Verifying Interaction Protocol Compliance of Service Orchestrations[⋆]

Andreas Schroeder and Philip Mayer

Ludwig-Maximilians-Universität München, Germany
{schroeda, mayer}@pst.ifi.lmu.de

Abstract. An important aspect of service-oriented computing is the ability to invoke services without knowledge of the actual implementation. This requires at least a description of the service interface; better yet is a specification of the complete interaction protocol. This applies to atomic services as well as service compositions. In both cases, however, guaranteeing that a service complies with the promised interaction protocol is crucial for deadlock-free communication. In this paper, we present an analysis method and tool for verifying compliance of service orchestrations with service interaction protocols given as UML models. Our method is part of a larger suite of methods and tools for model driven development of service oriented architectures covering code generation for the Web service stack and other service platforms: MDD4SOA.

1 Introduction

A core aspect of Service-Oriented Computing (SOC) is enabling (semi-) automatic discovery and composition of services. To achieve this aim, services are accompanied with a description of their interface, consisting of a description of accepted and sent messages and its interaction protocol detailing which interactions with the service are legal at a certain point in a conversation. Interaction protocols greatly help in assembling services and enable more thorough mechanical checkings of whether services fit together. However, for this checking to be helpful, it is crucial that service orchestrations conform to their interaction protocols; otherwise, services may be proven incompatible although fitting, or even worse, services may be considered compatible when they are not. Therefore, tools for checking the conformance of service implementation (in our case, the orchestration) and service description (i.e., the interaction protocol) are urgently needed.

In this paper, we present an approach for checking the conformance of service orchestrations given as UML activity diagrams and interaction protocols given as UML protocol state machines. We use the UML as it constitutes the lingua franca of software engineers, and furthermore allows to follow a model driven approach: to create platform independent software models that can be transformed into platform specific realizations.

The remainder of this paper is structured as follows. First, we give a brief overview over our UML profile for services in Sec. 2. In Sec. 3, we detail the basic ideas of

[⋆] This work has been partially supported by the EC project SENSORIA, IST-2005-016004.

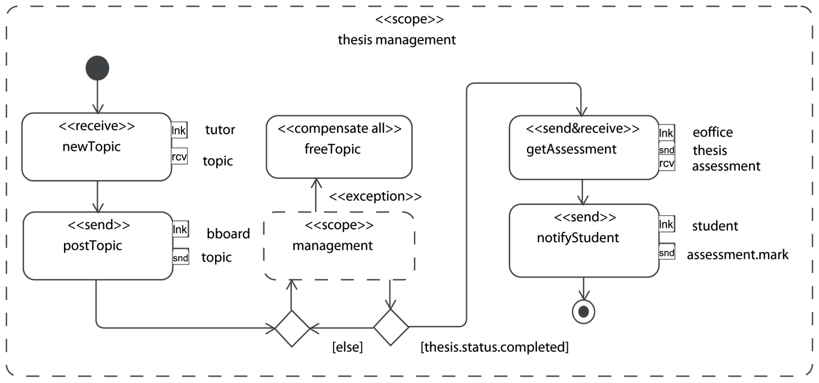


Fig. 1. Thesis Management: Service Orchestration.

our refinement analysis, and describe an accompanying tool. Finally, we present related work in Sec. 4, and conclude in Sec. 5.

The work presented in this paper has been developed in the EU project SENSORIA, which aims at developing a novel comprehensive approach to the engineering of service-oriented software systems. More about SENSORIA may be found in [14].

2 Modelling Services with UML4SOA

Our UML profile UML4SOA [9] enables convenient modelling of service-oriented systems on a platform independent level. So far, UML4SOA included stereotypes for modelling (a) service orchestrations only, and was extended for the purpose of conformance checking to address (b) service architecture modelling with UML composite structure diagrams, and (c) service interface descriptions using class diagrams (for the static part) and protocol state machines (for the dynamic part).

Service Orchestration. UML4SOA includes a profile for modelling service orchestrations with UML activity diagrams. The profile employs native UML2 elements where possible, e.g. for modelling branches, loops, and parallel control flow, and extends the UML2 with service-specific concepts like service calls as well as event- and compensation handling. For details on the extension for service orchestration, see [9].

