A Model-Driven Approach to Service Orchestration

Philip Mayer, Andreas Schroeder, Nora Koch

Research Unit Programming and Software Engineering
Institute for Informatics, Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 München, Germany

{mayer, schroeda, kochn}@pst.ifi.lmu.de

Abstract

Software systems based on Service-Oriented Architectures (SOAs) promise high flexibility, improved maintainability, and simple re-use of functionality. A variety of languages and standards have emerged for working with SOA artifacts; however, service computing still lacks an effective and intuitive model-driven approach starting from models written in an established modeling language like UML and, in the end, generating comprehensive executable code. In this paper, we present a conservative extension to the UML2 for modeling service orchestrations at a high level of abstraction, and a fully automatic approach for transforming these orchestrations down to the well-known Web Service standards BPEL and WSDL.

1. Introduction

With the introduction of the Service-Oriented Architecture (SOA), formerly proprietary software systems are being opened up and made available as services. On top of these services, business processes and technical workflows are being (re-)implemented as compositions of services, which has come to be known as service orchestration. Service computing has quickly been embraced by both academy and industry, as it promises highly flexible software systems, simple re-use of functionality, and improved maintainability.

While model-driven approaches are already in use for object-oriented languages – for example, by employing engineering tools which offer code generation for Java from models written in the Unified Modeling Language (UML) – service-oriented design still falls short of effective and comprehensive domain-specific modeling and code generation tools. In order to support software engineers with intuitive and easy to adopt design and implementation techniques for service-oriented software, we propose (1) to extend the reach of UML2 to the modeling of SOA systems, and (2) to exploit so-designed models for creating running systems, in particular through code generation.

In this paper, we discuss both points – modeling of service orchestrations in UML2, and how to utilize these models for code generation with the target of the Web services standards WS-BPEL (Web Service Business Process Execution Language) [9] and WSDL (Web Service Description Language) [14].

UML2 is a well-known and mature language for modeling software systems, however it is strenuous right now to model SOA artifacts with UML2, as native support for service and service orchestration concepts is missing. We therefore introduce a UML extension for SOA – called the UML4SOA profile – which is a high-level domain specific language for modeling service orchestrations. One of the main goals of UML4SOA is minimalism and conciseness: service engineers should have to provide only as much information as necessary for the generation of code, and at the same time as little as possible in order to keep diagrams readable.

Based on this profile, we introduce a transformation mechanism from UML4SOA models to BPEL and WSDL, whose main aim is the generation of comprehensible and maintainable code. To achieve this goal, our approach follows the current evolution of BPEL 2.0: instead of using flows to represent the control flow, we employ structured BPEL constructs such as conditions and loops.

This paper is structured as follows: We will discuss current problems with modeling service orchestration using the UML in section 2, and present our UML2 profile to deal with these problems in section 3.

Section 4 then discusses a fully automatic transformation for creating BPEL and WSDL code out of UML2 orchestrations modeled as presented in section 3. We put our work into perspective in section 5, and conclude our findings in section 6.

2. Modeling Orchestration in Plain UML

UML2 is accepted as the de facto standard for the modeling of software systems. With its support for profiles, it comes with a very flexible extension mechanism that facilitates the definition of domain

specific languages (DSLs), rendering UML an excellent solution for modeling service-oriented architectures. For modeling the structure of SOAs, UML2 component diagrams and deployment diagrams can be used and extended in a straightforward way; more challenging is the task of modeling the behavior of service-oriented systems, in particular the orchestration of services.

Service orchestration is the process of combining existing services together to form a new service to be used like any other service. Service orchestrations introduce a set of key distinguishing concepts: partner services, message passing among requester and provider of services, long-running transactions, compensation, and events.

We select UML2 activity diagrams for the modeling of service orchestrations as we assume that business modelers are most familiar with this kind of dynamic behavior diagrams. Other workflow languages use similar graphical representations and petri-net like semantics [1].

As a running example to illustrate our approach, we have chosen an orchestration scenario from the eUniversity domain: we model the management process of a student thesis from the announcement of a thesis topic by a tutor to the final assessment and student notification.

In this orchestration scenario, a tutor provides a thesis topic that is announced to a black board regularly read by students. Once a student decides to pick up the topic, it is removed from the black board, and the student is registered at the examination office as working on this thesis topic.

