$ Atomic mode 2
   $\rightarrow sm :=$ Atomic mode 2; $u3 :=$ true;
      Update(now)
[] $\neg u4 \wedge sm =$ Atomic mode 1 $\wedge y \leq b$                       Delay in Atomic mode 1
   $\rightarrow r1(t) := r1(now) + n * (t - now)$;
      $r2(t) := r2(now) + m * (t - now)$;
      $\{y \leq b\}$; $u4 :=$ true;
      Update(now)
[] $\neg u5 \wedge sm =$ Atomic mode 1 $\wedge y = b$
   $\rightarrow d := u$; $u5 :=$ true;
      Update(now)
[] $\neg u6 \wedge sm =$ Atomic mode 2 $\wedge y \leq c$                       Delay in Atomic mode 2
   $\rightarrow r2(t) := r2(now) + p * (t - now)$;
      $\{y \leq c\}$; $u6 :=$ true;
      Update(now)
[] $\neg u7 \wedge sm =$ Atomic mode 2 $\wedge y = c$                          Atomic mode 2 $\rightarrow$ Exit
   $\rightarrow d := e$;
      $h := 1$; $u7 :=$ true;
      Update(now); $u1, \ldots, u7 :=$ false
FI
                                                                                          (8.4)
```

In the GCL description (8.4), the variables x, y are clocks, h is the history variable that is used to decide where to enter the composite mode, sm is the variable ranging over submodes, and $r1 : \mathsf{Real}_{+} \rightarrow \mathsf{T}_1$, $r2 : \mathsf{Real}_{+} \rightarrow \mathsf{T}_2$ are the continuous resources of the model, defined as functions over the non-negative reals that are used as the time domain. In addition, u_i are local variables used for preventing executing the same action more than once, at the same time point. These variables are reset each time the mode Service exits. Similar to the approach taken for action system models [11], the variable now shows the current time, and it is explicitly updated by statement Update(now). The assertions $\{y \leq b\}, \{y \leq c\}$ model the invariants (Inv) of Atomic mode 1, and Atomic mode 2, respectively.

We define $\mathsf{Update}(\mathsf{now})$ as follows:

$$Update(now) \triangleq now := \mathsf{next}.now$$

The submodes can be urgent (no delays are allowed), or non-urgent (where delays can happen, until an invariant Inv is violated); also, guarded actions can annotate edges connecting the entry points of the composite mode with submodes, via some conditional connector (denoted by encircled C in Figure 8.1). Given these, and assuming that gg is the disjunction of the action guards of the edges leaving a mode (or a conditional connector), and that Inv is the invariant of the respective mode, next is defined by:

$$\mathsf{next}.t \quad \triangleq \quad \begin{cases} min\{t' \geq t \ \mid \ \neg Inv \vee gg\}, & \text{if exists } t' \geq t \text{ such that } \neg Inv \vee gg \\ +\infty, & \text{otherwise.} \end{cases}$$

If a mode is urgent, or the guards correspond to a conditional connector, then $I \equiv \mathsf{false}$, so the next moment of time is identical to the current one, no delay being possible.

The mode $\mathsf{Service}$, modeled by (8.4), can be iterated for as long as needed, so the complete specification is:

$$\mathsf{status_{Service}} := \mathsf{active};\ (\mathsf{DO\ g} \rightarrow \mathsf{Service}\ \ [\!]\ \neg\mathsf{g} \rightarrow \mathsf{status_{Service}} := \mathsf{idle\,OD}).$$

According to rule (8.3), the strongest postcondition of the conditional statement is:

$$\begin{array}{ll} & sp.Service.p \\ \equiv & \\ & sp.(r3 := r3 + q;\ sm := \text{Atomic mode 1};\ \mathsf{Update(now)}).(h = 0 \wedge y \leq b \wedge p) \\ & \vee \\ & \ldots \\ & \vee \\ & sp.(d := e;\ h := 1;\ \mathsf{Update(now)}).(sm = \text{Atomic mode 2} \wedge y = c \wedge p) \end{array}$$

Assuming that $\mathsf{sp}.\{\mathsf{y} \leq \mathsf{c}\}.\mathsf{p} \equiv y \leq c \wedge (\exists \mathsf{x} \cdot \mathsf{p}(\mathsf{x}))$, the above sp can be mechanically computed by successively applying rules (8.1) - (8.2). The correctness proofs reduce to checking whether each of the strongest postconditions of the above disjunction implies the requirement q, given earlier.

In service-oriented systems, there is often the case that services need to synchronize their behaviors. In order to model synchronized behavior, we introduce a special kind of REMES mode, given in Figure 8.2, which can act either as an AND mode, or as an OR mode, depending on whether the services need to be entered simultaneously, or not.

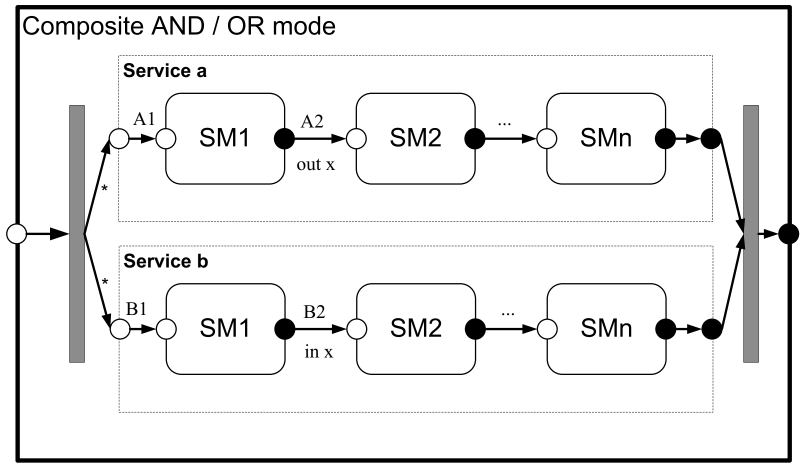

Figure 8.2: AND/OR REMES mode.

The composite mode of Figure 8.2 contains as sub-modes the services that need to be synchronized. For AND modes, both Service a, and Service b are entered at the same time (through their entry point). This means that the edges marked with (*) do not have guards. In case of OR modes, one or all constituent services are entered, so the edges marked with (*) are annotated with guards. If some of the edges need to be taken at the same time in both services, the communication between Service a and b is realized via synchronization variables, chan (in x), (out x), which are used similarly to the PTA channels $x?, x!$, respectively. Depending on the required synchronization type and starting time of the constituent services' execution, AND modes, but also OR can be employed when either "**and**" synchronization (both services should finish execution at the same time), or "**max**" synchronization (the composite mode finishes when the slowest service finishes) is required.

In Figure 8.2, Service a, Service b need to synchronize actions $A2$, $B2$. This can be done by decorating the respective edges with channel variables *out x* for $A2$, and *in x* for $B2$, meaning that the respective edges are taken simultaneously in both services, $A2$ writing variables that $B2$ is reading. The same applies if the services need to "and"-synchronize at the end of their execution. The exit edge of each service, respectively, needs to be annotated with chan variables.

The GCL representation of such synchronization requires strengthening the guards of the respective synchronized commands of the conditional statement, as follows: $(in\ x) \wedge g_{A2} \rightarrow S_{A2}$, $(out\ x) \wedge g_{B2} \rightarrow S_{B2}$, where S_{A2}, S_{B2} are the action bodies of $A2$, $B2$, respectively. The actions can then be executed in a sequence, with the one writing variables, first. The "**max**" synchronization can be represented in GCL by using a virtual selector (variable *sel*) [11], which selects for execution the modes $SM1, \ldots, SMn$, according to the control flow, marks them as executed after they finish their execution, and keeps the time values of *now* in a copy variable now_c, which is updated only after the slowest service finishes executing; the latter translates in exiting the composite AND, or OR mode.

