Model-Driven Development of SOA Services

Christian Emig, Karsten Krutz, Stefan Link,
Christof Momm, Sebastian Abeck

Cooperation & Management, Universität Karlsruhe (TH), Germany
{ emig | krutz | link | momm | abeck } @ cm-tm.uka.de

Abstract. Service-oriented architectures (SOA) will form the basis of future information systems. Basic web services are being assembled to composite web services in order to directly support business processes. As some basic web services can be used in several composite web services, different business processes are influenced if for example a web service is unavailable or if its signature changes. Yet the range of such a change is often ambiguous due to a missing overall SOA service model pointing out the influence of services on business processes. In this paper we present a SOA service model defined as a UML-based metamodel and its integration into a model-driven service development approach. In contrary to existing approaches we explicitly address deployment issues

Problem Statement

With the evolution of service-oriented architecture (SOA) the focus in software development changes from applications to reusable services. These (atomic) services that offer coarse-grained functionality required for accomplishing the business processes and are then being assembled in a process-oriented way to composite services implementing fully automated and reusable parts of business processes [AH+03, LR02]..This approach allows for flexible adjustments in quickly changing business processes. Web services with the Web Service Description Language (WSDL) for interface description and SOAP as communication protocol are the most promising technologies for the implementation of SOA, but also other technologies like for instance CORBA are conceivable.

Concerning the development process for SOA a model-driven approach is commonly embraced. More precisely, various approaches for the mapping of business processes to an SOA-based IT support have been proposed [BM+04, KH+05]. Thereby, business processes are formally described in a notation which allows the automated mapping to an execution language and the execution by a process engine. As these kinds of execution language mainly facilitate the possibility for composing services in a process-oriented way, the development is also referred to as programming-in-the-large [Le03]. In the web service context, especially the Business Process Modeling Notation [OMG-BPMN] supports such a programming-in-the-large by introducing an adequate metamodel for specifying executable business processes [EW+06]. In case of BPMN, the abovementioned automatic mapping is already defined for the Business Process Execution Language (BPEL) [OASIS-BPEL], which represents the most prominent execution language for specifying executable business processes. Typically, an SOA has to support multiple business processes, which currently are specified by means of several independent BPMN models.
In this scenario of model-driven SOA development there are two problems which we address in this paper:

1. Due to the nature of SOA, particularly the atomic services are meant to be used in different business processes. There is no SOA service model yet depicting the overall usage relationships between business processes and services, while still supporting a process-oriented development. Hence, we introduce a UML-based SOA service metamodel that allows an explicit design of atomic and composite services along with the dependencies between them. This ensures consistency of functional dependencies within the integrated SOA.
2. The services designed and implemented within the development process are eventually operated and offered by a provider. At runtime several instances of one and the same service

implementation may exist. But so far, the modeling of relationships between the conceptual service design and the corresponding instances at runtime is not explicitly supported. In consequence, we extend our SOA service metamodel with deployment information, both for atomic and composite services to establish this link between service design and deployment.

The paper is organized as follows: section 2 introduces our solution, the SOA service metamodel consisting of the conceptual part, the deployable part and the formal definition as a UML profile. In section 3 we present a process-oriented methodology for designing an SOA on basis of our metamodels. The methodology is exemplified by means of a concrete scenario taken from the field of higher education. Section 4 provides a comparison of our service metamodel in relation to existing approaches. A conclusion and an outlook on future work in this area close the body of the paper.

SOA Service Metamodel

In this section our SOA service metamodel is introduced. This metamodel is supposed to allow a comprehensive modeling of SOA, including atomic and composite services along with the components implementing them. Furthermore, a distinction is drawn between a solely conceptual service model and a deployable service model which extends the conceptual model by deployment-specific information, like for instance the actual service endpoints. Figure 1 shows the conceptual part of the SOA service metamodel.

