Behavioral Modeling and Refinement of Services

Aida Čaušević Cristina Seceleanu Paul Pettersson

Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University,
Västerås, Sweden
{aida.delic, cristina.seceleanu, paul.pettersson}@mdh.se

Service-oriented systems (SOS) have recently emerged as context-independent component-based systems (CBS). The SOS architecture assumes the *service* as the smallest functional unit, with capabilities of being published, invoked, composed and destroyed at run-time. Services are more loosely coupled if compared to components in CBS, and enjoy higher level of independence from implementation specific attributes. They are exposed in order to be discovered, and since they are invoked and composed at run-time by the service user, the quality-od-service (QoS) issues are emphasized.

As mentioned above, services can be perceived as independent and distributed functional units, which can be composed to form new services. The service as a composition of smaller services is called an *orchestration*. The orchestration can be defined as a composition of requested and provided services, that is, each service is defined from a user and/or provider perspective. Modeling service orchestration might require successive *refinements* of the service models. A refinement between services is a relation that establishes the fact that the new, more concrete model defines additional properties of the service, while preserving the properties of the initial model, which has been refined.

In order to understand the ways in which services can evolve, a *service behavioral description* is required. A service behavioral description includes not only the interface description, but also representation of functionality, enabled actions, resource annotations, and possible interactions within the service. Such behavior is in general hidden from the service user, but 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. Most of the SOS frameworks treat service behavioral description either in close connection to the underlying programming language [3, 4], or the behavioral description is given at a higher level of abstraction than programming languages, but lack a corresponding formal description [5, 6].

Many approaches have tried to model services in new and dedicated SOS frameworks. In this work, we assume that the SOS's architecture is modeled in the real-time component-based framework called ProCom (A Component Model for Distributed Embedded Systems) [9]. The ProCom component model defines the CBS as a collection of hierarchical, structured and interconnected ProCom components, be they *active* or *passive*, with well defined *input* and *output interfaces*. The *data* and *control flow* are modeled separately in ProCom, respectively: data can be read and written through the *data ports*, whereas *trigger ports* ensure the component's activation.

Modeling the behavior of ProCom components is realized in Remes (REsource Model for Embedded Systems) [8]. Remes provides a meaningful basis for modeling and analyzing the resource-constrained behavior of embedded systems. Moreover, the modeling of both continuous (i.e., energy) and discrete resources (i.e., memory access to external devices) is accounted for, a resource being a built-in type in the model. Remes is a state-machine-based behavioral language that supports hierarchical modeling, continuous time, and has

notions of explicit entry and exit points that make it suitable for component-based system modeling. The formal semantics of REMES models is given in terms of timed automata (TA) [1] or priced timed automata (PTA) [2], which sets the ground for formally analyzing such models. Transforming REMES models into a TA or PTA network depends on the type of the analysis needed (i.e., timing analysis, resource consumption analysis, etc.). Here, we show how services can be (internally) described by employing an extended version of the behavioral language REMES, and to present an accompanying formal definition of a resource-wise simulation relation between services, as a sufficient condition for proving refinement of services.

Behavioral Modeling of Services and their Semantics. In REMES, a service is represented by an *atomic* or a *composite* mode. The atomic mode does not contain any submodes, whereas a composite one may contain one or more atomic submodes. 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. Such attributes are as follows:

- service type - defines whether the given service is a web service (i.e., weather report, currency exchange, etc.) or a database service (i.e., ATM services);
- service capacity - describes the service's ability to handle a given number of messages per time unit. It can represent the maximum frequency of a given service;
- time-to-serve - defines the worst-case time needed for a service to respond and serve a given request.

Given the REMES service *Mode*, when publishing *Mode*, the above mentioned information will be given as follows:

$$\begin{array}{lcl} Mode.Type & = & < web\ service, database > \\ Mode.Capacity & = & < number\ of\ messages\ per\ time\ unit > \\ Mode.TimeToServe & = & < number\ of\ time\ units > \end{array}$$

Such properties are also used for discovering the service *Mode*: the attributes are specified by an interested party and, based on the specification, the service is retrieved or not.

Let X be a finite set of clocks and $\mathcal{B}(X)$ the set of formulas obtained as conjunctions of atomic constraints of the form $x \bowtie n$, where $x \in X$, $n \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$. The elements of $\mathcal{B}(X)$ are called *clock constraints* over X. We denote by *Act* the set of actions that assign values to clocks and other variables. Then, the formal definition of a REMES service is a PTA in which we deliberately replace the function that assigns costs to locations and edges with a function that assigns resource usage values to modes and edges in REMES:

Definition 1 *A* REMES *service over clocks X is a tuple is a tuple (M, m_0, E, I, Res), where M is a finite set of modes m, m_0 is the initial mode, $E \subseteq M \times \mathcal{B}(X) \times Act \times 2^X \times M$ is the set of edges, $I : M \rightarrow \mathcal{B}(X)$ assigns invariants to modes, and $Res : (M \cup E) \rightarrow \mathbb{N}$ assigns resource-usage values to both modes and edges.* ■