The student now provides regular updates to the thesis, while the tutor is able to read the status. At the same time, the exercise office may request the cancellation of the thesis if e.g. the deadline for thesis submission elapsed. Upon cancellation of the thesis processing, the thesis topic is freed and re-posted to the black board, and the student is informed of the abnormal cancellation.

Once the thesis is completed, an assessment of the thesis is requested from the examination office. This request is dispatched by the office to the authorized supervisor of the thesis. Finally, the student is notified once the assessment of the thesis is received.

Modeling this orchestration example in plain UML2 (Figure 1) reveals the following shortcomings:

- It is not possible to restrict the set of valid callers – as needed e.g. to ensure that only the tutor is able to cancel the thesis – on an UML *AcceptCallAction*. All restrictions must be implemented manually (area 1).

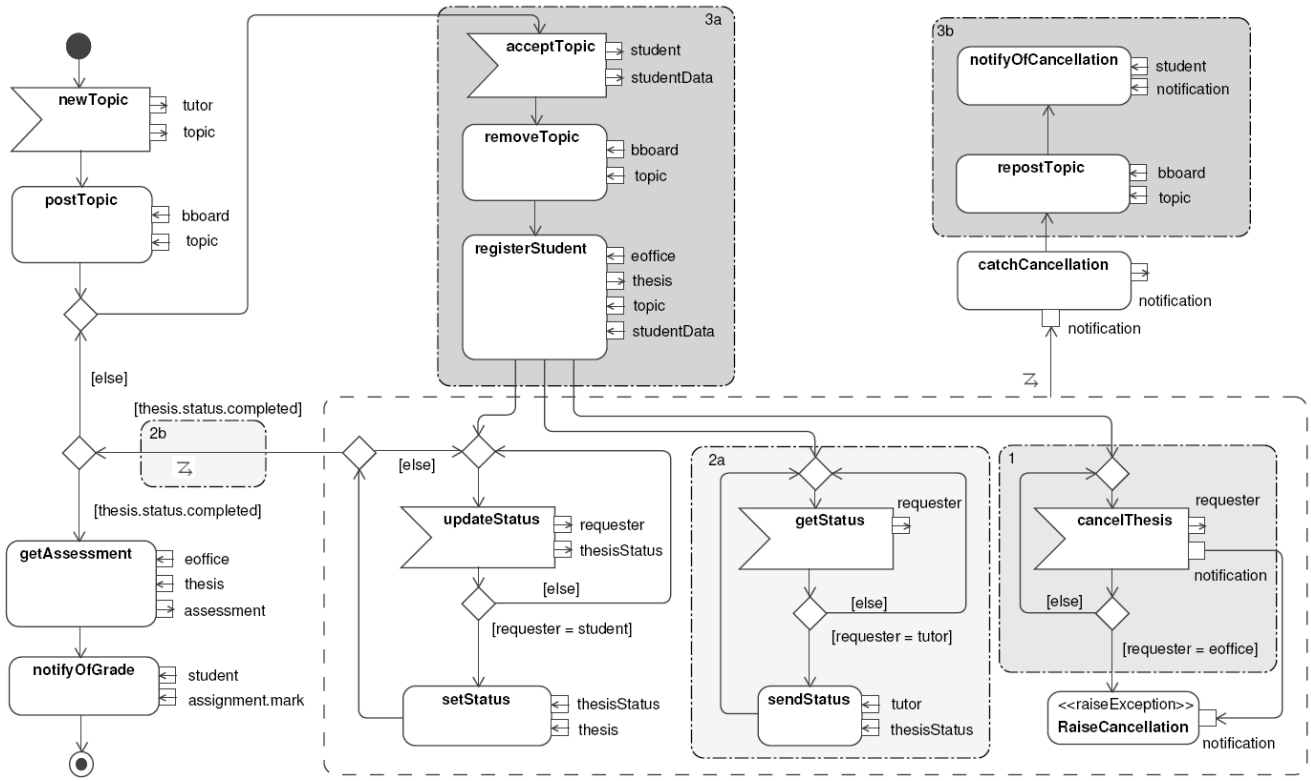

Figure 1: Thesis management modeled with plain UML

- Temporally enabled event handlers must be disabled using technical constructs. Russel et al. [11] suggest using *InterruptibleActivityRegions* containing the tasks to disable, and interrupting edges for normal task completion. Although this may be the best achievable solution with plain UML2 activity diagrams, using these technical constructs makes diagrams harder to understand (areas 2a and 2b).
- Similarly, the compensation for an activity is not associated directly with it, but programmed within explicit compensation logic. In addition, programming the compensation logic for more than one compensable activity is a tedious and error prone task [15] (areas 3a and 3b).