Conceptual SOA Service Model

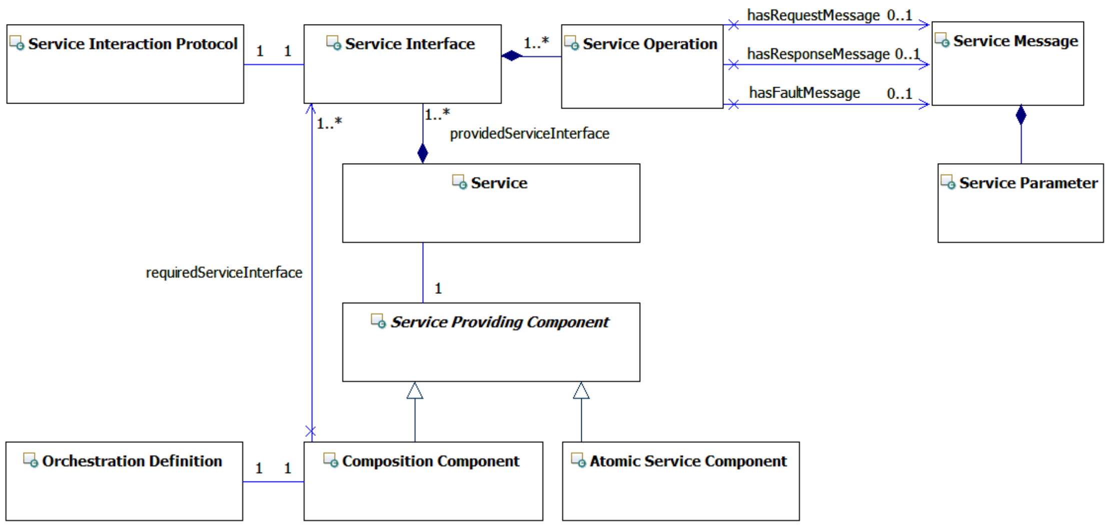


Figure 1: Conceptual Part of the SOA Service Metamodel

As stated before, within an SOA a general distinction is drawn between composite and atomic services. Composite services use the explicit composition functionality of SOA. In the following we first present the relevant concepts used to model both types of services.

The central element of our metamodel represents the *Service*. It provides a set of *Service Interfaces* each of them consisting of *Service Operations*. This containment relation is strictly enforced. The provision of *Service Interfaces* is modeled using the association *providedServiceInterface*. The usage view on a service is defined via the signatures of its *Service Operations* and the corresponding *Service Messages* which can be of type *Request Message* (incoming) or *Response Message* (outgoing). This is modeled using associations between *Service Operation* and *Service Message*. In this way, *Service*

Messages can be used both in different *Service Operations* and in different contexts: they can be *Request Messages* for one *Service Operation* and *Response Message* for another *Service Operation.* As a constraint, a *Service Operation* does either have to have a *Request Message* or a *Response Message*. Additionally, a *Fault Message* can be defined for each *Service Operation* which is used if an error occurs. Each *Service Message* consists of a set of *Service Parameters*; at least one has to be defined.

So far, we defined the external view on a service. The aforementioned elements do not describe the *Service's* functional part (i.e. the *Service Providing Component*) yet. This is why we put a n:1 association between a *Service* and its implementation, the (abstractly defined) *Service Providing Component*. In case of atomic services, this *Service Providing Component* is an *Atomic Service Component*, which basically relates to a traditional component artifact. The previously introduced elements allow the modeling of atomic services. Neither the specification of services composition nor the explicit modeling of dependencies between the composite and the included atomic services is supported yet. For this purpose, we introduce the *Composition Component* as a *Service Providing Component* that provides the implementation for the composite service. Unlike atomic services, the composition service's application flow (i.e. orchestration) is implemented using explicit SOA composition technology. The required information is held as the executable *Orchestration Definition*, for instance based on the Business Process Modeling Notation or UML Activity Diagrams as defined in UML Superstructure [OMG-Super]. In order to execute these definitions they have to be transformed to an executable language like BPEL, which are then being deployed on a BPEL engine. Concerning the *Service Interface*, there are no considerable differences between compositions and atomic services. Just as well, regular *Service Operations* are provided. However, regarding long-running compositions [MM+07], for example, a *Service Interaction Protocol* has to be additionally defined. This protocol is also referred to as the abstract process or orchestration and defines the sequences of operation invocations. Note that atomic services may be implemented as stateful services and therefore also require such a protocol specification [AC+04].

One essential feature of composite services represents the composition of already existing services to more complex services. Thereby, the included services may be either atomic or composite. Hence, the metamodel has to support the modeling of dependencies between service compositions and the included services. For this purpose, we introduce the association *requiredServiceInterface* which allows the linkage of a *Composition Component* with the required *Service Interfaces*. The *Orchestration Definition* in turn refers to the imported *Service Operations,* in case of BPMN for instance within the scope of embedded receive, reply or service tasks [EW+06].

Enhancing SOA Service Metamodel with Deployment Information

At this point, we are able to model atomic as well as composite services including the relationships between them in a purely conceptual way. The services are fully specified regarding their offered functionality along with the service providing components. However, in order to operate the services, additional deployment information is required. In consequence, for each conceptually specified *Service* there may be several *Deployable Services*. With the word "deployable" we express that the service model comprises additional deployment-relevant information, but the services do not have to be actually deployed yet. However, the deployment enhanced metamodel may form the basis for a corresponding (operational) deployment model, parts of which could be generated automatically.