Fig. 1 shows the orchestration of the student thesis management example taken from [9]. It models the process of handling a student thesis from the initial topic provided by a tutor, through managing the ongoing thesis writing process, to the final assessment and notification.

Service Composition Structure. The service composition structure describes the participants in a service oriented architecture and how services may be used by participants. For this, we use UML2 composite structure diagrams. To specify the services of a SOA, UML4SOA follows the UML2 component-based design approach: ≪service≫-stereotyped Ports represent services, and ≪service provider≫- stereotyped Components represent service providers, that is to say, software entities that expose and offer services to other software entities.

Service Interfaces. While the service composition structure gives an overview over the SOA infrastructure, it does not describe the interfaces of the involved services. For this task, we use UML2 class diagrams to model service providers, their services, provided operations, and required callbacks. Going beyond static service descriptions, we use UML2 protocol state machines to describe behavioural aspects of service interfaces. In this way, the intended usage can be specified more precisely than by relying on operation names and message types as done e.g. in WSDL.

UML protocol state machines (PrSM) are used to describe valid interaction sequences of a port with finite state machines consisting of protocol states and protocol transitions between them. Transitions of PrSMs may contain a precondition, event, and postcondition. In UML4SOA, we use events to model incoming and outgoing messages. Pre- and postconditions are not considered yet, but may be in the future.

Fig. 2(a) shows the protocol offered to a thesis tutor by the thesis management service provider. It specifies that the provider first requires to ≪receive≫ a newTopic message, and only allowing status queries (performed by ≪receive≫ getStatus and ≪send≫/≪reply≫ getStatus from the service) from a tutor once the topic was picked up by a student, which is invisible to the tutor an hence ≪internal≫. In addition to the correct protocol, we introduced a ≪receive≫ transition from and to topicPosted with the reception of a postTopic message, hence requiring that the topic for a thesis may be posted arbitrarily often; obviously, the orchestration does not satisfy this requirement, which is discovered by the conformance verification tool, as discussed in Sec. 3.

3 Analysing Service Orchestrations

Once a service orchestration and its service interface is modelled, the question arises whether the orchestration conforms to the defined protocol – which is part of the published service description. In more formal terms, the question is whether the orchestration constitutes a valid refinement of the protocol.

For this, we implemented an approach that (1) allows to select service providers and services for checking, (2) analyses the associated service orchestrations and interaction protocols for syntactic consistency, (3) allows to select failure configurations, (4) generates modal input/output labelled transition systems (modal I/O LTS, see [8] for the complete formal framework) from both service orchestrations and interaction protocols, (5) performs the conformance analysis and (6) outputs the results graphically.

Selecting service providers and services. The MDD4SOA conformance checker accepts UML XMI files and allows users to select, from the model, the services and service providers they would like to validate.

Checking Syntactic Consistency. Since not all UML activity diagrams may be translated into a modal I/O LTS, the MDD4SOA analysis tool checks the syntactic consistency of a given service orchestration before performing any refinement analysis, and informs the service engineer about syntactic inconsistencies found in the model. We cover general consistency constraints such as forbidding non-executable activities and MDD4SOA specific constraints restricting the branching and forking structure.

Selecting Failure Configurations. Before performing the conformance analysis, the tool allows the user to choose possible erroneous service interactions, i.e. service interactions that may fail due to unavailability of the communication partner or transmission errors. By this, the user is able to adjust the generated LTS and hence analyse the service orchestration conformance with varying fault assumptions. This feature emerged from practice, as conformance to reasonable service protocols is only achievable if some service interactions are assumed as being reliable.

Generating Modal I/O LTS. In order to reach an answer to the question of protocol conformance, a common semantic basis for service orchestrations and interaction protocols must be found. We chose as this common basis modal I/O LTS. Since the simple UML protocol state machines we use in our approach can be easily translated to modal I/O LTS, one half of the transformation is very straightforward.