Due to these shortcomings, modeling service orchestrations with plain UML is a cumbersome task. At the same time, the resulting UML models are difficult to transform to orchestration skeletons for established service platforms, as the patterns used to handle the issues named above need to be recognized appropriately.

3. Modeling Orchestration with UML4SOA

To overcome the difficulties of modeling services with plain UML2, we extend the UML with service-specific model elements. Our UML extension is built on top of the Meta Object Facility (MOF) metamodel [9] and defined as a conservative extension of the UML metamodel. For the new elements of this metamodel, a UML profile is created using the extension mechanisms provided by UML2. The principle followed is that of minimal extension, i.e. to use UML constructs wherever possible and only define new model elements for specific service-oriented features and patterns making diagrams simple, consistent and easy to understand.

The resulting UML profile for service-oriented architectures (UML4SOA) provides model elements for structural and behavioral aspects, business goals, policies and non-functional properties of SOAs.

In this paper we present the metamodel (section 3.1) and the elements of the service orchestration part of the profile (section 3.2); for a complete overview of the extension the reader is referred to [7]. The orchestration scenario from the eUniversity domain is used to illustrate our approach and compare it to the model built with plain UML2.

3.1. UML4SOA Metamodel

For modeling orchestrations in UML2, we add specific service-aware elements for activity diagrams. The metamodel depicted in Figure 2 shows these model elements, their relationships with UML elements and the following relationships among each other:

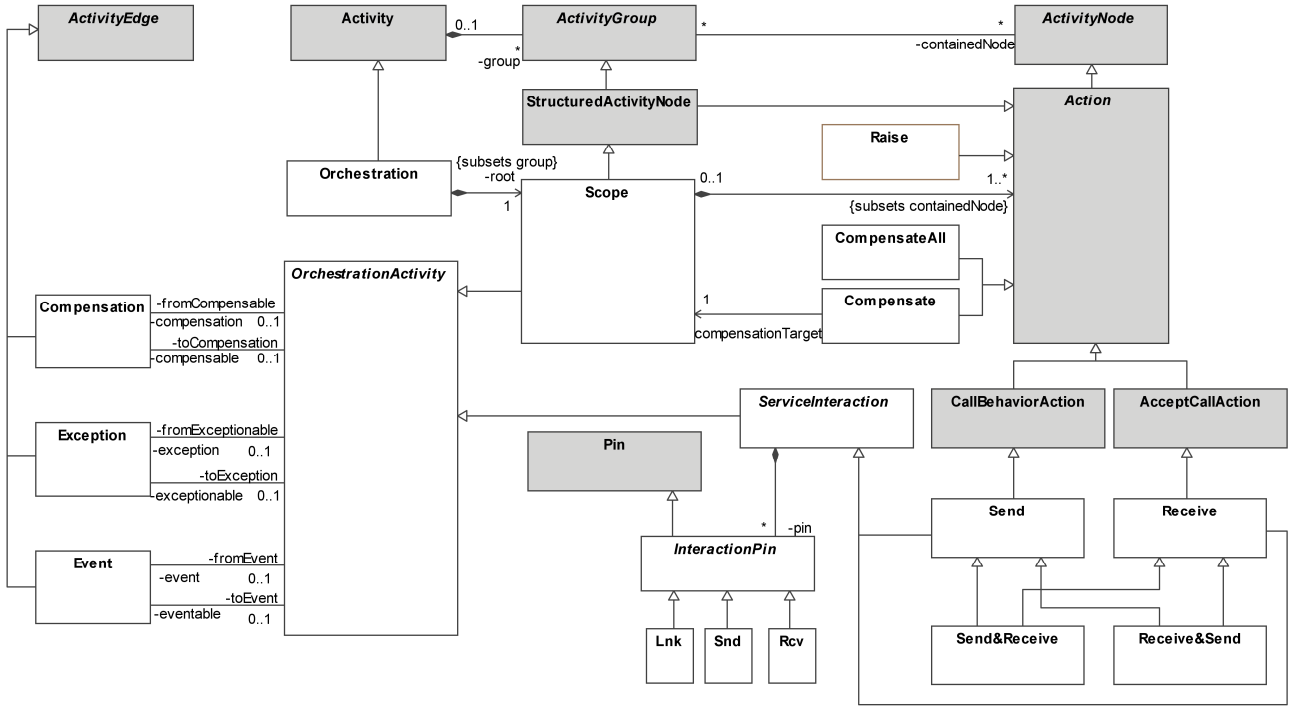

Figure 2: UML4SOA Metamodel (UML metaclasses in grey)