Figure 2 shows the extensions of the previously presented metamodel required for specifying *Deployable Services*. The newly introduced elements extend their conceptual counterpart by deployment-relevant information. Note that the associations shown in Figure 2 are actually inherited from the conceptual elements, which – for the sake of clarity – are hidden in this diagram. A *Deployable Service Interface* for instance inherits all features of the related conceptual service interface, but is extended by the supported binding type and the service's endpoint reference. Each specified deployable element is associated with one distinct conceptual element via a designated association (e.g. *hasConceptual*). Compiling a model of the deployable services several constraints apply depending on the used element. These constraints do not only apply to the elements of the metamodel (M2 level according to the UML 4-layer metamodeling hierarchy [OMG-Infra], but also on the instantiated models (M1 level).

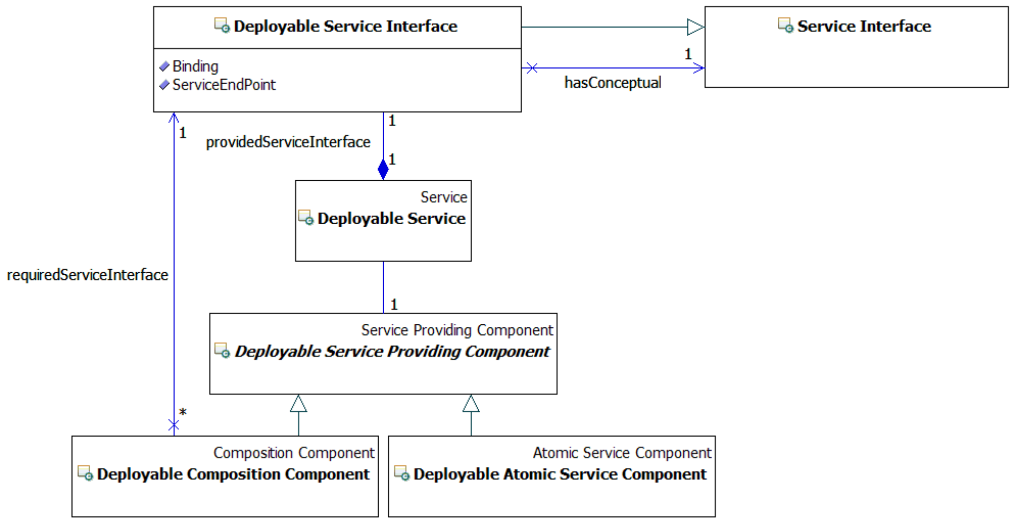


Figure 2: Deployable Part of the SOA Service Metamodel

- On the M1 level, a *Deployable Service Interface* which is associated with a *Deployable Service* has to comprise exactly the same features as the conceptual *Service Interface* belonging to the associated conceptual *Service*. For example, a *Deployable Service Interface* has to offer exactly the same *Service Operations* as the associated conceptual *Service Interface* on the M1 level. The same applies for *Deployable Atomic Services* and *Deployable Composition Components*.
- In case of a *Deployable Composition Component* only *Deployable Service Interfaces* may be included via the (inherited) association *requiredServiceInterface*.
- On the M1 level, for each conceptual *Service Interface* the corresponding *Composition Component* includes via the association *requiredServiceInterface*, the respective *Deployable Composition Component* requires exactly one *Deployable Service Interface* that corresponds to the included conceptual *Service Interface*. For example, if a *Composition Component* “c1” requires a *Service Interface* “s1” and there are two *Deployable Service Interface* for “s1”, namely “s1,1” and “s1,2”, a corresponding *Deployable Composition Component* “dc1,1” requires exactly one of them.

Using this extension for the (conceptual) SOA service model we are able to bridge the gap between a pure design model and an operational model. Furthermore, this approach conforms to the distinction that is drawn between the abstract and the concrete part within WSDL [W3C-WSDL]. Accordingly, the specified atomic *Deployable Services* hold all information needed for an automated generation of a fully-fledged WSDL along with skeletons for the specific implementation. In case of composite services, the specific BPEL deployment descriptor holding the binding information about the included partner’s endpoints can also be generated via parsing all corresponding associations of type *requiredServiceInterface*.

UML Profile

To be able to apply our service model in an UML-based software development process, it is a prerequisite to define a UML-Profile deriving our concepts to metaclasses defined by UML superstructure [UML-Super]. Thereby, we followed related approaches to SOA modeling and basically specified an extended component model. According to Table 1, we regard all kinds of *(Deployable) Service Providing Components* as specific types of components known from the UML metamodel. Thus, all the required stereotypes in this case – directly or indirectly – extend the UML metaclass *Component*. The *(Deployable) Service* itself extends the UML metaclass *Port*. As a service from an engineering point of view is often defined as a software entity that offers functionality in a standardized way [Le03], we regard the semantics of the element *Port* as most suitable. But in contrast to the UML component diagram, where several *Ports* may be attached to one *Component*, a *Service Providing Component* may only offer one *Service*. A *Service* on the other hand may be comprised of several *Service Interfaces*, which in turn offer several *Service Operations*. These stereotypes extend the corresponding UML metaclasses *Interface* and *Operation*.

