Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 295 (2013) 95–100

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Design Tool for Service-oriented Systems

Eduard Paul Enoiu Raluca Marinescu Aida Čaušević
Cristina Seceleanu [1]

Mälardalen Real-Time Research Centre
Mälardalen University
Västerås, Sweden

Abstract

In this paper we present a modeling and analysis tool for service-oriented systems. The tool enables graphical modeling of service-based systems, within the resource-aware timed behavioral language REMES, as well as a textual system description. We have developed a graphical environment where services can be composed as desired by the user, together with a textual service composition interface in which compositions can also be checked for correctness. We also provide automated traceability between the two design interfaces, which results in a tool that enhances the potential of system design by intuitive service manipulation. The paper presents the design principles, infrastructure, and the user interface of our tool.

Keywords: Service-oriented systems, Behavioral language, textual system description, traceability

1 Introduction

The recently introduced paradigm of Service-oriented Systems (SOS) provides the basis for dealing with software integration and composition, by exploiting loosely coupled and autonomous abstract modeling entities called *services* [4]. The nature of services calls for methods and automated support to design the system, as well as to ensure the quality of service (QoS) of the result. To address such needs, an extension of REMES, an already existing resource-aware timed behavioral modeling language, has been proposed [5]. This extension has enriched REMES with service-oriented features, i.e., service interface description such as type, capacity, time-to-serve, status, pre-, and postcondition, a Hierarchical Language for Dynamic Service Composition (HDCL), as well as with means to check service compositions. All these features make REMES suitable for behavioral modeling and analysis of SOS. Due to the pre-, postcondition annotations, the correctness check for a REMES service,

[1] {eduard.paul.enoiu, raluca.marinescu, aida.delic, cristina.seceleanu}@mdh.se

http://dx.doi.org/10.1016/j.entcs.2013.04.008

or a composition of such services reduces to verifying simple boolean implication between the respective services.

In this paper, we present a tool for designing SOS in Remes. The tool consists of a graphical environment for behavioral modeling of services, a service composition view integrated as a textual interface, and a correctness condition generator for service compositions. The distinguishing features of our tool reside in the possibility of tracking changes between the graphical and textual views, automatically, as well as in the automated generation of corresponding verification conditions for the composed services [5]. Last but not least, connecting to a prover and/or model-checker, from the tool, to discharge the verification conditions, is one-step away.

There has been a large body of work on tools on component-based systems (CBS) and SOS modeling and analysis. Many existing approaches for CBS are intended to support graphical interfaces, compositional verification, and different kind of analysis [9,3,10]. Although these tools offer reusability and user-friendliness, they are not integrated in a graphical and textual interface with traceability features. Also, there are several tools that support SOS modeling and analysis [7,8,11,1]. One of them is KarmaSIM tool [2] that uses DAML-S ontology to describe the capabilities of web services. It supports interactive simulation and various verification and performance analysis techniques, but in comparison to our work shows limited capabilities to automatically support these processes. There are some tool solutions gathered around BPEL service description. One of them is WSAT tool [8], which provides verification of LTL properties on BPEL processes using SPIN model checker. The tool covers only the untimed aspects of BPEL. Another tool [11] translates BPEL processes directly into state/transitions graphs, and analyzes behavioral and discrete-time aspects of BPEL description. In comparison to our approach these tools lack resource-aware analysis capabilities. SRML [1] is a service modeling framework that relies on UML state machines to model service behavior. It supports the formal analysis of functional and timing properties, whereas we can also cater for resource-oriented modeling and analysis.

The rest of the paper is organized as follows: Section 2 presents the overall approach together with the tool workflow (Subsection 2.1), the user interface (Subsection 2.2), the model traceability and verification condition generator (Subsection 2.3), whereas Section 3 concludes the paper and gives some hints of the future work.

2 The SOS Design Tool: Workflow and User Interface

Our tool [3] supports behavioral modeling of services by allowing their graphical specification, including their attributes and internal behavior. It provides an environment [4] to specify, model, and compose Remes services, graphically, while

[2] More information available at http://www.ai.sri.com/daml/services/.

[3] The current version of the tool is available online via the webpage http://www.idt.mdh.se/personal/eep/reseide/

[4] The client front-end is based on *NetBeans Visual Library API* to display the graphical models and *Java Swing* as the user interface toolkit.