- The orchestration contains a root scope, which in turn contains all necessary elements for modeling the workflow of the orchestration.
- Four specialized actions have been defined for sending and receiving data.
- Service interactions may have interaction pins for sending or receiving data.
- Compensation edges link orchestration activities to actions or scopes that model the compensation.
- The Compensate and CompensateAll actions are used to trigger compensation of scopes or other actions.
- Event and exception handlers are used to handle emerging events and abnormal conditions, respectively.

Hence, the focus of the UML4SOA metamodel is on service interactions, long running transactions and their compensation and exception handling. This metamodel and the corresponding UML2 profile constitute the basis for model transformations and code generation defining a model-driven development process.

3.2. UML Profile for SOA

In order to be able to use the elements of the UML4SOA metamodel in UML2 tools, a UML profile must be specified by means of stereotypes and their relationships to the classes of the UML2 metamodel. The objective is to have a distinct graphical representation and clear semantics for service-oriented concepts.

The orchestration part of UML4SOA presented in this paper features constructs for modeling behavior of SOAs, i.e. stereotypes for service interaction based on the exchange of messages as well as compensation of services. A brief description of the most distinguishing stereotypes is given below.

- *scope*: A UML *StructuredActivityNode* that may have associated event, exception and compensation handlers.
- *send*: A UML *CallBehaviourAction* that sends a message. Does not block.
- *receive*: A UML *AcceptCallAction*, receiving a message. Blocks until a message is received.