Stereotype in Service Model	Extension of Metaclasses of UML Superstructure
(Deployable) Service	Port
(Deployable) Service Interface	Interface
Service Operation	Operation
(Deployable) Service Providing Component, (Deployable) Atomic Service Component, (Deployable) Composition Component	Component
hasConceptual, hasRequestMessage, hasResponseMessage, hasFaultMessage	Association
providedServiceInterface	Provided interface
requiredServiceInterface	required interface
Service Interaction Protocol	Sequence Diagram or Protocol State Machine
Orchestration Definition	Activity Diagram or BPMN.BPD
Service Message, Service Message Parameter	Class

Table 1: UML Profile for the SOA Service Metamodel

Due to the fact that a *Service Operation* in our case refers to different *Service Messages*, we did not directly use the accordant UML metaclass. Consequently, we also had to define a new stereotype for *Service Interface*, as this element may only provide such *Service Operations*. The same holds for the newly introduced associations *providedServiceInterface* and *requiredServiceInterface*, which extend the UML metaclasses *provided interface* and *required interface*. All the remaining custom associations are derived from the UML Kernel metaclass *Association*. In contrast to these straightforward profile extensions, for the elements *Service Interaction Protocol* and *Orchestration Definition* in each case several feasible options are conceivable.

According to [Jo05] the Interaction Protocol may be specified through a protocol state machine offered by UML. As an alternative to this approach, [BM+04] propose the employment of UML sequence diagrams for this purpose. Within this paper we limit the scope to stateless services, which do not require such an *Interaction Protocol*. A final decision in this matter is part of our future work.

The *Orchestration Definition* on the other hand may for instance be specified by means of UML activity diagrams. So the stereotype would extend the UML metaclass *Activity Diagram*. Unfortunately, activity diagrams are designed for a very general purpose. Unlike BPMN, the specific semantics of orchestration models is not regarded. But if BPMN were used for modeling orchestrations, these models could not be part of an integrated UML profile. Nevertheless, the different models could be synchronized through adequate transformations. This would be rather complex approach. With the introduction of the BPDM [OMG-BPDM] these discrepancies might be resolved. Therefore, a seamless integration of the *Orchestration Definition* into our SOA service metamodel will be part of our future work. In this paper we use models based on BPMN, which we created within scope of our preliminary work.

Process-Oriented Methodology for Developing an SOA

In this section we introduce a methodology for developing an SOA using our previously introduced SOA service metamodel. To demonstrate that our solution can be applied we like to present an example taken from a scenario as found at our university. Within an integration project the development and establishment of a university-wide service-oriented architecture is aimed. This architecture is to support business processes within the examination and course management in an integrated way and to provide a study assistance system to students, which also integrates offered services. Figure 3 depicts two business processes taken from the study assistance system which we will now explain in detail and use to demonstrate our approach later. As we focus the service relationships we do not explicitly model the exchanged service messages with service parameters.

Step 1: Identification and Modeling of Executable Business Processes

The first executable process in Figure 3 is named *Get Transcript of Records* and it enables a student to get his/her personal transcript of records. The process comprises the following steps: first, the matriculation status of the student is checked and core information like the name and address are returned. Second, the examination results are fetched and finally the transcript is returned. The main service operations in use are *Get Matriculation Status* and *Get Examination Results*.