8.4 Hierarchical Language for Dynamic Service Composition: Syntax and Semantics

Service compositions may lead to complex systems of concurrently executing services. An important aspect of such systems is the correctness of their temporal and resource-wise behavior. In the following, we propose an extension to the REMES language, which provides means to define and support creation, deletion, and composition of fine-grained or coarser-grained services, applicable to different domains. We also investigate a formal way of ensuring the correctness of the composition, based on the strongest postcondition semantics of services.

Let us assume that a service, whose behavior is described by a REMES mode, is denoted by $service\_name_i$, $i \in [1..n]$; then, a service list, denoted by *s_list*, is defined as follows:

$$\mathsf{s\_list} ::= [\mathsf{service\_name}_1, \ldots, \mathsf{service\_name}_n]$$

In order to support run-time service manipulation, we define a set of REMES interface operations, by a pre- postcondition specification. We denote by Σ the set of service states, respectively, that is, the current collection of variable values.

- **Create service**: *create service_name*
  [*pre*] : *service_name* = NULL

create : $Type \times\ N \times N \times\ ''passive'' \times (\Sigma \to bool) \times (\Sigma \to bool) \to service\_name$
$\{post\}$: $service\_name \neq$ NULL

- **Delete service**: *del service_name*
  $[pre]$: $service\_name \neq$ NULL
  del : $service\_name \to$ NULL
  $\{post\}$: $service\_name =$ NULL

- **Create service list**: *create s_list*
  $[pre]$: $s\_list =$ NULL
  create_list : $s\_list \to s\_list,\ s\_list = List()$
  $\{post\}$: $s\_list \neq$ NULL

- **Delete service list**: *del s_list*
  $[pre]$: $s\_list \neq$ NULL
  del_list : $s\_list \to$ NULL
  $\{post\}$: $s\_list =$ NULL

- **Add service to a list**: *add service_name, s_list*
  $[pre]$: $service\_name \notin s\_list$
  add : $s\_list \to s\_list$
  $\{post\}$: $service\_name \in s\_list$

- **Remove service from the list**: *del service_name, s_list*
  $[pre]$: $service\_name \in s\_list$
  del : $s\_list \to s\_list$
  $\{post\}$: $service\_name \notin s\_list$

- **Replace service in the list**: *replace* $service\_name_1$, $service\_name_2$
  $[pre]$: $s\_list(p) = service\_name_1$
  replace : $s\_list \to s\_list$
  $\{post\}$: $s\_list(p) = service\_name_2$

- **Insert service at a specific position**: *insert* $service\_name_i$, s_list
  $[pre] : s\_list(p) \neq service\_name_i$
  $\mathsf{add} : s\_list \rightarrow s\_list$
  $\{post\} : s\_list(p) = service\_name_i$

Note that a new service list can be created by using the constructor *List()*, which holds list values of any type. Such a constructor enables the creation of both empty list and also list with some initial value:

$$s\_list = \mathsf{List} : \mathsf{String}([``\texttt{Shuttle1}", ``\texttt{Shuttle2}"]).$$

Also, adding a service to a list means, in this context, appending that service, that is, adding it at the end of the list. Replacing a service with another one, and inserting a service at a specific position requires the use of parameter p, which specifies the position at which the service is replaced or inserted.

Most often, services can be perceived as independent and distributed functional units, which can be composed to form new services. The systems that result out of service composition have to be designed to fulfill requirements that often evolve continuously and therefore require adaptation of the existing solutions.

Alongside the above operations, we also define a hierarchical language that supports dynamic REMES service composition (HDCL), that is, facilitates modeling of nested sequential, parallel or synchronized services:

$$\mathsf{DCL} \quad ::= \quad (\mathsf{s\_list}, \mathsf{PROTOCOL}, \mathsf{REQ})$$

$$\mathsf{HDCL} \quad ::= \quad (((\mathsf{DCL}^+, \mathsf{PROTOCOL}, \mathsf{REQ})^+, \mathsf{PROTOCOL}, \mathsf{REQ})^+, \ \ldots)$$

The formula above allows a theoretically infinite degree of nesting. The positive closure operator is used to express that one or more DCLs (Dynamic Composition Languages) are needed to form an HDCL. The PROTOCOL defines the way services are composed, that is, the type of binding between services, as follows:

$$\mathsf{PROTOCOL} \quad ::= \quad \mathsf{unary\_operator}\ service\_name \ \mid \ service_m\ \mathsf{binary\_operator}\ service_n$$

The requirement REQ is a predicate ($\Sigma \rightarrow \mathsf{Bool}$) that can include both functional and extra-functional properties/constraints of the composition. It identifies the required attribute constraints, capability, characteristics, or quality of a system, such that it exhibits the value and utility requested by the user. The above unary and binary operators are defined as follows:

$$\begin{array}{lll} \mathsf{Unary\_operator} & ::= & \mathsf{exec-first} \\ \mathsf{Binary\_operator} & ::= & ; \quad | \quad \| \quad | \quad \|_{SYNC-and} \quad | \quad \|_{SYNC-or} \end{array}$$

Let us assume that two services s_1, s_2 are invoked at some point in time, and their instances are placed in the service list s_list. Also, we assume that $s_i.Pre_i$ is the strongest postcondition of s_i, $i \in 1, 2$, w.r.t. precondition Pre_i. Then, the semantics of the unary and binary protocol operators, as well as the correctness conditions for such compositions are given as follows.

- **Exec-first** (specifies which service should be initially executed in a composition) - below we formalize the fact that s_1 should execute first, and only when it finishes and establishes its postcondition, service s_2 can become active:

  $$status_{s_1} = active \wedge status_{s_2} = idle \wedge Post_{s_1} \Rightarrow (status_{s_2} = active)$$

  If we assume n services $s_1, \ldots, s_n$ of a list, executing s_1 first is defined as:

  $$\mathsf{Exec-first\ s_1} \triangleq \mathsf{s_1} \; [\!] \; \neg \mathsf{g_{s_1}} \rightarrow (\mathsf{s_2\ Binary\_operator} \ldots \mathsf{Binary\_operator\ s_n})$$

  This means that, even if any other service (or service composition) could be executed, it will be executed only after s_1 has finished execution.