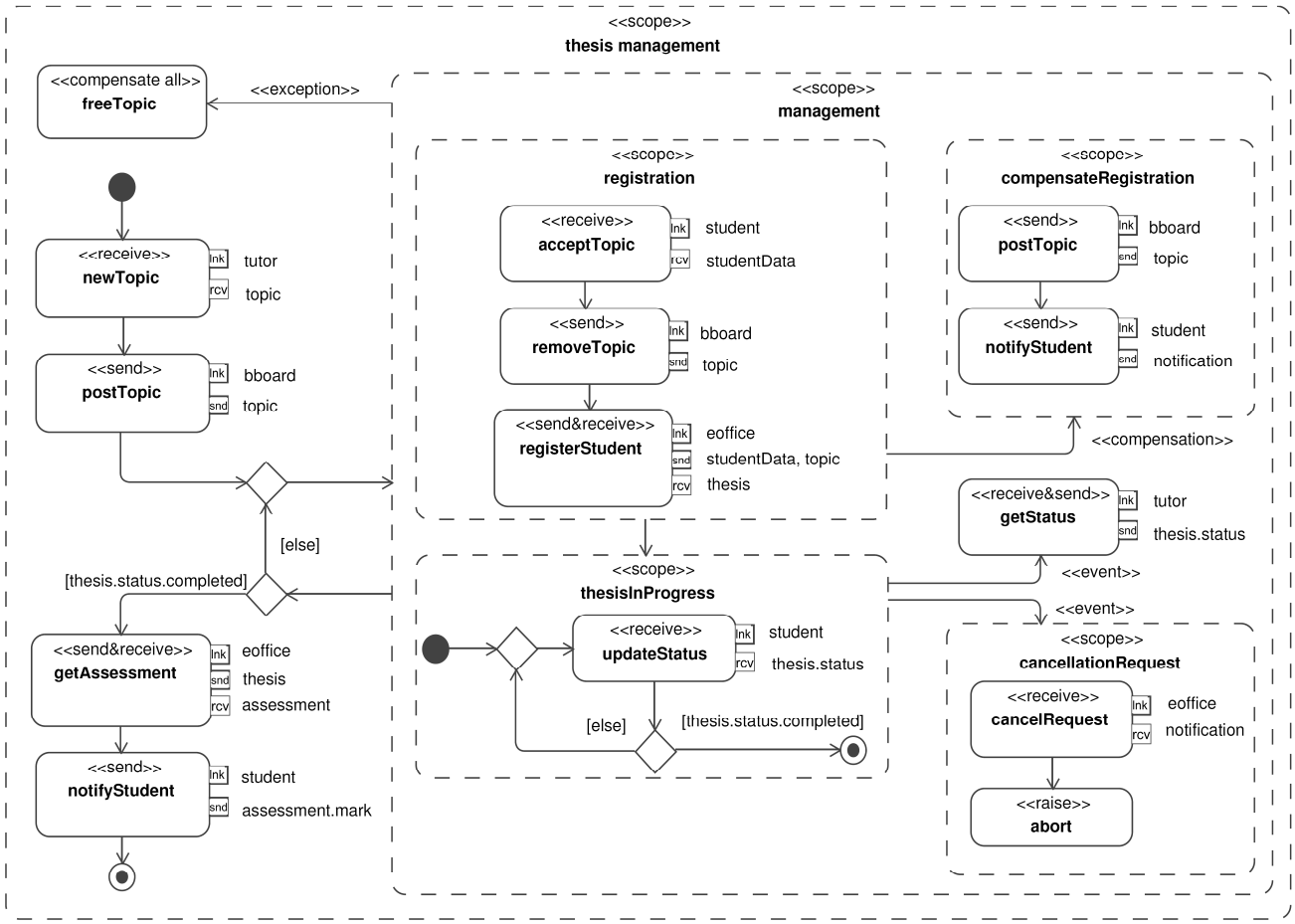

Figure 3: Thesis management modeled with UML4SOA

- *receive&send*: A UML *AcceptCallAction/ CallBehaviourAction,* denoting a sequential order of receive and send actions.
- *send&receive:* A UML *CallBehaviourAction/ AcceptCallAction*, denoting a sequential order of send and receive actions.
- *link*: A UML *Pin* that holds a reference to the service involved in the interaction.
- *snd*: A UML *Pin* that holds a container with data to be sent.
- *rcv*: A UML *Pin* that holds a container for data to be received.
- *exception*: A UML *ActivityEdge* to associate exception handlers to actions and scopes.
- *raise*: A UML *Action* that causes normal execution flow to stop and invokes associated exception handlers.
- *compensation*: A UML *ActivityEdge* to add compensation handlers to actions and scopes.
- *compensate*: A UML *Action* that triggers the execution of the compensation defined for a scope or activity.
- *compensateAll*: A UML *Action* that triggers compensation of the actually compensated scope (i.e. calling compensation on all subscopes in the reverse order of their completion). Can be inserted only in scopes defined for compensation.
- *event:* A UML *ActivityEdge* to associate event handlers to actions and scopes.

Figure 3 shows the example orchestration scenario again, this time modeled with the profile discussed above. The example shows that the control flow complexity is reduced considerably. In particular, all loops introduced for technical reasons become superfluous. Similarly, as UML4SOA offers specialized «event» edges, the use of exception edges to model completion of activity regions with event handlers become unnecessary. Using the service concepts defined in the UML4SOA profile reduces the number of edges from 32 to 23 and the number of decision nodes from nine to four, hence allowing the service modeler to focus on implementing service business logic instead of technical constructs. Thus, the value of the produced diagrams is increased for both human reading and automatic processing: the former profits from the conciseness and explicit – but minimalistic – labeling of constructs, while the latter profits from the simpler model structure.

4. Code Generation

The previous two sections have introduced a profile for modeling SOA orchestrations using UML2 activity diagrams. While these models have great value for communicating the orchestration workflow, they are not yet executable. In this section, we present a code generation approach for converting activity diagrams based on the UML4SOA profile to BPEL and WSDL.

4.1. Structuring the BPEL Process

There are basically two alternatives for converting activity diagrams to BPEL. The first alternative employs a graph-based BPEL process, i.e. creates a BPEL process with a *flow* activity as its root and only structured activity; the control nodes of UML2 – decisions, forks, and loops – are replaced with edge and activity guards. This yields another graph similar to the activity diagram; however, it ignores plenty of BPEL activities dedicated to structuring the orchestration, which would render it more readable. Indeed, with BPEL 2.0 there seems to be a shift towards a more structured approach to the modeling of processes, as more structuring activities have been added. Therefore, we have opted for the second alternative which is creating a structured BPEL process by converting the UML2 activity constructs to their BPEL equivalents – *if/elseif* for decisions, *flow* for forks, and *repeatUntil* for loops.