The semantics of a REMES service is given as a *labeled timed transition system with resources* (LTTS), with *discrete transitions* that result in changing the current state, and *delay*

transitions that do not change the state but result in time progress. A run of a REMES service is a path in the underlying transition system. Given a run $\xi = s_0 \stackrel{r^0}{\rightarrow} s_1 \stackrel{r^1}{\rightarrow} \ldots \stackrel{r^{n-1}}{\rightarrow} s_n$, where $s_0, \ldots, s_n$ are states, and $r^0, \ldots, r^{n-1}$ are the corresponding resource usage values per transition, respectively, its i^{th} accumulated resource-usage value is $Res_i(\xi) = \sum_{j=0}^{n-1} r_i^j$.

Our REMES language needs to be extended with constructs for creating a new service by connecting two services (*Mode1* and *Mode2*) at run-time, that is, by binding the exit point of *Mode1* to the entry point of *Mode2*, and also for destroying a service:

`build` $NewMode.Type \equiv$ `connect` $Mode1.Exit$ `to` $Mode2.Entry$	:	create a service
`destroy` $NewMode$	:	delete a service

The formal semantics attached to such constructs is subject to future work.

Resource-wise Refinement of REMES Services. 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.

Let us assume that the total accumulated value of resource-usage over various resources $R_1, \ldots, R_n$ of some system P is given as the following weighted sum: $r^P =_{def} \sum_{i=1}^{n} w_i * r_i^P$, where w_i are numbers that encode the relative importance of each resource, respectively, and r_i^P are the resource-usage values corresponding to R_i, respectively. Then, we introduce the definition of the *resource-wise simulation relation* between two LTTS, as follows:

Definition 2 *Given two LTTS, P and Q, P comparable to Q (same set of labels and global variables), a relation $\mathcal{R} \subseteq S_P \times S_Q$ is a resource-wise simulation relation iff: $\forall a \in \Sigma_P$ (Σ_P is the set of synchronization channels, labels of P), $\forall r_i^P$, $\forall (p,q) \in \mathcal{R}$, and $\forall p' \in S_P$, we have:*

$$
\begin{array}{l}
(p \stackrel{a,r_i^P}{\rightarrow} p') \implies \\
\quad (\exists\, q' \in S_Q, \exists\, r^Q \cdot r^Q = \sum_{i=1}^{n} w_i * r_i^Q \wedge \\
\quad\quad (\exists\, i \cdot r_i^Q \leq r_i^P \wedge\ (\forall\, j \neq i \cdot r_j^P \leq r_j^Q \leq m_j)) \wedge q \stackrel{a,r_i^Q}{\rightarrow} q' \wedge (p', q') \in \mathcal{R})
\end{array}
$$

■

The above definition says that Q simulates P if Q can match every step of P by a step with the same label, and the accumulated usage value of at least one of the existing resource variables is decreased, while the resource-usage values of the other resource variables are at least the same as previously, or increased up to a given upper bound, m_j, respectively. Currently, we investigate how the simulation relation between two REMES services can be proved non-algorithmically, as there is no decidability result regarding computing a simulation relation between two PTA.

Discussion and Related Work. One might wonder about the benefit of using REMES in SOS behavioral descriptions, instead of other already existing approaches. If we take a look into code-driven approaches, such as WS-CDL [4] or BPEL [3], we notice that either the service behavioral description is missing, or it involves only externally observable or public aspects of service behavior. These approaches are valuable when the access to the service implementation exists, or in cases when the service is tailored to a specific model. REMES offers the possibility for a more abstract service description, including service behavior, suitable for early stages of systems development, even when no detailed design description

exists. UML profile MARTE (Modeling and Analysis of Real-Time and Embedded systems) provides means for specifying, designing, and verifying/validating system models, but lacks a formal behavioral description, which makes it hard to employ it for detailed analysis purposes. Rychly [7] describes the service behavior and structure found in service-oriented architectures (SOA) as CBS with features of dynamic and mobile architectures. The underlying formalism is based on $\pi - calculus$. Comparing to our approach, Rychly's work is more related to interface behavior and inner service communication and bindings, while REMES supports more detailed behavioral characteristics, including describing resource usage at modes and on edges, timing constraints (invariants), and discrete/timed actions.

Since it can be easily transformed into TA/PTA, the REMES language has been appealing for describing services, and it can been seen as an intermediate layer between higher levels of abstraction and detailed, formal behavioral models. Moreover, we have found it interesting to be able to manipulate different types of resources within the same service model, and carry out various types of analysis (i.e., feasibility analysis, trade-off analysis, and optimal/worst-case resource analysis), as described in [8], for SOS. Future work includes continuing the work on mechanizing the proof of the simulation relation between two PTA, as well as formalizing the notions of service composition and compositional analysis with REMES.

References

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

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

[3] 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.

[4] 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.

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

[6] 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.

[7] M. Rychly. 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.

[8] 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.

[9] 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.