- **Sequential composition** - two services are executed in a sequence, uninterrupted, e.g., $s_1;\ s_2$. The correctness condition of $s_1;\ s_2$ is:

  $$(sp.s_2.(sp.s_1.Pre_{s_1}) \Rightarrow Post_{s_2}) \quad \wedge \quad (Post_{s_2} \Rightarrow REQ)$$

- **Parallel composition's** ($s_1 \parallel s_2$) correctness condition is:

  $$(sp.s_1.Pre_{s_1} \vee sp.s_2.Pre_{s_2}) \Rightarrow REQ$$

- **Parallel composition with synchronization** - we denote by S-AND the set of services belonging to an AND mode, which need to synchronize their executions in the end. Then, the "and" synchronization of such services is defined as:

$$\begin{aligned}(s_1 \parallel_{SYNC-and} s_2) \triangleq\ & (s_1, s_2 \in \mathsf{S{-}AND} \\ & \Rightarrow ((\forall\, now \cdot status_{s_1} = status_{s_2} = active) \\ & \wedge (start_{s_1} + TimetoServe_{s_1} = start_{s_2} + TimetoServe_{s_2})))\end{aligned}$$

The correctness condition of the "and-AND" synchronization is given below:

$$(sp.(s_1 \parallel_{SYNC-and} s_2).Pre_{AND} \Rightarrow (Post_{s_1} \wedge Post_{s_2})) \wedge (Post_{s_1} \wedge Post_{s_2} \Rightarrow REQ)$$

A service user, but also a developer of services, might need to replace a service with one with possibly better QoS. It follows that one needs to be able to check whether the new service still delivers the original functions, while having better time-to-serve or resource-usage qualities. Verifying such a property reduces to proving refinement of services. Either weakening the service precondition or strengthening its postcondition qualifies as service refinement.

8.5 Example: An Autonomous Shuttle System

In this section, we consider an example, previously modeled and analyzed in the PTA framework, in our recent work [12].

We consider a simplified version of a three train system that provides transportation service to three different locations. The system has been developed at University of Paderborn within the Railcab project [13]. While in our previous work [12], we have focused on resource effective design, in the current example, we extract parts of the behavior described by Giese and Klein [13], to show how services are created, invoked, composed, and idled, by using the REMES extended interface and behavioral language.

Each of the trains has a well-defined path to follow, as shown in Fig. 8.3. During the transport, the shuttles might meet at point B, in which they are forced to create a convoy. In order to enter the convoy, they have to respect given speed and acceleration limits, measured in points A1, A2, and A3, respectively, otherwise they may stop to let others that fulfil the given requirements join the convoy. After a convoy is formed and has left, those that were stopped are allowed to continue their journey to previously assigned destination, if the sensor at point C, in Fig. 8.3, has sent the "safe to continue" signal.

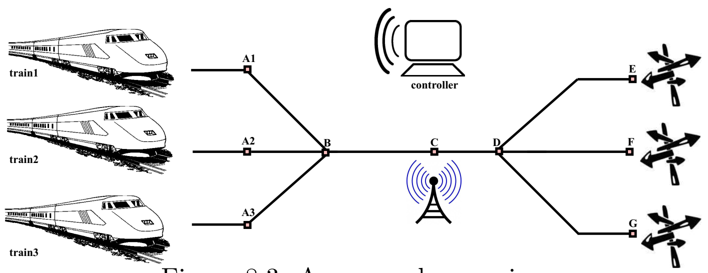

Figure 8.3: An example overview.

After the destination point is being reached, a shuttle is free to turn to the idle state, and wait for new orders. The system described above is equipped with one central controller, as shown in Fig. 8.3, which decides when and which shuttle to invoke, based on the service descriptions for each shuttle, respectively.

8.5.1 Modeling the Shuttle System in Remes

We model the behavior of the Autonomous shuttle system services as modes in the extended Remes. The composite mode of Shuttle1 is depicted in Fig. 8.4, yet, due to lack of space, we do not show here the constituent submodes, but we briefly explain them instead (for more details we refer reader to [12]).

The mode consists of the *atomic* modes (i.e., Acceleration1, STOP, and Destination). They communicate data between each other using the global variables: speed_i, status_i, t_i, and StatusConvoy. The control interfaces are used to expose mode attributes relevant for mode discovery. Shuttle1 and Shuttle3 have the same behavior, while Shuttle2 is an older shuttle than the other two, and therefore it requires more time to start, accelerate, slow down.

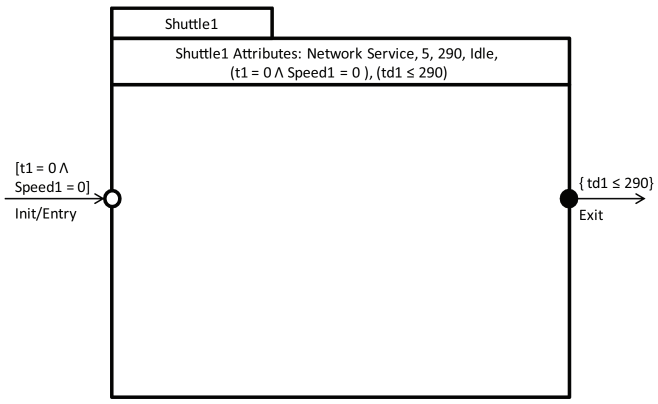
Shuttle1
Shuttle1 Attributes: Network Service, 5, 290, Idle, (t1 = 0 ∧ Speed1 = 0), (td1 ≤ 290)
[t1 = 0 ∧ Speed1 = 0]
Init/Entry
{ td1 ≤ 290}
Exit

Figure 8.4: The model of Shuttle1 given as a REMES service.

8.5.2 Applying the Hierarchical Language

Below, we illustrate the use of our proposed hierarchical language for modeling service composition, as depicted in Table 8.1, on the example described in Section 8.5.

Table 8.1: An illustration of the REMES language

00 **declare** Shuttle1 $::=<$ network service,	18 **create** Shuttle1
01 5,	19 **create** Shuttle2
02 290,	20 **create** Shuttle3
03 passive,	21 **create** list_Convoy
04 $(t1 = 0 \wedge speed = 0)$,	22 **add** Shuttle1 list_Convoy
05 $(t1 \leq 290) >$	23 **add** Shuttle2 list_Convoy
06 **declare** Shuttle2 $::=<$ network service,	24 **DCL**_Convoy $::=$ (list_Convoy, ; , $t \leq 300$)
07 7,	25 **HDCL**_Convoy $::=$ ((DCL_Convoy, Shuttle3), , $t \leq 300$)
08 300,	26 **check**(sp.(Shuttle1; Shuttle2).$(t1 = 0 \wedge speed = 0) \wedge (t = t1 \vee t = t2)) \Rightarrow (t \leq 300)$
09 passive,	27 **check**(sp.Shuttle3.$(t3 = 0 \wedge speed = 0) \wedge (t = t3)) \Rightarrow (t \leq 300)$
10 $(t2 = 0 \wedge speed = 0)$,	28 **del** HDCL_Convoy
11 $(t2 \leq 300) >$	
12 **declare** Shuttle3 $::=<$ network service,	
13 5,	
14 290,	
15 passive,	
16 $(t3 = 0 \wedge speed = 0)$,	
17 $(t3 \leq 290) >$	

The needed services are introduced through the declarative part (lines 00-17 in Table 8.1). A service declaration contains the service name, type, status, TimeToServe, precondition and postcondition. The corresponding requirement is matched against such attribute information, when choosing a service. After the selection, the instances of the selected services are created (lines 18-20 in Table 8.1), and added to the service list using the *add* command (lines 22-23 in Table 8.1). Finally, the chosen services are composed by DCL. The list of services, employed protocol (type of service binding), and DCL requirements are given as parameters. Moreover, the language provides means to compose the existing DCLs with other services, through HDCL, as shown in line 25 of Table 8.1. If not anymore needed, the composition can be deleted.

The advantage of this language is that, after each composition, one can check 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. Due to space limitation, we show only the final computed result. Below, $\mathsf{p1} \equiv (\mathsf{t1} = 0 \wedge \mathsf{speed} = 0)$.