4.2. Partners of the BPEL Process

A BPEL process does not stand alone – it interacts with other services and is itself invoked by clients as a service. Thus, we also need to look at generating WSDL for describing partner services and the service provided by the BPEL process itself, and where to retrieve this information from the input UML model.

There are essentially two options for describing services along with their operations in UML: One option is to specify services and operations explicitly, for example by using class diagrams and component diagrams. Services and operations can then be referenced from within the orchestration. Another option is to infer the services and the roles they play in the process from the orchestration specification itself. This approach is particularly suited for rapid prototyping.

Our approach uses the second option, i.e. it is not necessary to specify any services or operations beforehand; they can simply be used as appropriate in the activity diagram. How a service is used in the orchestration defines its type:

- Some services are partners, i.e. the services are external to the orchestration and are called upon to perform some action.
- Some services are performed by the orchestration itself, i.e. the orchestration implements the functionality and offers it to partners.

The type is inferred from the use of the orchestration actions *send*, *receive*, *send&receive* and *receive&send*. There are three possibilities:

- If the orchestration only uses *send* (and *send&receive*) on a service, the service is clearly external to the orchestration and the orchestration itself is a client of the service. Thus, a WSDL service description needs to be generated which is to be implemented by the external service, and used by the BPEL process for invocation.
- If the orchestration only uses *receive* (and *receive&send*) on a service, the service is offered by the orchestration itself and the partner calls upon the service to perform some action. Thus, a WSDL description needs to be generated which is to be implemented by the BPEL process itself.
- Thirdly, the orchestration may use both *receive* and *send* on a service. In this case, a flow analysis is employed to find the initial interaction with the service. If the first interaction starts with a *receive*, we assume that the orchestration itself implements the service and then uses call-backs to send information back to the client. If the first interaction starts with a *send*, we assume that the service is external to the orchestration and uses call-backs to send information back to the orchestration. In both cases, we need to generate a service description which contains two port types – one for the service, and one for the client containing the call-backs.

4.3. The Transformation Algorithm

Having these prerequisites identified, we can move on to the actual transformation. The UML4SOA code generator uses a model-2-model approach, starting off with an XMI EMF model of the UML2 activity diagram, which can be read from XMI output which many UML modelers are able to produce, and converting to an EMF model of BPEL and WSDL, which can then be serialized to actual code. Thus, the code generator is, in effect, a model-2-model transformer.

The UML4SOA model transformer employs a depth-first rule-based approach for converting UML2 activity diagrams into BPEL and WSDL. In particular, we developed a partitioning algorithm which groups UML activity diagram nodes for implementation by a certain BPEL structured activity. There are three types of partitions which need to be identified in the UML source:

- Branches. Branching the control flow is modeled in UML with *decision* and *merge nodes*. In BPEL, branching is modeled with an *if* structured activity which may contain *elseif* branches for alternatives.
- Loops. We assume loops in the control flow to be modeled in UML with *merge* and *decision nodes*, with one control path leading from the decision at the end to the merge at the beginning. The equivalent BPEL construct for this is the *RepeatUntil* loop, which runs at least once.
- Parallel flows. Parallel execution is modeled in UML by using *fork* and *join nodes*. In BPEL, parallel flow is handled through the *flow* construct, which contains sequences for modeling sequential behavior inside each of the paths of the fork/join group.

Besides these induced partitions, we also exploit explicit structuring mechanisms. The UML4SOA profile already introduces the concept of a *scope*, which greatly eases structuring of activity diagrams and can be directly converted to a BPEL *scope*. The UML profile also allows handlers – exception, compensation, and event – to be attached to a scope. While these handlers are external to the scope in UML, they are defined within scopes in BPEL. Thus, the actions defined within the handlers in UML need to be moved to the appropriate code block inside the generated BPEL scopes.

Having handled structural aspects, there are also numerous smaller conversions to be done. As an example, we discuss handling of partner interaction actions. The UML4SOA profile discusses four actions for interactions with other services:

- *Send*. The send action is intended for sending a call to an external partner. It is modeled as a BPEL invoke with only an input variable.
- *Receive*. The receive action is intended for receiving incoming calls from external partners. It is modeled as a BPEL receive.
- *Send&Receive*. The send-and-receive action is intended for invoking an operation on a partner and receiving a result. It is modeled as a BPEL invoke with both an input and output variable.
- *Receive&Send*. The receive-and-send action first waits for an incoming call and then sends back the value of a predefined variable. It is modeled as a sequence of BPEL *receive* and *reply* actions.