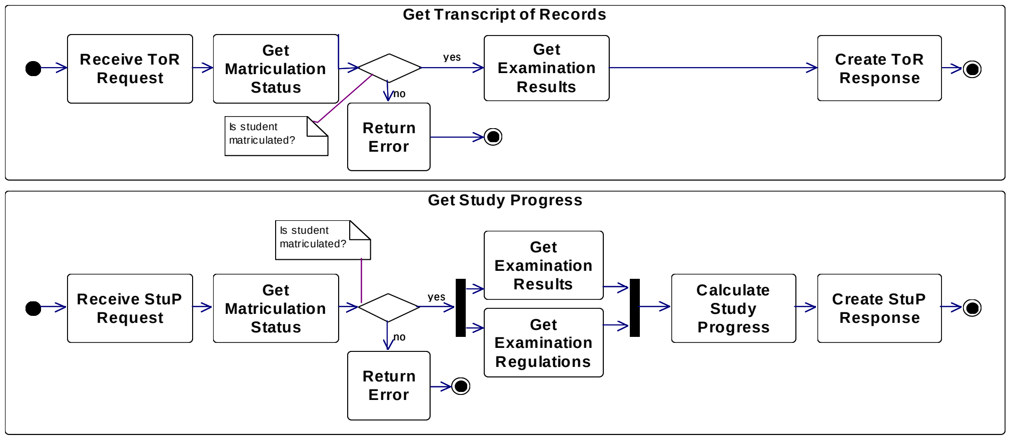


Figure 3: Examples of Fully IT-supported Processes

The second process *Get Study Progress* visualizes the progress the student has made with his/her studies so far. The following operations are required: first again the matriculation status of the student is checked in *Get Matriculation Status*. Next, two operations are executed parallel: the student's examination results and the examination regulations are fetched in *Get Examination Results / Get Examination Regulations*. Finally the results are mapped according to the regulations and the progress is calculated in *Calculate Study Progress*.

Step 2: Identification and Modeling of the Atomic Services

Having identified and modeled the executable processes, the next step comprises the identification and modeling of the required atomic services. Within the process models we already pointed up the necessary service operations. After equivalent operations have been identified, these consolidated operations have to be grouped to services. This grouping can for instance be accomplished by creating a service for each involved legacy system. If the services grow too large a further segmentation may be performed by means of the process they support, the coarse-grained modules of an existing application they belong to, or along the involved business objects in terms of Create, Read, Update and Delete (CRUD) operations [HZ05]. In our case we mainly decided on the latter approach. Accordingly, for the business objects handled within the processes, in particular "Student", "Examination Result" and "Examination Regulation", we defined CRUD-like services. Additionally, a service for calculating the study progress is needed.

According to Figure 2, we assign these services to components and depict the corresponding service interfaces. Here we start form atomic service self-contained components. The first component *StudentDB* provides the service *Student* that offers service operations like *Get Matriculation Status* via the Service Interface *Student*. The same applies do the next component *ExaminationResultDB* providing the service operation *Get Examination Results*. Finally the component *ExaminationRegulationDB* is providing the service operation *Get Examination Regulations* and the component *StudyProgressCalculator* facilitates the functionality to map the examination results of as student to the corresponding regulations.

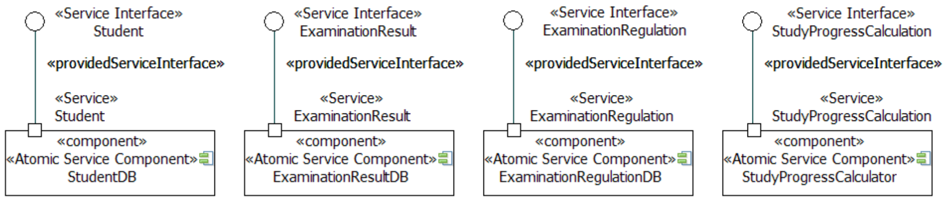

«Service Interface» Student
«providedServiceInterface»
«Service» Student
«component» «Atomic Service Component» StudentDB
«Service Interface» ExaminationResult
«providedServiceInterface»
«Service» ExaminationResult
«component» «Atomic Service Component» ExaminationResultDB
«Service Interface» ExaminationRegulation
«providedServiceInterface»
«Service» ExaminationRegulation
«component» «Atomic Service Component» ExaminationRegulationDB
«Service Interface» StudyProgressCalculation
«providedServiceInterface»
«Service» StudyProgressCalculation
«component» «Atomic Service Component» StudyProgressCalculator

Figure 4: Service Model of the Atomic Services

Step 3: Specification of Composite Services

In a next step these services are combined to composite services that implement the abovementioned executable processes *Get Transcript of Records* and *Get Study Progress.*

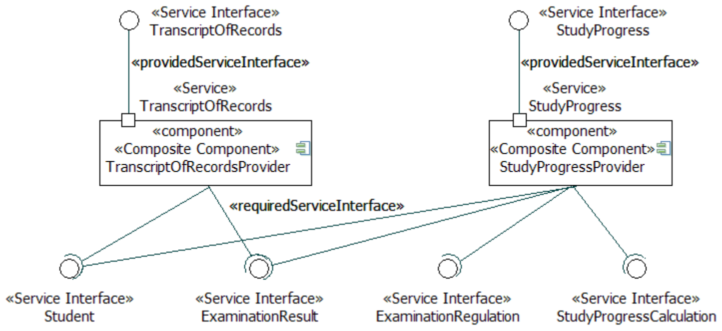

«Service Interface» TranscriptOfRecords
«providedServiceInterface»
«Service» TranscriptOfRecords
«component» «Composite Component» TranscriptOfRecordsProvider
«Service Interface» StudyProgress
«providedServiceInterface»
«Service» StudyProgress
«component» «Composite Component» StudyProgressProvider
«requiredServiceInterface»
«Service Interface» Student
«Service Interface» ExaminationResult
«Service Interface» ExaminationRegulation
«Service Interface» StudyProgressCalculation