By applying the sp rules (8.1) - (8.3), we get the following:

$$\begin{array}{lcl}
\mathsf{sp.(Shuttle1;\ Shuttle2).p1} & \equiv & \mathsf{sp.Shuttle2.(sp.Shuttle1.p1)} \\
\mathsf{sp.Shuttle1.p1} & \equiv & (\mathsf{t1} = 0 \wedge 245 \leq \mathsf{t} \leq 266 \wedge \mathsf{speed1} = 0 \wedge \\
 & & \wedge\ \mathsf{mode} = \mathsf{Destination} \wedge \mathsf{r1} = 0 \wedge \\
 & & \wedge\ \mathsf{status1} = \mathsf{end1} = \mathsf{idle}) \\
\mathsf{sp.Shuttle2.(sp.Shuttle1.p1)} & \equiv & (\mathsf{t1} = \mathsf{t2} = 0 \wedge 264 \leq \mathsf{t} \leq 285 \wedge \\
 & & \wedge\ \mathsf{speed1} = \mathsf{speed2} = 0 \wedge \mathsf{r1} = \mathsf{r2} = 0 \wedge \\
 & & \wedge\ \mathsf{status1} = \mathsf{end1} = \mathsf{idle} \wedge \mathsf{status2} = \mathsf{end2} = \mathsf{idle})
\end{array}$$

One can notice that the requirement $\mathsf{REQ} \equiv (\mathsf{t} \leq 300)$ is implied by the calculated strongest postcondition to which the condition $(\mathsf{t} = \mathsf{t1} \vee \mathsf{t} = \mathsf{t2})$ is added. This is actually what the command check should return as a main proof obligation, provided that the method is implemented in the Remes tool-chain.

Next, we have Shuttle3 composed in parallel with the sequential composition of the other two shuttles, with $\mathsf{p3} \equiv (\mathsf{t3} = 0 \wedge \mathsf{speed} = 0)$. Then, according to the composition semantics of section 8.4, proving the correctness of the (Shuttle3 (Shuttle1; Shuttle2)) composition reduces to showing that:

$$(\mathsf{sp.Shuttle3.p3} \vee \mathsf{sp.Shuttle2.(sp.Shuttle1.p1)}) \Rightarrow \mathsf{REQ}$$

As already shown, the sequential composition of the first two shuttles implies the requirement. What is left to be proven is that the strongest postcondition of Shuttle3, w.r.t. p3, also implies the requirement. The calculated strongest postcondition of the latter is as follows:

$$\mathsf{sp.Shuttle3.p3} \equiv (\mathsf{t3} = 0 \wedge 245 \leq \mathsf{t} \leq 266 \wedge \mathsf{speed3} = 0 \wedge$$
$$\wedge\ \mathsf{mode} = \mathsf{Destination} \wedge \mathsf{r3} = 0 \wedge \mathsf{status3} = \mathsf{end3} = \mathsf{idle})$$

It is easy to check that the requirement REQ is actually implied by $\mathsf{sp.Shuttle3.p} \wedge \mathsf{t} = \mathsf{t3}$. This concludes our service composition correctness verification.

8.6 Discussion and Related Work

Based on the level of details that are provided through the behavioral description, all approaches related to services and SOS can be in principle divided into three groups.

Code-level behavioral description approaches are mostly based on XML language (e.g., BPEL, WS-CDL). BPEL [1] 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 [3] 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.

The third group includes approaches with formal background. Rychlý describes the service behavior as a component-based system for dynamic architectures [14]. 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, while we can also cater for service descriptions including timing and resource annotations [12]. Foster et al. present an approach for modeling and analysis of web service compositions [15]. The approach takes BPEL4WS service specification and translates it into Finite State Processes (FSP), and Labeled Transition Systems (LTS), for analysis purposes.

A comprehensive survey on several approaches that are accommodating service composition, and are checking the correctness of compositions [1–4] is given by Beek et al. [16]. Regarding service modeling, all these approaches are solid; however, w.r.t. service compositions and their correctness checking [17–19] (usually by employing formal methods), such approaches show limited capabilities to automatically support these processes. In comparison, as shown in this paper, 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 [9] or PTA [10], and analyzed with UPPAAL , or CORA tools [2], for functional but also extra-functional behaviors (timing and resource-wise behaviors).

8.7 Conclusions

In this paper, we have presented an approach for formal service description by extending the resource-aware timed behavioral language REMES. Attributes such as type, time-to-serve, capacity, etc., together with precondition and postcondition are added to REMES to enable service discovery, as well as service interaction. Even if the original semantics of REMES [6] is given in terms of Priced Timed Automata (PTA), here, we have chosen to use Hoare triples and the strongest postcondition semantics to prove service correctness, motivated by the lack of decidability results for computing simulations relations on PTA. We have also proposed a hierarchical language for service composition, which allows for the verification of, e.g., service composition correctness. The approach is demonstrated on a simplified version of an intelligent shuttle system.

As future work, we plan to look into the algorithmic computation of strongest postconditions of priced timed automata, by building on preliminary results of Badban et al. [20]. We also intend to extend the REMES tool-chain with a postcondition calculator.

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

Bibliography

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

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

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

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

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

[11] Cristina Seceleanu. *A Methodology for Constructing Correct Reactive Systems*. PhD thesis, Turku Centre for Computer Science (TUCS), December 2005.

[12] 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.

[13] Holger Giese and Florian Klein. Autonomous shuttle system case study. In *Scenarios: Models, Transformations and Tools*, pages 90–94, 2003.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Bahareh Badban, Stefan Leue, and Jan-Georg Smaus. Automated predicate abstraction for real-time models. *EPTCS*, 10:36, 2009.

Chapter 9

Paper D: Checking Correctness of Services Modeled as Priced Timed Automata

Aida Čaušević, Cristina Seceleanu, Paul Pettersson
(Submitted to conference)

Abstract

Service-oriented systems have gained importance in different application domains thanks to their ability to enable reusable functionality provided via well-defined interfaces, and the increased opportunities to compose existing units, called services, into various configurations. Developing applications in such a setup, by reusing existing services, brings some concerns regarding ensuring the expected quality, and correctness of the employed services. In this paper, we assume service models annotated with pre- and postconditions, their semantics given as priced timed automata (PTA), and the forward analysis method for checking the service correctness w.r.t. given requirements. For such models, we show how to algorithmically compute the strongest postcondition of the corresponding automata, with respect to the specified precondition. The approach is illustrated on a small example of services modeled as PTA.

9.1 Introduction

Service-oriented systems (SOS) are becoming one of the dominant paradigms for designing, implementing and developing large scale systems out of self-contained and loosely coupled services. Among the main benefits of the approach, the reusable functionality via well-defined interfaces, the service infrastructure equipped to enable services to be published, discovered, invoked, and if needed destroyed on-the-fly, and the fast application development by employing existing services are most appealing.

In systems built up in such a setup, it becomes essential to ensure a satisfying level of the system's quality-of-service (QoS). Sometimes, it might be the case that one needs to decide which service to select out of a number of available services that offer similar functionality. To deliver guarantees on provided QoS, some SOS approaches [1–4] support formal analysis, yet in most cases building the formal system model, out of formalized services, is far from straightforward.

However, once a model is created, it becomes essential to be able to check the correctness of the employed services, both in isolation, as well as in the context of the newly created system, which involves service compositions. An as important aspect, many times ignored, is the service's resource consumption. Any analysis approach that would abstract from service resource constraints would produce analysis results that are insufficiently correct, or reliable. For example, let us assume that we have a web payment service that includes several currency services, where some services return prices in Swedish Krona (SEK), some in US Dollar (USD), and some in Euros (EUR). If the developer has forgotten to introduce a service that converts prices from one available currency to the other, it is important to detect this error, by formally checking the correctness of the existing service composition, on-the-fly. Another scenario could assume that the system in which some service is incorporated has limited memory resources, which the respective service does not comply with (i.e., some service that draws the chart that displays currency change over some period of time).