As pointed out above, conversion of other activities such as compensation invocations and exception raising are simply converted to their BPEL equivalents same as the interaction actions.

4.4. Transformation Examples

As an example for the transformation, Figure 4 shows the BPEL code generated for the scope *registration* from the example introduced in the previous two sections. Namespace prefixes and some code have been removed to make the example easier to read.

```
...
<scope name="registration">

 <!-- Compensation Handler -->
 <compensationHandler>
    <!-- compensation code -->
 </compensationHandler>

 <!-- Actual scope code -->
 <sequence
       name="sequence inside registration">

  <receive name="acceptTopic"
        operation="acceptTopic"
        partnerLink="student"
        variable="student"/>

  <invoke name="removeTopic"
        operation="removeTopic"
        outputVariable="topic"
        partnerLink="bboard"/>

  <invoke name="registerStudent"
        inputVariable="thesis"
        operation="registerStudent"
        outputVariable="student&topic"
        partnerLink="eoffice"/>

 </sequence>

</scope>
...
```

Figure 4: BPEL code for scope registration

To give an overview of the created WSDL code, Figure 5 shows the relevant code generated for the partner bboard which has one port type and two operations, and is to be implemented by an external service and used by the orchestration.

5. Related Work

Several other attempts exist to define UML extensions for service-oriented systems.

The UML 2.0 profile for software services [6] provides an extension for the specification of services addressing structural aspects, but neither behavior of services nor orchestration of services is addressed in that work.

The work of Skogan et al. [13] has a similar focus as our approach, i.e. a model-driven approach for services based on UML models and transformations to executable descriptions of services. The main difficulty in the use of this approach lies in modeling the composition of services. Although they identify patterns to ease the transformations, the approach lacks an appropriate UML profile preventing building models at a high level of abstraction; thus producing overloaded diagrams.

```
...
<portType name="bboard service porttype">

   <operation name="postTopic">
     <input name="msg input topic"/>
   </operation>

   <operation name="removeTopic">
     <input name="msg input topic"/>
   </operation>

</portType>

<partnerLinkType
    name="bboard partnerLinkType">

    <role
       name="bboard role service"
       portType="bboard service porttype"/>

</partnerLinkType>
...
```

Figure 5: WSDL code for service student

The UML extension for service-oriented architectures described by Baresi et al. [4] focuses mainly on modeling such architectures by refining business-oriented architectures. The refinement is based on conceptual models of the platforms involved as architectural styles, formalized by formal graph transformation systems. The extension includes stereotypes for the structural specification of services. However, it does not introduce specific model elements for the orchestration of services.

In a recently published article, Ermagan and Krüger [5] extend the UML2 with components for modeling services defining a UML2 profile for rich services. Collaboration and interaction diagrams are used for modeling the behavior of such components. Neither compensation nor exception handling is explicitly treated in this approach.

In 2006, the OMG started an effort to standardize a UML profile and metamodel for services (UPMS). A first draft recently published [10] presents a set of requirements for such a profile and metamodel, a set of related profiles already defined within the scope of different projects by industrial and academic forums, and a first draft to an integrated solution for heterogeneous architectures. The current version only supports the concepts of service components, service specifications, service interfaces and contracts for services.

Another approach to model services is the Service Component Architecture (SCA) [12], which is not based on UML, but is strongly supported by the industry on its way to become an OASIS standard. It focuses on policies and implementation aspects of services. By contrast, Amsden [3] uses plain UML and focuses on the development process of services.

A first automated mapping of UML models to BPEL [2] defines a very detailed UML profile that introduces stereotypes for almost all BPEL 1.0 activities – even for those already supported in plain UML, which makes the diagrams drawn with this profile hard to read.

Several other approaches have been implemented for the automated transformation from UML to BPEL with the commonality of requiring very detailed UML diagrams from designers. An example is the UML profile described in [8], which defines BPEL-like stereotypes to handle data flow, but does not provide support for compensation. Conversely to these approaches, UML4SOA focuses on the improvement of the expressive power of UML by defining a small set of stereotypes for modeling SOA orchestrations.