Service orchestration models on the other hand use a subset of UML activity diagram constructs as well as the stereotypes introduced by UML4SOA. We defined a translation of service orchestration models to modal I/O LTS by following the widespread interpretation of activity diagrams as Petri nets in the sense that a state is a set of markers, and markers can be roughly interpreted as Petri net tokens (cf. Fig. 2(b) for an extract of the LTS generated for the thesis management orchestration).

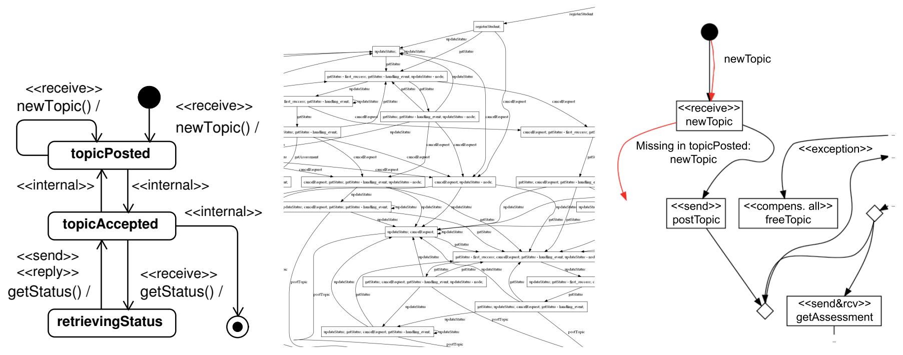
The modal I/O LTS that is constructed from a service orchestration S roughly consists of states representing possible execution states of S, and transitions from one execution state to its successor for each service interaction or event that may occur. An execution state of S consists of the orchestration S with markers of different kinds (e.g. “program counter” markers, “active” markers on currently processed scopes, or “completed” markers attached to activity nodes).

Performing Conformance Analysis. Now that both service orchestration and interaction protocol are represented as modal I/O LTS, we can verify whether a refinement exists using observational modal refinement as defined in [8].

Observational modal refinement is computed in the MDD4SOA conformance analysis tool by – starting from the pair of initial orchestration and protocol states – enumerating all alternative successor pairs using and-or trees. If a pair of states is found during process which violates the observational modal refinement conditions in one step (by not providing a required action or performing an illegal action), this fact is memorized within the and-or tree, and propagated to its predecessor nodes in a final step. All propagated paths that lead (together with others or alone) to a falsification of the initial pair represent execution traces that must be reported to the user, as they constitute violation traces of the refinement between interaction protocol and service orchestration.

Graphical Output. Considering the thesis management example, Fig. 2(c) shows how an error trace is displayed: the service orchestration does not support the reception of postTopic after it was once received. The analysis result states that the service orchestration fails to support the reception of a postTopic message once it completed the first postTopic receive activity.

The MDD4SOA conformance analysis tool is an open source Eclipse plugin available at mdd4soa.eu. The MDD4SOA tool suite also features code generation tools that allow the generation of e.g. BPEL/WSDL code from UML4SOA models.



(a) Wrong Tutor Protocol (b) Orchestration LTS (c) Resulting Analysis Output

Fig. 2. Analysing with UML4SOA

4 Related Work

Several proposals exists for modelling service-oriented systems defining own UML profiles for SOA or proposing a proprietary approach. For a comparison of these approaches to our UML4SOA profile, see [9].

Several other approaches to activity diagram semantics exist, of which a comprehensive overview can be found in [12]. Other tools for refinement analysis based on bisimulation exist, e.g. Ticc [1] and Chic [4], where the latter is also applicable to Web services [3]. Both tools however define their own textual notations for modelling.

Other approaches to analyzing services often focus on BPEL [5,7,6], XLANG [15] or own notations (based on petri nets, e.g. [13], finite state machines, e.g. [2], or process-calculi, e.g. [11]), entailing platform lock-in or a steeper learning curve for software engineers compared to our UML-based approach, respectively. Furthermore, most approaches focus the analysis of single services (e.g. verifying internal consistency of single BPEL processes) or on composability of services (e.g. checking whether a composition of BPEL processes is deadlock-free or satisfies other properties). However, conformance of services to behavioural descriptions will constitute a vital part in service engineering and deployment [11,10]. This may be related to the fact that the Web service stack does not yet offer means for behavioural description of Web services.