To establish means for checking the correctness of services, in our recent work [5], we have presented an approach that extends the resource-aware timed behavioral language REMES, recalled in Section 9.2, with constructs for a formal service description. In the same work, we have shown how service correctness can be checked using Hoare triples, and strongest postcondition semantics, described in Section 9.3.1. However, such check is deductive, and it is not completely automated.

Since the original semantics of REMES [6] is given in terms of Priced Timed Automata (PTA), in this paper, concretely in Section 9.3, we go one step further, and present an algorithmic way to compute strongest postcondition of services modeled as PTA. We consider the service resource usage in REMES as a cost variable in PTA, and consequently we include in our algorithms well known approaches for computing the minimal and maximal reachability costs of a final PTA location [7], alongside calculating the strongest postcondition of reaching such location, over symbolic states. In Section 9.4, we exemplify our approach on a simple model of PTA. We compare our approach with some relevant work, in Section 9.5, before concluding the paper in Section 9.6.

9.2 Preliminaries

9.2.1 REMES modeling language

The REsource Model for Embedded Systems REMES [6] is initially intended as a meaningful basis for modeling and analysis of resource-constrained behavior of embedded systems. It provides means for modeling of both continuous (i.e., power) and discrete resources (i.e., memory access to external devices). REMES is a state-machine behavioral language that supports hierarchical modeling, continuous time, and a notion of explicit entry and exit points, making it fit for the modeling of component-based and service-oriented systems.

To enable formal analysis, REMES models can be transformed into timed automata (TA) [8], or PTA [9], depending on the analysis type.

In REMES services are represented in terms of modes that can be either *atomic* (do not contain submode(s)), 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). In order for a service to be published and later discovered, a list of attributes (i.e. service type, time-to-serve, service pre-, and postcondition, etc.) is exposed at the interface of a REMES mode. Based on the service attributes specification and a requirement given by a service user a service is either retrieved or not.

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.

The control flow is given by the set of directed lines (i.e., *edges*) that connect the control points of (sub)modes. REMES supports *delay/timed* actions and *discrete* actions. The former describe the continuous behavior of the mode, and their execution does not change the current mode; the latter, discrete actions (represented as edge annotations), when fired, result in a mode change. The delay/timed actions are not exposed in the model, but are constrained by the above mentioned differential equations. In order for a discrete action to be executed, the corresponding boolean *guard*, which prefixes the action body, must hold. Modes may also be annotated with *invariants*, which bound from above the current mode's delay/execution time.

It is often the case that services in SOS need to synchronize their behaviors. To model service synchronization a special kind of REMES mode is introduced. It can act either as an AND, or an OR mode, depending whether services have to be entered simultaneously or not. 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.

REMES language provides means for service creation, delation, and composition via REMES interface operations. To facilitate modeling of nested sequential, parallel or synchronized services, REMES includes a hierarchical language that supports REMES service composition (HDCL).

For a more thorough description of the REMES model, we refer the reader to [5, 6]. The REMES language benefits from a set of tools[1] for modeling, simulation and transformation into PTA, which could assist the designer during system development.

9.2.2 Priced Timed Automata

In the following, we recall the model of PTA [9,10], an extension of timed automata [8] with prices on both locations and transitions.

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

Let χ be a finite set of clocks and $\mathcal{B}(\chi)$ the set of formulas obtained as conjunctions of atomic constraints of the form $x \bowtie n$, where $x \in \chi$, $n \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$. The elements of $\mathcal{B}(\chi)$ are called *clock constraints* over χ.

Definition 1. *A linearly Priced Timed Automaton (PTA) over clocks* χ *and actions Act is a tuple* (L, l_0, E, I, P)*, where* L *is a finite set of locations,* l_0 *is the initial location,* $E \subseteq L \times \mathcal{B}(\chi) \times Act \times \mathcal{P}(\chi) \times L$ *is the set of edges,* $I : L \to \mathcal{B}(\chi)$ *assigns invariants to locations, and* $P : (L \cup E) \to \mathbb{N}$ *assigns prices (or costs) to both locations and edges. In the case of* $(l, g, a, r, l') \in E$*, we write* $l \overset{g,a,r}{\to} l'$*.* ∎

The semantics of a PTA is defined in terms of a priced transition system over states of the form (l, u), where l is a location, $u \in \mathbf{R}^X$, and the initial state is (l_0, u_0), where u_0 assigned all clocks in χ to 0. Intuitively, there are two kinds of transitions: delay transitions and discrete transitions. In delay transitions,

$$(l, u) \overset{d,p}{\to} (l, u \oplus d)$$

the assignment $u \oplus d$ is the result obtained by incrementing all clocks of the automata with the delay amount d, and $p = P(l) * d$ is the cost of performing the delay. Discrete transitions

$$(l, u) \overset{a,p}{\to} (l', u')$$

correspond to taking an edge $l \overset{g,a,r}{\to} l'$ for which the guard g is satisfied by u. The clock valuation u' of the target state is obtained by modifying u according to updates r. The cost $p = P((l, g, a, r, l'))$ is the price associated with the edge.

A timed trace σ of a PTA is a sequence of alternating delays and action transitions

$$\sigma = (l_0, u_0) \overset{a_1,p_1}{\to} (l_1, u_1) \overset{a_2,p_2}{\to} \ldots \overset{a_n,p_n}{\to} (l_n, u_n)$$

A network of PTA $A_1 \ldots A_n$ over χ and *Act* is defined as the parallel composition of n PTA over χ and *Act*. Semantically, a network again describes a timed transition system obtained from those components, by requiring synchrony on delay transitions and requiring discrete transitions to synchronize on complementary actions (i.e., $a?$ is complementary to $a!$) [10].

9.2.3 Symbolic Optimal Reachability

The text in this subsection is an adaptation for single-cost PTA, of the similar one presented by Larsen and Rasmussen, for dual-priced PTA [7]. Symbolic techniques are required in analysis of infinite state systems. They provide effective way to describe and manipulate set of states simultaneously. To enable cost-optimal analysis such techniques are enriched with cost information annotated to each individual symbolic state [11].

A priced transition systems with a structure $\tau = \langle S, s_0, \Sigma, \rightarrow \rangle$, where S is a set of states, $s_0 \in S$ is the initial state, Σ is a finite set of labels, and $\rightarrow$ is a partial function from $S \times \Sigma \times S$ into the non-negative reals, $\mathbb{R}_{\geq 0}$ defines all possible systems transitions with their respective costs. An execution of τ is a sequence $\gamma = s_0 \overset{a1,p1}{\rightarrow} s_1 \overset{a2,p2}{\rightarrow} \dots \overset{an,pn}{\rightarrow} s_n$. The cost of γ with respect to some goal state $G \subseteq S$, is defined as:

$$\mathsf{COST}_G(\gamma) \quad = \quad \begin{cases} \infty, & \text{if } \forall \;\; i \geq 0 \;\; : \;\; s_i \notin G \\ \sum_{i=1}^{n} p_i, & \text{if } \exists \; n \geq 0 \;\; : \;\; s_i \in G \;\; \wedge \;\; \forall \; 0 \leq i < n \;\; : \;\; s_i \notin G. \end{cases}$$

For a given state s, the minimum cost of reaching s is the infimum of the costs of the finite traces ending in s. Dually, the maximum cost of reaching s is the supremum of the costs of the finite traces ending in s. Similarly, the minimum/maximum cost of reaching a set of states $G \subseteq S$ is:

$$\begin{aligned} &\inf \{\mathsf{COST}_G(\gamma) : \;\gamma \in \Gamma\}, \text{ and} \\ &\sup\{\mathsf{COST}_G(\gamma) : \;\gamma \in \Gamma\} \end{aligned}$$

where Γ is a set of all executions of a priced transition system τ.