6. Conclusion & Outlook

In this paper, we have presented the UML4SOA approach for modeling service orchestrations in UML2 and utilizing these models for code generation with the target of the Web Services standards BPEL [9] and WSDL [14].

The main advantage of our approach is the provision of a concise and intuitive solution to the modeling of services in UML: a UML2 profile with a small set of model elements that allow the service engineer to produce diagrams which on the one hand visualize an orchestration of services in a simple fashion, and on the other hand contain enough information for the generation of executable code.

Our translation to BPEL follows the current evolution of BPEL 2.0: using flows to represent the control flow is avoided in favor of more readable structured activity nodes such as conditions and loops.

We believe that being able to model service orchestrations in UML and generating executable code is an important step towards an effective model-driven development of services. We will continue to work on modeling and transformation of other service artifacts, in particular on modeling service interfaces and protocol specifications.

The UML4SOA profile and model transformer discussed in this paper are available for download on www.pst.ifi.lmu.de/projekte/uml4soa/.

Acknowledgements

Thanks go to Alexander Knapp for fruitful discussions on the UML4SOA profile. This research has been partially supported by the EC project SENSORIA "Software Engineering for Service-Oriented Overlay Computers" (6th Framework IST 016004).

References

[1] W.M.P. van der Aalst. "Chapter 10: Three Good reasons for Using a Petri-net-based Workflow Management System", Information and Process Integration in Enterprises: Rethinking Documents, Intl. Series in Engineering and Computer Science, Vol 428, Kluwer, 161–182, 1998.

[2] J. Amsden, T. Gardner, C. Griffin, S. Iyengar. "Draft UML 1.4 Profile for Automated Business Processes with a Mapping to BPEL 1.0", IBM, 2003, updated 27.12.05, ibm.com/developerworks/rational/library/content/04April/3103/3103_UMLProfileForBusinessProcesses1.1.pdf.

[3] J. Amsden. "Modeling SOA", IBM, 2007, ibm.com /developerworks/rational/library/07/1002_amsden.

[4] L. Baresi, R. Heckel, S. Thöne, D. Varró. "Style-Based Modeling and Refinement of Service-Oriented Architectures", Journal of Software and Systems Modeling (SOSYM), Vol 5 (2), Springer, 187-207, 2005.

[5] V. Ermagan, I. Krüger. "A UML2 Profile for Service Modeling". In Proc. of Int. Conf. on Unified Modeling Language, LNCS 4735 Springer, 360-374, 2007.

[6] S. Johnson, "UML 2.0 Profile for Software Services", IBM, www-128.ibm.com/developerworks/rational/ library/05/419_soa, 13.4.2005.

[7] N. Koch, P. Mayer, R. Heckel, L. Gönczy, C. Montangero, "UML for Service-Oriented Systems", SENSORIA D1.4a, 2007, www.pst.ifi.lmu.de/projekte/Sensoria/del_24/D1.4.a.pdf.

[8] K. Mantell. "From UML to BPEL", IBM, http://www.ibm.com/developerworks/webservices/library/ws-uml2bpel/, 2005.

[9] OASIS. "Web Services Business Process Execution Language", Version 2.0 (WS-BPEL 2.0). docs.oasis-open.org/wsbpel/2.0/, visited: 01-21-08.

[10] OMG. "UML Profile and Metamodel for Services", www.omg.org/docs/ad/07-06-02.pdf, visited: 10-01-07.

[11] N. Russel, A.H.M. ter Hofstede, W.M.P. van der Aalst, N. Mulyar. "Workflow Control Patterns. A Revised View", *BPM Center Report BPM-06-22*, 2006.

[12] SCA Consortium. "Service Component Architecture (SCA) Policy Framework", Version 1.0. 2007, ibm.com/developerworks/library/specification/ws-sca/.

[13] D. Skogan, R. Grønmo, I. Solheim. "Web Service Composition in UML", Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC'04), 47-57, 2004.

[14] W3C. "Web Services Description Language", Version 1.1, www.w3.org/TR/wsdl, visited: 01-21-08.

[15] M. Wirsing, A. Clark, S. Gilmore, M. Hölzl, A. Knapp, N. Koch, A. Schroeder. "Semantic-Based Development of Service-Oriented Systems". In Proc. of FORTE06, Paris, France, LNCS 4229, pp. 24–45. Springer, 2006.