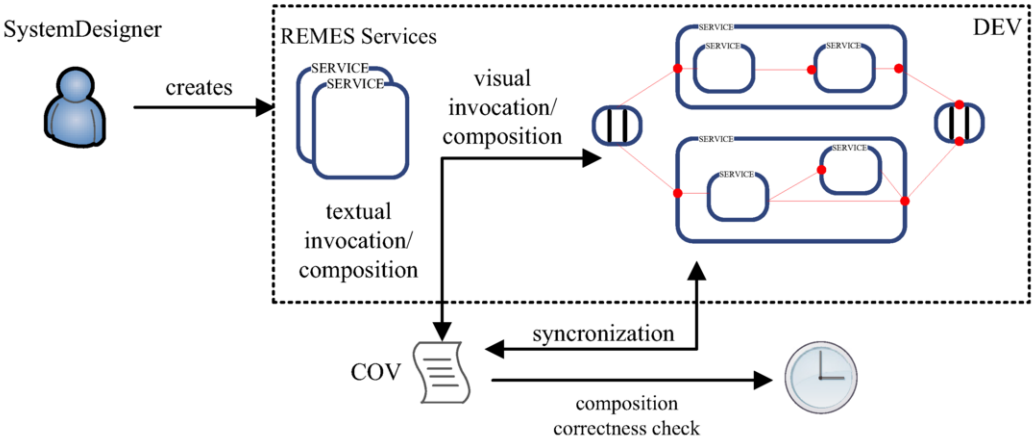

Fig. 1. The tool workflow

generating the equivalent textual system representation, and corresponding correctness verification conditions of the compositions.

2.1 Workflow

The tool is divided into two functional units: Diagram Editor View (DEV) and Console View (COV). DEV is the top level unit in charge for opening a new diagram editor, creation of services, and for displaying a service composition. It uses the NetBeans Visual Library API to render the created diagram and contains a large visual modeling interface. COV supports the textual description of the system, including service declarations, lists of services, and their composition; in this console-like interface, the correctness verification conditions for services can be generated, once a composition is created.

Fig. 1 illustrates the design flow implemented in our tool. The designer uses: (i) DEV for building and composing services in a graphical environment and (ii) COV for invoking services using HDCL. One can synchronize DEV and COV in case the model has been modified in one of the views, and to check whether the given requirement is satisfied, by forward analysis (strongest postcondition calculus [5]).

Our tool enables system composition by using services as basic units. A *service repository* is available to service users, and consists of services modeled using the REMES extended interface and the behavioral language. These registered services can be invoked and composed in different ways, based on the preferences of the service user.

2.2 User Interface

Fig. 2 depicts a screenshot of the tool displaying a behavioral model of a service. The Pallete (located on the righ-hand side of the graphical environment) provides quick access to all the graphical elements. The user can create REMES services, *AND/OR services* (which model synchronized behavior), and compose created / retrieved

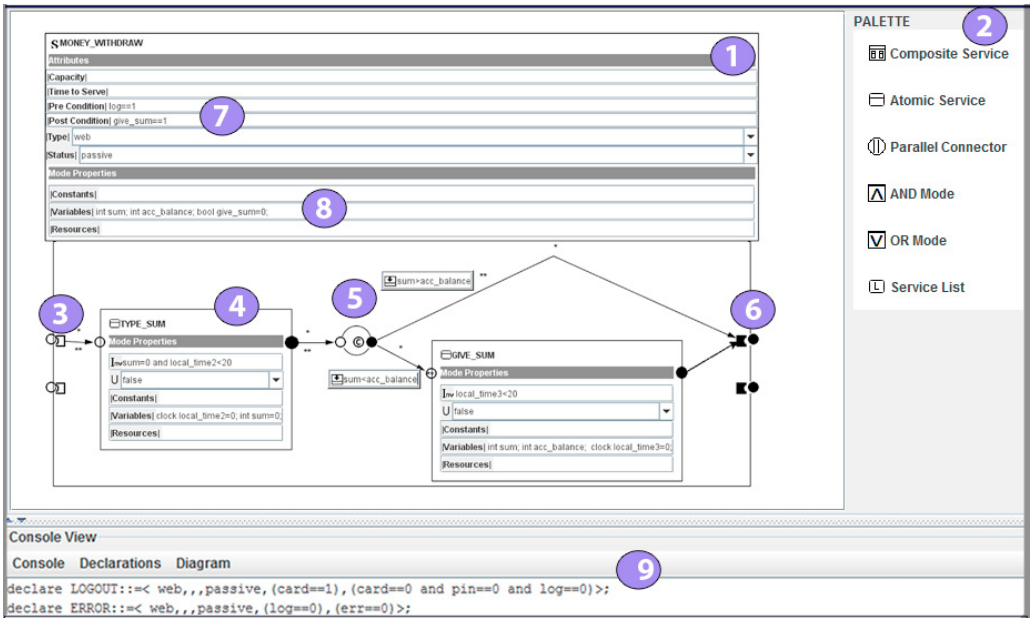

Fig. 2. A screenshot of the tool. A composite service (1) can be created by using the Palette (2) and can have a number of associated service attributes (7) , constants, variables, and resources (8), displayed in separate compartments. The services are entered via their init-,or entry points (3). They can be described using the REMES language (4), connected by edges and conditional connectors (5), and exited through their exit points (6). After each diagram composition, one can check whether the given requirement is satisfied (9).

services in new services, by using serial, parallel operators, and list constructors. Services are connected via *edges* on control points. REMES models and *conditional connectors* can be nested inside a composite service. A REMES model is described in terms of the REMES hierarchical language [5].