To effectively analyze the priced transition systems, priced symbolic states of the form (A, π) are used, where $A \subseteq S$ is a set of states, and $\pi : A \rightarrow \mathbb{R}_{\geq 0}$ assigns non-negative costs to all states of A. The reachability of the priced symbolic state (A, π), assumes that all $s \in A$ are reachable with all cost in $\pi(s)$. To express successors of priced symbolic states, a Post-operator $Post_a(A, \pi) = (post_a(A), \eta(A))$ is expressed as:

$$post_a(A) = \{s^{'} \;\; | \;\; \exists s \in A : s \overset{a,p}{\rightarrow} s'\}$$

$$\eta(A) = \{\pi(s') + p \;|\;\; s' \in A \wedge s' \overset{a,p}{\rightarrow} s\}$$

A symbolic execution of a priced transition system τ is a sequence $\beta = (A_0, \pi_0), \dots, (A_n, \pi_n)$, where for $i < n$, $(A_{i+1}, \pi_{i+1}) = Post_{ai}(A_i, \pi_i)$

for some $a_i \in \Sigma$ and $A_0 = \{s_0\}$ and $\pi_0(s_0) = 0$. The relation between executions and symbolic executions is expressed as follows:

- For each execution γ of τ ending in s, there is a symbolic execution β ending in (A, π) such that $s \in A$ and $\mathsf{COST}(\gamma) \in \pi(s)$.
- Let β be a symbolic execution of τ ending in (A, π); then for each $s \in A$ and $p \in \pi(s)$, there is an execution $\gamma \in s$ such that $\mathsf{COST}(\gamma) = p$.

From the statements above one can notice that symbolic states accurately capture the cost of reaching all states in the state space.

9.3 Algorithms for Service Strongest Postcondition Calculation

In cases when more REMES services that deliver the same or similar functionality are available, it becomes beneficial to check the correctness of those services w.r.t. the requirement defined by a service user. To provide constructs for correctness check of a REMES service, described in Section 9.2 and introduced in [5], we will use Hoare triples and a forward analysis technique, which assumes computations of the strongest postconditions 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 user requirement, reducing it to a simple proof.

Previously [5], we have focused on less complex systems and employed the guarded command language (GCL) [12] to prove service correctness by hand. In this paper, we aim for a more automated mechanism to check service correctness, focusing on defining algorithms that facilitate such computation for REMES services formally described as PTA. Each REMES service can be transformed into PTA, by the set of rules defined in [13]. Equipped with such a transformation, we can perform minimum/maximum resource-usage trace computation on the resulting PTA, while carrying out the strongest postcondition analysis.

The REMES models transformed to PTA are assumed to be deadlock free, limited[2], and no infinite delays are allowed, which are implemented

[2]A global finite-domain data variable can be incremented on each edge, in order to prove model boundness.

by associating a time invariant to each automaton location. The algorithms presented in this paper are inspired by the symbolic reachability algorithms for computing the minimal and the maximal reachability cost, respectively, proposed by Larsen and Rasmussen [7].

In the following, we recall the notion of strongest postcondition, as introduced by Dijkstra and Sholten [14], and the program correctness check based on it. Next, we introduce two algorithms that compute the strongest postcondition of a REMES service formally described as PTA, together with the minimum/maximum cost reachability analysis, respectively.

9.3.1 Strongest Postcondition

Assume $\{p\}\mathcal{S}\{q\}$ is a Hoare triple, where p and q are predicates denoting the partial correctness of service $\mathcal{S}$ with respect to precondition p and postcondition q. According to Dijkstra and Sholten [14], the strongest postcondition transformer, denoted by $(sp.\mathcal{S}.p)$, is a set of final states for which there exists a computation controlled by $\mathcal{S}$, which belongs to class "initially p". Given that p holds, execution of a service $\mathcal{S}$ results in $sp.\mathcal{S}.p$ true, if $\mathcal{S}$ terminates. Proving the Hoare triple, that is, proving the correctness of service S, reduces to showing that $(sp.\mathcal{S}.p \Rightarrow q)$ holds.

To illustrate the strongest postcondition calculation on a simple statement, let us assume that a service performs a simple subtraction operation $(x := x - 5)$ and that the provided precondition is $p = (x > 15)$, while the requirement is $q = (x > 10)$. Then, calculating the strongest postcondition reduces to the following:

$$sp.(x := x - e).p(x) = (\exists x_0 \cdot x = x_0 - 5 \wedge (x_0 > 15))$$

where x_0 is the initial value of x. Verifying the correctness of S, with respect to p, and q above, reduces to showing that:

$$\exists x_0 \cdot x = x_0 - 5 \wedge (x_0 > 15) \Rightarrow (x > 10)$$

9.3.2 Strongest postcondition calculation and minimal cost reachability

In this section, we show the algorithm that computes the strongest postcondition, and the minimal cost of resource consumption for a given REMES service, formally described as a PTA.

Algorithm 1 employs two data-structures, Waiting and Passed to hold the priced symbolic states waiting to be examined, and those that are already explored, respectively. At each iteration, the algorithm selects a priced symbolic state (A, π) from Waiting. If (A, π) is a goal state[3] not contained in a goal state previously stored in SP, it is added to the calculated postcondition SP. Otherwise, if it is not a goal state and not contained in a symbolic state previously stored in Passed, it is added to Passed, and all its successor states are added to Waiting. When Waiting is empty, the strongest postconditions calculated for each path reaching the goal state are returned.

We define Final (A, π) as follows:

$$\text{Final}\,(A, \pi) \quad = \quad \begin{cases} \mathit{true}, & \text{if } (A, \pi) \in \ G \\ \mathit{false}, & \text{otherwise} \end{cases}$$

where G denotes the final state.

The algorithm terminates when Waiting is empty, that is, when no further priced symbolic state is left to be examined. The algorithm results in a set of strongest postconditions SP. Termination of the algorithm is guaranteed, provided that $\not\sqsubseteq_{inf}$ is a well quasi-ordering on symbolic states [11].

Relying on the fact that the algorithm takes into consideration priced symbolic states, and that by checking $(B, \eta) \not\sqsubseteq_{inf} (A, \pi)$ we ensure that there are no states in Passed that dominate the current state, that is, the state (A, π) is "as big and cheap" as (B, η), information about the minimum cost is carried within the calculated strongest postcondition.

