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

Aida Čaušević, Aneta Vulgarakis
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Västerås, Sweden
{aida.delic, aneta.vulgarakis}@mdh.se

Abstract

There is no clear distinction between service-oriented systems (SOS) and component-based systems (CBS). However, there are several characteristics that could let one consider SOS as a step further from CBS. In this paper, we discuss the general features of CBS and SOS, while accounting for behavioral modeling in the language called REMES. *First, we present* REMES *in the context of CBS modeling, and then we show how it can become suitable for SOS. We also discuss the relation between our model and the current state of the art.*

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) [10], and service-oriented software engineering (SOSE) [6]. Since SOSE evolved from CBSE [26], 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) [12, 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 [26, 20], 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 [4, 15], or behavior is described at higher levels of abstraction than programming languages, but they lack a formal description [18, 22]. In component-based approaches authors usually give detailed description of behavior in terms of interfaces, connectors, and protocols [7, 21, 16], 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 [12, 5], yet detailed service behavioral modeling with support for formal analysis is still needed. Seceleanu et al. introduce a behavioral modeling language called REMES [24] 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., ar-

chitecture description languages (ADLs) [17]) and formal analysis models (e.g., timed automata [2]). 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 2 draws a parallel between basic characteristics of CBSE and SOSE. Section 3 discusses the behavior modeling of both CBS and SOS, and shows how REMES can be extended towards modeling service behavior. In Section 4, we present the related work, and in Section 5 we conclude the paper.

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 [10]:

- 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 [13]. 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 [9]. The development process of CBS is separated from the development process of its components, but is influenced by it. As shown in Figure 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 [9].

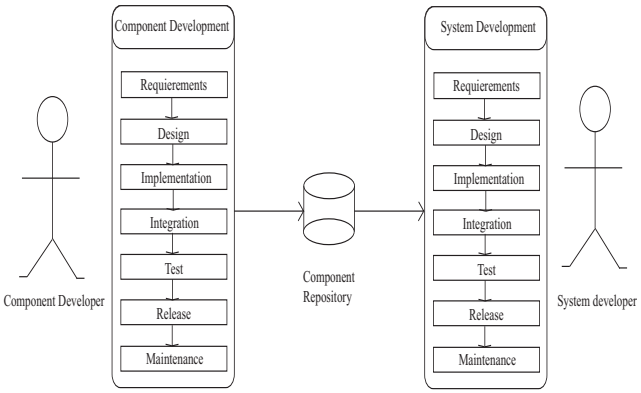

Figure 1. CBSE development process

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 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 runtime. 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 run-

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

time 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 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.

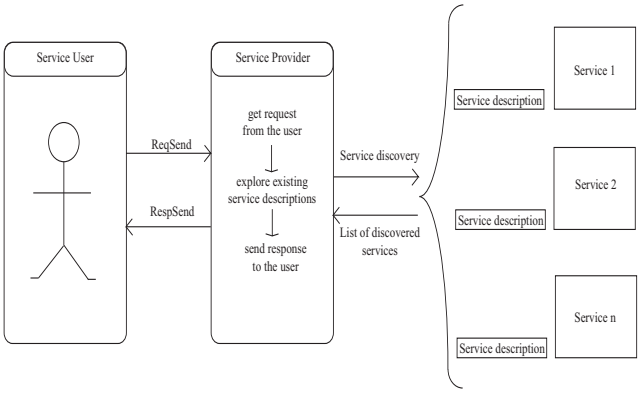

Figure 2. SOSE overview

In order to get the best out of both CBSE and SOSE, these approaches need to be combined, as shown by Collet et al. [8], through unified modeling languages.

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 [11]. 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.

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) [25]. 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) [24] 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) [2] or Priced Timed Automata (PTA) [3] 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 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.

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 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 *control 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 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 [24].