Also, the user has the possibility to use HDCL to compose services in COV. Services can be viewed as units that can be composed to create new services in order to fulfill requirements that might change, and consequently involve adaptation of existing services. Moreover, COV displays the verification condition that should be proven in order infer correctness of service composition.

2.3 Model Traceability and Verification Condition Generator

The user can compose services, either graphically or textually, and the tool offers him/her the possibility to visualize the transformations from one interface to another. It is important to note that this feature enables consistency checks, information exchange, and *traceability links*. The idea is to perform a round-trip between views. The service repository is passed to both views, which in turn allows the user to compose services with that selection. Through the process of defining traceability links we have defined each link with a root class representing a created service. Moreover, a link contains a number of *traceability link attributes*, all of which are associated with service attributes of different types. Apart from the references that represent link ends, each traceability link typically stores some additional primitive

information too. This information either applies to a service composition or to some particular list of services. In addition, establishing a traceability between the tool views extends beyond construction the service syntax and also involves specifying the service composition and the derived correctness conditions.

The tool supports the specification of the composition correctness conditions using the *strongest postcondition predicate transformer* (sp) [6], and therefore allows the user to refine services by weakening service preconditions, or strengthening the service postconditions in order to satisfy the system requirement. Assuming p and q are predicates that describe the initial condition and the final guarantee of a service S, the notation $\{p\}\ S\ \{q\}$ means that if p holds initially, then S is guaranteed to establish q, provided that it terminates. The strongest post-condition $sp.S.p$ states that if p holds then the execution of S results in $sp.S.p$ being true. This means that for S to be correct with respect to p and q, then $sp.S.p \Rightarrow q$ should hold. An example of pre-, post-condition specification of a service $S \stackrel{\wedge}{=} x := x+2$ is as follows:

$$\{x = 0\}\ x := x + 2\ \{x = 2\} \tag{1}$$

For this example in (1) the strongest postcondition can be computed as follows:

$$sp.(x := x + 2).(x = 0) \equiv (\exists x_0 \cdot x = x_0 + 2 \wedge x_0 = 0) \Rightarrow x = 2 \tag{2}$$

where x_0 is the fresh variable storing the initial value of x. With this information at hand, one is able to reason about service compositions in an automatic way, by using a model checker or theorem prover.

3 Conclusions

In this paper, we have presented a tool for specification, modeling, and analysis of SOS, which provides interfaces to graphically and textually design the system, but it can be also used to detect errors. The tool adopts the language Remes as the service model, and uses a hierarchical language for composing existing services into new ones, depending on user needs. Through a palette and a repository, the tool makes it easy to specify both functional, as well as extra-functional behavior of services i.e., timing, resource usage etc., whereas the console lets one to rapidly modify an already existing composition, by adding textual information, which is then automatically reflected into the graphical representation. The correctness verification of services is done via strongest postcondition calculus, and reduces to discharging boolean formulae automatically generated by the tool, from the graphical view. However, the tool still awaits integration with model-checkers like Uppaal [5], since Remes semantics is given in terms of Priced Timed Automata (PTA) [2], as well as with a prover for automatically verifying the generated correctness conditions.

[5] The Uppaal tool is available at http://uppaal.com/

Acknowledgment

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement number 269335 and from VINNOVA, the Swedish Governmental Agency for Innovation Systems.

References

[1] João Abreu, Franco Mazzanti, José Luiz Fiadeiro, and Stefania Gnesi. A model-checking approach for service component architectures. In *Proceedings of the International Conference on Formal Techniques for Distributed Systems*, pages 219–224. Springer-Verlag, 2009.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, April 1994.

[3] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*, pages 3 –12, sept. 2006.

[4] Manfred Broy. Service-oriented systems engineering: Modeling services and layered architectures. In *Formal Techniques for Networked and Distributed Systems*, Lecture Notes in Computer Science. Springer, 2003.

[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*. Springer, October 2010.

[6] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18:453–457, August 1975.

[7] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *Network and Parallel Computing*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005.

[8] Xiang Fu, Tevfik Bultan, and Jianwen Su. Wsat: A tool for formal analysis of web services. In *the Proc. of 16th Int. Conf. on Computer Aided Verification, year = 2004, publisher = Springer.*

[9] Dimitra Giannakopoulou, Jeff Kramer, and Shing Chi Cheung. Behaviour analysis of distributed systems using the tracta approach. *Automated Software Engineering*, 6:7–35, 1999. 10.1023/A:1008645800955.

[10] Juan Lara and Hans Vangheluwe. Atom3: A tool for multi-formalism and meta-modelling. In *Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science. 2002.

[11] Radu Mateescu and Sylvain Rampacek. Formal modeling and discrete-time analysis of bpel web services. Lecture Notes in Business Information Processing. Springer, 2008.