```
00  SP := {}
01  PASSED := {}
02  WAITING := $\{(\{A_0\}, \pi_0)\}$
03  **while** WAITING $\neq \{\}$ **do**
04    select $(A, \pi)$ from WAITING
05    **if** (Final$(A, \pi) \ \wedge \ \forall\ (B, \eta) \in$ SP : $(B, \eta) \not\sqsubseteq_{inf} (A, \pi)$)
06    **then** SP := SP $\cup\ (A, \pi)$ **else**
07        **if** $\forall\ (B, \eta) \in$ PASSED : $(B, \eta) \not\sqsubseteq_{inf} (A, \pi)$ **then**
08           PASSED := PASSED $\cup \{(A, \pi)\}$
09           WAITING := WAITING $\cup\ \bigcup_{a \in \Sigma} Post_a(A, \pi)$
```

[3]Note that, in a PTA describing a Remes service, the goal state is determined by a unique location and hence, if Final (A, π) holds, then the whole of (A, π) is a goal state.

```
10            end if
11        end if
12    end while
13    return SP
```

Algorithm 1. Abstract algorithm for computing the service strongest postcondition and the minimal reachability cost of reaching the goal state.

As stated, the algorithm provides a set of strongest postconditions calculated for distinctive paths that reach the goal state (location) in PTA. Finally, to provide the strongest postcondition, we also simplify the set SP. The strongest postcondition can be simplified as follows:

$$\forall\ (A,\pi)_i \in \mathrm{SP}\ :\ \bigcup_{j \neq\ i} (A,\pi)_j \not\sqsubseteq_{inf} (A,\pi)_i$$

The simplification assumes that each symbolic priced state that is not included into the reunion of all other symbolic priced states is subtracted. For more details regarding simplification, we refer the reader to [15, 16].

9.3.3 Strongest postcondition calculation and maximal cost reachability

Algorithm 1 can be modified to provide the strongest postcondition calculation together with maximal cost. We here briefly sketch the required modifications. Like in the above, we assume that all paths eventually reach the goal state. The modification concerns the lines 05 to 07 of Algorithm 1, which should be modified as follows:

```
05    if (Final $(A,\pi)$ $\wedge$ $\forall$ $(B,\eta) \in$ SP : $(B,\eta) \not\sqsupseteq_{sup} (A,\pi)$)
06    then SP := SP $\cup$ $(A,\pi)$ else
07        if $\forall$ $(B,\eta) \in$ PASSED : $(B,\eta) \not\sqsupseteq_{sup} (A,\pi)$) then
```

Algorithm 2. Extract of abstract algorithm for computing the service strongest postcondition and the maximal reachability cost of reaching the goal state.

Hence, the only difference from the previous algorithm is in the pruning of symbolic priced states before adding them to PASSED or SP. Any symbolic state (A,π) can be pruned if there exists already a symbolic state (B,η) such that $A \subseteq B$ and $\pi(s) \preceq \eta(s)$ for all states $s \in A$.

Similarly, the strongest postcondition can be simplified as follows:

$$\forall\ (A,\pi)_i \in \mathrm{SP}\ \ :\ \bigcup_{j \neq\ i} (A,\pi)_j \not\sqsupseteq_{sup}\ (A,\pi)_i$$

9.4 An Illustrative Example

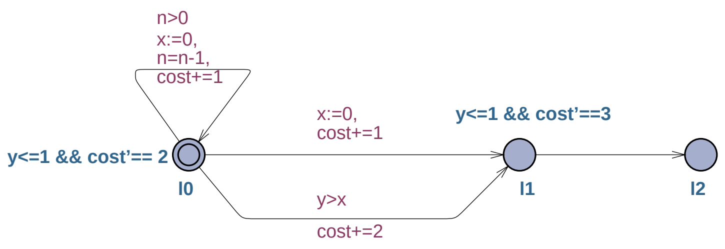

Figure 9.1: PTA model of example

To illustrate our approach, we consider the simple PTA shown in Fig. 9.1. It consists of three locations l_0, l_1, and l_2, and four transitions. The timing behavior is expressed using the two clocks x and y, initially set to zero. In the automaton, it is possible to delay either in location l_0 or l_1. Location l_2 is assumed to be the final location (in which the automaton immediately terminates). From l_0 it is possible to take a self-loop, for maximum two times (if integer n is initially set to two) and then take one of the available transitions, or directly take one of the available transitions that lead to location l_1, and finally end up in the final location l_2. Staying in either l_0 or l_1, or taking any of the available transitions brings the additional cost, 2 or 3 per time unit, respectively, annotated via the cost variable. We are interested in calculating both, the minimum and the maximum cost for reaching the final location (l_2) and the respective strongest postcondition.

Let us now assume that our example PTA describes a REMES service, annotated with precondition p, which we assume satisfied, and postcondition q, which represents the service requirement, as follows:

$$p = (n = 2 \wedge x = 0 \wedge y = 0)$$
$$q = (n \geq 0 \wedge y \geq x \wedge 0 \leq r \leq 10)$$

In the above, x and y are clock variables, n is an integer variable that bounds the number of loop iterations in location l_0, and r is the variable modeling the resource usage of the original service. In the corresponding PTA representation, r translates into the automaton's cost variable.

According to our methodology, to verify the correctness of the service with respect to p and q, we need to first compute the strongest postcondition of the corresponding PTA, under the assumption of worst-case resource usage, meaning of maximum cost in PTA terms. This is needed in order for the requirement to be met for any possible behavior, including the worst-case one.

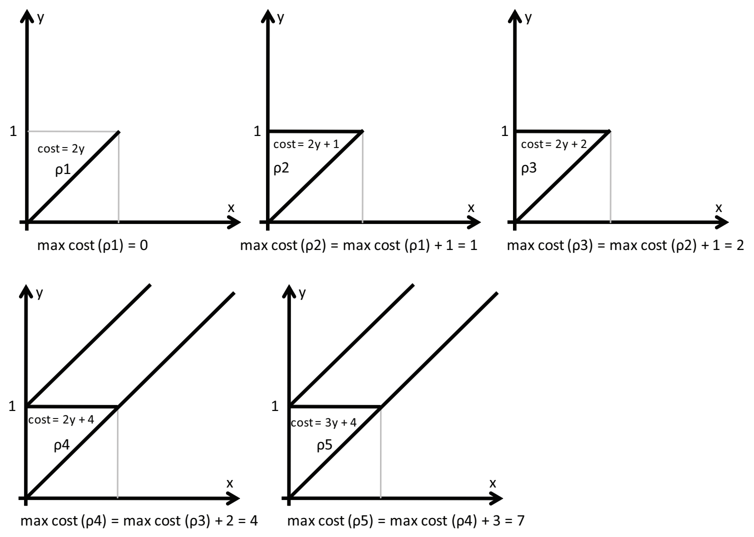
y
x
1
cost = 2y
ρ1
max cost (ρ1) = 0
cost = 2y + 1
ρ2
max cost (ρ2) = max cost (ρ1) + 1 = 1
cost = 2y + 2
ρ3
max cost (ρ3) = max cost (ρ2) + 1 = 2
cost = 2y + 4
ρ4
max cost (ρ4) = max cost (ρ3) + 2 = 4
cost = 3y + 4
ρ5
max cost (ρ5) = max cost (ρ4) + 3 = 7

Figure 9.2: Symbolic states for maximum reachability cost

Fig. 9.2 depicts a trace of the reachability analysis, assuming the maximum cost of reaching the goal location. In this case, we illustrate a trace that performs two self-loops in l_0, and then transits to l_1 via the lower of the two possible edges. The costs are $2y$, $2y + 1$, and $2y + 2$ in l_0 and then $3y + 4$ in l_1 (and in l_2). The strongest postcondition w.r.t. maximum cost of the trace becomes cost $= 3y + 4 \wedge y \leq 1 \wedge x <$

$y \wedge n = 0$. The total SP of the whole PTA w.r.t. maximum cost becomes

$$\begin{array}{l}(\mathsf{cost} = 3y + 1 \wedge y \leq 1 \wedge x \leq y \wedge n = 2) \vee \\ (\mathsf{cost} = 3y + 2 \wedge y \leq 1 \wedge x \leq y \wedge n = 1) \vee \\ (\mathsf{cost} = 3y + 3 \wedge y \leq 1 \wedge x \leq y \wedge n = 0) \vee \\ (\mathsf{cost} = 3y + 3 \wedge y \leq 1 \wedge x < y \wedge n = 1) \vee \\ (\mathsf{cost} = 3y + 4 \wedge y \leq 1 \wedge x < y \wedge n = 0)\end{array}$$

By applying simple rules of logic, it is easy to verify that the above maximum cost strongest postcondition implies the following predicate:

$$w = (1 \leq \mathsf{cost} \leq 7 \wedge x \leq y \wedge n \geq 0)$$

Next, after replacing r by cost in q, it follows straightforwardly that $w \Rightarrow q$, which entails, by transitivity of the implication, that the strongest postcondition for maximum cost implies the requirement q. This actually proves the correctness of our original service, including its feasibility w.r.t. worst-case resource usage.