Figure 5: Service Model for the Composite Services

As depicted above, for each defined executable process one *Composition Component* is created. As in both cases reading access to (more complex) business objects should be provided, the CRUD- *Service Operations* are offered through corresponding *Services*. The required atomic services are included via the association *requiredServiceInterface* connecting the *Composition Component* with the corresponding *Service Interfaces*. For the additionally needed *Orchestration Definition*, the already presented executable process models are used and extended by the concretely included *Service Interfaces* or *Service Operation* respectively. If the information on how the operations specified within the process models have been merged and group was available, this step could even be fully automated.

Step 4: Specification of Deployable Services

The final step in the SOA service modeling represents the extension of the previously created conceptual services by deployment information. As an example, Figure 6 shows the extended (deployable) model for the atomic service *ExaminationResult*. Thus, for this one conceptually modeled service, two *Deployable Services* should be offered. In consequence, different *ServiceEndPoints* are specified for the two *Deployable Service Interfaces*. The *Binding* in both cases is set to *SOAP*. The *Deployable Atomic Service Components* may be as well extended deployment-specific information, like for instance the additional information required to generate an deployment descriptors for application servers like BEA WebLogic, IBM WebSphere or Redhat JBoss AS. As one can observe, each deployable element is connected with its corresponding conceptual element via the association *hasConceptual*. On basis of this association, the two different models can be synchronized in case of changes, like for instance the specification of a further *Service Operation* within the conceptual part. Note that the specification of the synchronization algorithm is not in scope of this paper.

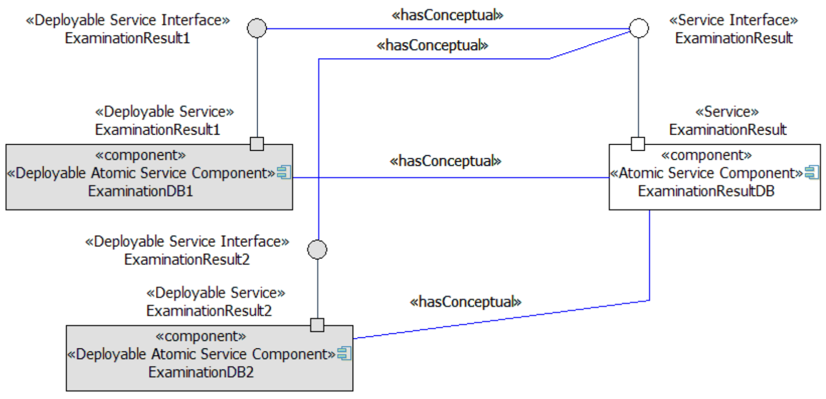

«Deployable Service Interface» ExaminationResult1
«hasConceptual»
«hasConceptual»
«Service Interface» ExaminationResult
«Deployable Service» ExaminationResult1
«Service» ExaminationResult
«component» «Deployable Atomic Service Component» ExaminationDB1
«hasConceptual»
«component» «Atomic Service Component» ExaminationResultDB
«Deployable Service Interface» ExaminationResult2
«Deployable Service» ExaminationResult2
«hasConceptual»
«component» «Deployable Atomic Service Component» ExaminationDB2

Figure 6: Deployable Service Model of an Atomic Service

Now, deployable versions of the composite services are created on basis of the deployable atomic services. The following figure provides an example for this procedure and shows a deployable version of the composite service *TranscriptOfRecords*.