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, [4, 15, 22, 7, 21, 16] 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

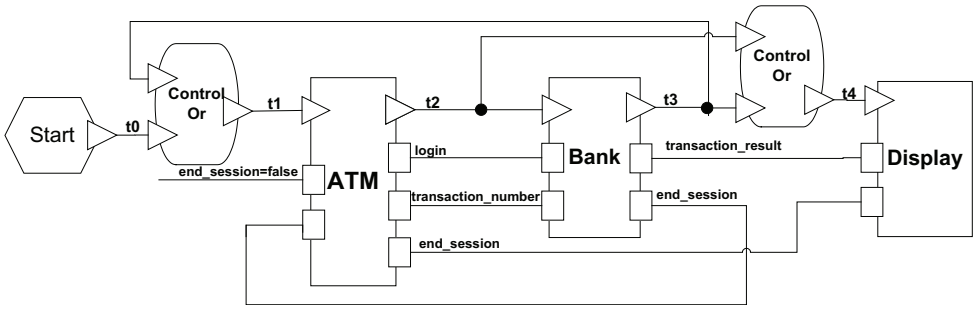

Figure 3. Component based ATM system as a ProCom-based description

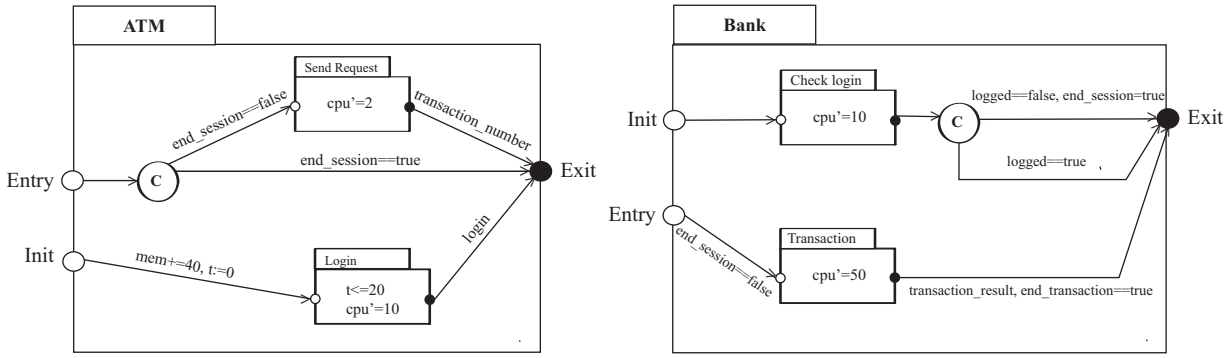

Figure 4. REMES **modes for ATM and Bank**

(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 [14]).
- *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 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.

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 [19]. 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 [15] 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 [4] 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 [1] and BPMN [18]. 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 [18] and WSMO [22] 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 [23] 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 [23] 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.

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 UPPAAL 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.

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).

References

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

[2] http://uppaal.com/

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

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

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

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*. IBM, 2003.

[5] S. Becker, H. Koziolek, and R. 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] M. Broy, N. Diernhofer, J. Grünbauer, M. Meisinger, M. Rappl, S. Rittmann, B. Schätz, M. Schoenmakers, and B. Spanfelner. Service-Oriented Development - Whitepaper. Whitepaper, Technische Universität München, 2006.

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

[8] P. Collet, T. Coupaye, H. Chang, L. Seinturier, and G. Dufrene. Components and services: A marriage of reason. Technical Report ISRN I3S/RR-2007-17-FR, CNRS, May 2007. Project RAINBOW.

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

[10] I. Crnkovic and M. Larsson. *Building Reliable Component-Based Software Systems*. Artech House publisher, 2002.

[11] D. Garlan and M. 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.

[12] V. Grassi, R. Mirandola, and A. 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.

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

[14] T. Henzinger, S. Qadeer, and S. 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.

[15] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. World Wide Web Consortium, Candidate Recommendation CR-ws-cdl-10-20051109, November 2005.

[16] G. T. Leavens and K. K. Dhara. Concepts of behavioral subtyping and a sketch of their extension to component-based systems. pages 113–135, 2000.

[17] 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.

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

[19] R. C. v. 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.

[20] H. Pei-Breivold and M. 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.

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

[22] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.

[23] M. Rychl. 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.

[24] C. Seceleanu, A. Vulgarakis, and P. 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.

[25] S. Sentilles, A. Vulgarakis, T. Bureš, J. Carlson, and I. 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.

[26] 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.