However, let us assume now that we would like to check whether our service could be optimized w.r.t. its resource usage, such that our system could accommodate more, within the same resource budget. For this, we formalize a new requirement, as follows:

$$q_{r_{min}} = (n \geq 0 \wedge y \geq x \wedge 0 \leq r \leq 6)$$

Proving correctness of the PTA w.r.t. this new requirement requires a new strongest postcondition computation, for minimum cost, according to Algorithm 1.

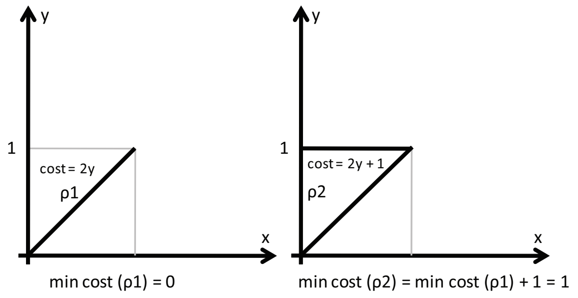

Figure 9.3: Symbolic states for minimum reachability cost

In Fig. 9.3, we illustrate one trace of the minimum cost reachability analysis that reaches the goal location l_2. Note that in the minimum cost case, it is optimal to reach l_2 in zero time units, via location l_1. The accumulated cost is then 1. In case of the total accumulated delay 1, it is optimal to delay in l_0 with cost 2, hence the cost of reaching l_2 is $2y+1$ and the strongest postcondition *for this trace* is $\mathsf{cost} = 2y+1 \wedge y \leq 1 \wedge x \leq y \wedge n = 2$.

There are four more traces reaching l_2. The total SP becomes

$$\begin{array}{l}(\mathsf{cost} = 2y + 1 \wedge y \leq 1 \wedge x \leq y \wedge n = 2) \vee \\ (\mathsf{cost} = 2y + 2 \wedge y \leq 1 \wedge x \leq y \wedge n = 1) \vee \\ (\mathsf{cost} = 2y + 3 \wedge y \leq 1 \wedge x \leq y \wedge n = 0) \vee \\ (\mathsf{cost} = 2y + 3 \wedge y \leq 1 \wedge x < y \wedge n = 1) \vee \\ (\mathsf{cost} = 2y + 4 \wedge y \leq 1 \wedge x < y \wedge n = 0)\end{array}$$

Again, we can easily prove that the above strongest postcondition implies the following:

$$v = (1 \leq \mathsf{cost} \leq 6 \wedge x \leq y \wedge n \geq 0),$$

which in turn implies $q_{r_{min}}$, with r replaced by cost. It then follows by the transitivity property of the implication that the minimum-cost strongest postcondition implies, in turn, $q_{r_{min}}$, which completes our correctness proof.

9.5 Discussion and Related Work

Beek et al. [17] give an exhaustive survey on several popular approaches [1–4] that provide means for service modeling, service composition, and service correctness check. While all described approaches provide rich environment for service modeling and composition, previously neither of them have included direct support for service correctness check. To overcome this limitation, recently, in some of these approaches [18–20] formal methods have been employed with intention to provide guarantees for web-service compositions.

Díaz et al. describe how BPEL and WS-CDL services can be automatically translated to timed automata and verified by UPPAAL model checker [18]. However,described approach is limited to checking 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 [19]. Analysis includes includes reachability and liveness properties and checking if the given service or service compositions are deadlock free.

Compared to these approaches, REMES services can be both mechanically reasoned about [5], but also, automatically translated to PTA [9] where one can apply algorithmic computation of the strongest postcondition of PTA, as presented in this paper. Moreover, REMES services formally described as PTA can be analyzed with UPPAAL , or CORA tools [4], for functional but also extra-functional behaviors.

9.6 Conclusions

In this paper, we have presented an approach that facilitates an automated correctness check for services formally described as PTA, by providing forward analysis algorithms that compute the most precise postcondition (strongest postcondition) that is guaranteed to hold upon termination of the service execution, which corresponds to reaching a final PTA location. The approach serves as the algorithmic alternative verification method for services modeled as REMES modes, to the deductive method that uses Hoare triples and the strongest postcondition semantics to prove service correctness [5].

In our previous work, we show that proving the correctness of a REMES service reduces to showing that the calculated strongest postcondition of that particular service is at least as strong as the user-defined requirement, if not stronger. The algorithms that we propose combine the strongest postcondition calculation for PTA with minimum- and maximum-cost reachability algorithms, already existing in the literature [7]. In our case, the cost variable models the service's accumulated resource-usage. Consequently, the computed strongest postcondition of a service modeled as a PTA could contain both functional, but also timing and resource-usage information, observable at the end of the service execution.

The approach is illustrated on a small example, on which we also show resource usage/cost calculation using symbolic states, for the example service.

As future work, we plan to investigate the possibility of implementing the introduced strongest postcondition algorithms in the CORA tool. We also intend to extend the REMES tool-chain with a postcondition calculator that would run CORA as a back-end.

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

Bibliography

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

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

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

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

[10] 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.

[11] Kim Guldstrand Larsen, Gerd Behrmann, Ed Brinksma, Ansgar Fehnker, Thomas Hune, Paul Pettersson, and Judi Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proceedings of the 13th International Conference on Computer Aided Verification*, CAV '01, pages 493–505, London, UK, 2001. Springer-Verlag.

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

[13] Marin Orlić. *Resource usage prediction in component-based software systems*. Phd thesis, Faculty of electrical engineering and computing, University of Zagreb, November 2010.

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

[15] Pao-Ann Hsiung and Shang-Wei Lin. Model checking timed systems with priorities. In *Proceedings of the 11th IEEE International*

Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA '05, pages 539–544, Washington, DC, USA, 2005. IEEE Computer Society.

[16] Alexandre David, John Håkansson, Kim Larsen, and Paul Pettersson. Model checking timed automata with priorities using dbm subtraction. In Eugene Asarin and Patricia Bouyer, editors, *Formal Modeling and Analysis of Timed Systems*, volume 4202 of *Lecture Notes in Computer Science*, pages 128–142. Springer Berlin / Heidelberg.

[17] 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.

[18] 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.

[19] 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.

[20] 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.