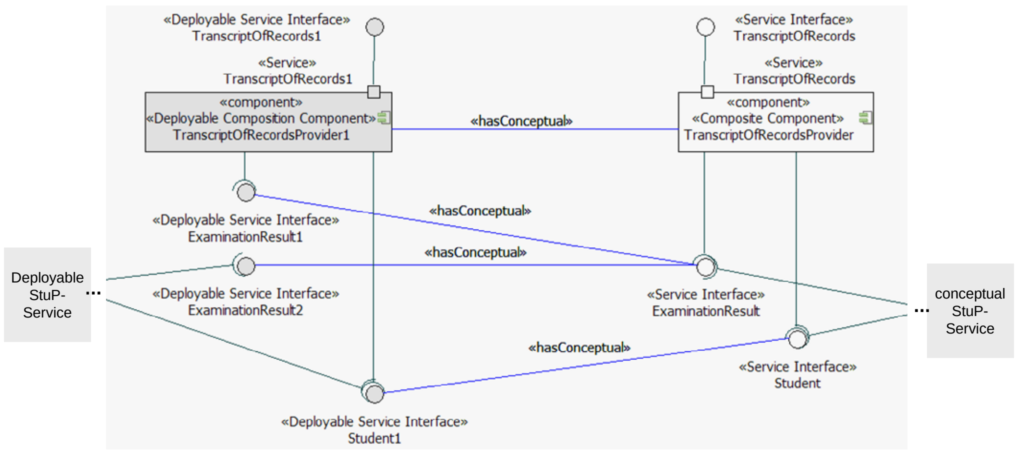
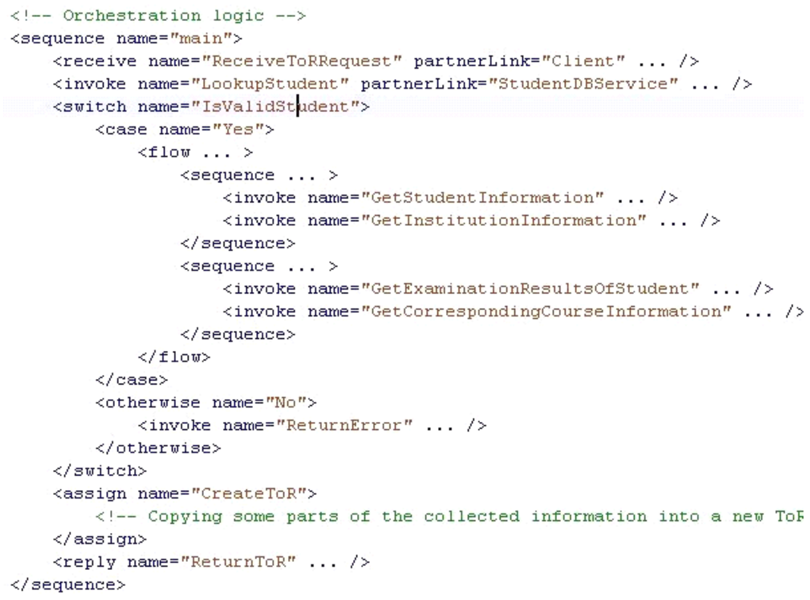

«Deployable Service Interface» TranscriptOfRecords1
«Service Interface» TranscriptOfRecords
«Service» TranscriptOfRecords1
«Service» TranscriptOfRecords
«component» «Deployable Composition Component» TranscriptOfRecordsProvider1
«hasConceptual»
«component» «Composite Component» TranscriptOfRecordsProvider
«hasConceptual»
«Deployable Service Interface» ExaminationResult1
«hasConceptual»
Deployable StuP-Service ...
«Deployable Service Interface» ExaminationResult2
«Service Interface» ExaminationResult
... conceptual StuP-Service
«hasConceptual»
«Service Interface» Student
«Deployable Service Interface» Student1

Figure 7: Deployable Service Model of a Composite Service

In this case, we introduce a *Deployable Service* named *TranscriptOfRecords1* for the respective conceptual composite service. According to the constraints defined within the metamodel, the corresponding *Deployable Composition Component* only includes *Deployable Service Interfaces* by means of the association *requiredServiceInterface*, in particular deployable versions of the services *ExaminationResult* and *Student*. Within our sample scenario, a second deployable version of the service *ExaminationResult* is linked to the deployable composite service *StudyProgress*. In doing so, a load distribution is achieved.