A particular close approach to ours is [15], which generates behavioural descriptions of orchestrations instead of checking their conformance. This is an interesting approach that however does not support top-down service engineering (first creating the description, then the realization) as a conformance verification approach does.

5 Conclusion

In this paper, we have addressed the problem of checking compliance of service orchestrations with their interactions protocols. We employ modal observational refinement based on a modal I/O LTS semantics of service orchestrations and service protocols, which allows to display informative error traces to the service engineer. Our work is built on an extension for UML2, the UML4SOA profile, which includes stereotypes for specifying both static and dynamic aspects of service compositions.

As conformance between service orchestration and interaction protocol is of crucial importance for (semi-) automatic service composition, we believe that the provided ability to mechanically verify and confirm that an orchestration follows its promised interaction protocol significantly eases achieving error-free communication between services. With the UML4SOA profile and the MDD4SOA tool suite, our approach offers straightforward modelling, checking, and generation of SOA artefacts.

References

1. B. T. Adler, L. de Alfaro, L. D. da Silva, M. Faella, A. Legay, V. Raman, and P. Roy. Ticc: A Tool for Interface Compatibility and Composition. In *18th Int. Conf. Computer Aided Verification*, volume 4144 of *LNCS*, pages 59–62. Springer, 2006.
2. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic Composition of E-Services that Export Their Behavior. In *1st Int. Conf. Service-Oriented Computing*, volume 2910 of *LNCS*, pages 43–58. Springer, 2003.
3. D. Beyer, A. Chakrabarti, and T. A. Henzinger. Web service interfaces. In *14th Int. Conf. on World Wide Web*, pages 148–159. ACM, 2005.
4. D. Beyer, K. Chatterjee, T. A. Henzinger, and F. Y. C. Mang. Chic: Checker for Interface Compatibility. www.eecs.berkeley.edu/˜arindam/chic.
5. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Compatibility Verification for Web Service Choreography. In *3rd Int. Conf. on Web Services*, pages 738–741. IEEE, 2004.
6. X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *3rd Int. Conf. on Web Services*, pages 621–630. IEEE, 2004.
7. M. Kovács and L. Gönczy. Simulation and Formal Analysis of Workflow Models. In *5th Int. Wshp. on Graph Transformation and Visual Modeling Techniques*, Electronic Notes in Theoretical Computer Science, pages 215–224. Elsevier, 2006.
8. K. Larsen, U. Nyman, and A. Wasowski. Modal I/O Automata for Interface and Product Line Theories. In *16th Eur. Symp. on Programming*, volume 4421 of *LNCS*, pages 64–79. Springer, 2007.
9. P. Mayer, A. Schroeder, and N. Koch. UML4SOA: Model-Driven Service Orchestration. In *12th Int. Enterprise Computing Conf.* IEEE, 2008.
10. Meredith, L.G. and Bjorg, S. Contracts and Types. *Comm. ACM*, 46(10):41–47, 2003.
11. G. Salan, L. Bordeaux, and M. Schaerf. Describing and Reasoning on Web Services Using Process Algebra. In *3rd Int. Conf. on Web Services*, pages 43–50. IEEE, 2004.
12. H. Störrle. Structured Nodes in UML 2.0 Activities. *Nord. J. of Comput.*, 11(3):279–302, 2004.
13. W. van der Aalst and M. Weske. The P2P approach to interorganizational workflows. In *13th Int. Conf. on Advanced Information Systems Engineering*, volume 2068 of *LNCS*, pages 140–155. Springer, 2001.
14. M. Wirsing, A. Clark, S. Gilmore, M. Hölzl, A. Knapp, N. Koch, and A. Schroeder. Semantic-Based Development of Service-Oriented Systems. In *26th Int. Conf. on Formal Methods for Networked and Distributed Systems*, volume 4229 of *LNCS*, pages 24–45. Springer, 2006.
15. A. Wombacher and B. Mahleko. Finding trading partners to establish ad-hoc business processes. In *Int. Conf. on Cooperative Information Systems*, volume 2519 of *LNCS*, pages 339–355. Springer, 2002.