Step 5: Mapping to BPEL

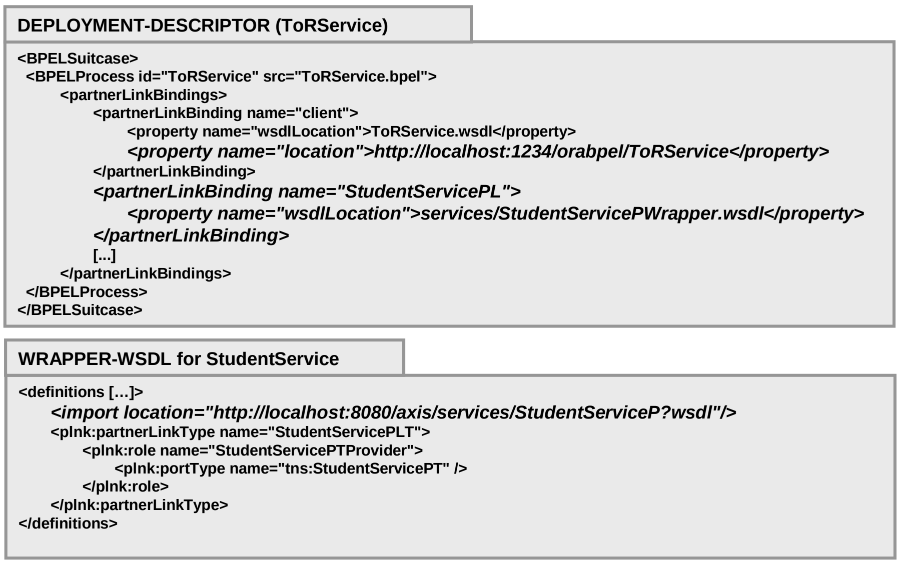
Using the process definition of Figure 3 we can automatically generate the relevant BPEL code. We exemplary show this code for the *TranscriptOfRecords*-Service in Figure 8. Note that the displayed code is fragmentary. Most of the attributes, variables and assigns are omitted for better readability. The complete error handling is also left out.



```
<!-- Orchestration logic -->
<sequence name="main">
    <receive name="ReceiveToRRequest" partnerLink="Client" ... />
    <invoke name="LookupStudent" partnerLink="StudentDBService" ... />
    <switch name="IsValidStudent">
        <case name="Yes">
            <flow ... >
                <sequence ... >
                    <invoke name="GetStudentInformation" ... />
                    <invoke name="GetInstitutionInformation" ... />
                </sequence>
                <sequence ... >
                    <invoke name="GetExaminationResultsOfStudent" ... />
                    <invoke name="GetCorrespondingCourseInformation" ... />
                </sequence>
            </flow>
        </case>
        <otherwise name="No">
            <invoke name="ReturnError" ... />
        </otherwise>
    </switch>
    <assign name="CreateToR">
        <!-- Copying some parts of the collected information into a new ToF
    </assign>
    <reply name="ReturnToR" ... />
</sequence>
```

Figure 8: Generated BPEL Code for TranscriptOfRecords-Service

To execute a BPEL process a BPEL engine is needed which parses the BPEL code and executes the contained instructions. Examples of existing BPEL engines are Oracle BPEL Process Manager and ActiveBPEL by ActiveEndpoints. All engines have in common that the BPEL process, which has to be deployed itself as well, needs to be supplemented. Of course the general BPEL code is always the same regardless which BPEL engine is used because it is standardized. But in practice the deployable BPEL packages differ from engine to engine. For instance, an engine-specific so-called deployment descriptor is additionally needed in order to execute the process. Using our approach, we can automatically generate the necessary deployment descriptors along with the required wrapper services, which extend the original WSDL by BPEL-specific information about the provided partner links. The resulting files are illustrated in Figure 9. Now the services can be automatically deployed and they are ready for use.



DEPLOYMENT-DESCRIPTOR (ToRService)

```
<BPELSuitcase>
  <BPELProcess id="ToRService" src="ToRService.bpel">
      <partnerLinkBindings>
          <partnerLinkBinding name="client">
              <property name="wsdlLocation">ToRService.wsdl</property>
              <property name="location">http://localhost:1234/orabpel/ToRService</property>
          </partnerLinkBinding>
          <partnerLinkBinding name="StudentServicePL">
              <property name="wsdlLocation">services/StudentServicePWrapper.wsdl</property>
          </partnerLinkBinding>
          [...]
      </partnerLinkBindings>
  </BPELProcess>
</BPELSuitcase>
```

WRAPPER-WSDL for StudentService

```
<definitions [...]>
    <import location="http://localhost:8080/axis/services/StudentServiceP?wsdl"/>
    <plnk:partnerLinkType name="StudentServicePLT">
        <plnk:role name="StudentServicePTProvider">
            <plnk:portType name="tns:StudentServicePT" />
        </plnk:role>
    </plnk:partnerLinkType>
</definitions>
```

Figure 9: Generated Deployment Descriptor and WrapperWSDLs for ToRService

Competitive approaches including comparison with selected approach

In the following we present other research initiatives that focus model-driven approaches for the development of services. The OASIS Reference Model for Service-Oriented Architecture [OASIS-SOA] is rather a guideline for the creation of a SOA model than a formal metamodel and thus cannot be used by itself for modeling SOA. The Object Management Group (OMG) currently tries to put together different efforts. Therefore they issued the request for proposal “UML Profile and Metamodel for Services” [OMG-UPMS]. As the submission date mid of June 2007 is not over yet, the OMG so far cannot provide a standardized approach. There are some approaches which define quite easy metamodels for the development of web services by simply creating a UML profile for WSDL. Examples of this category are [MC+03] and [JL+05]. A more comprehensive approach which provides additional modeling capabilities is provided by IBM [Jo05]. In Table 2 the description elements of the WSDL profile approaches and the IBM approach are compared with the proposed service description metamodel's description elements of this paper.

For the comparison the WSDL description elements are used in the notation of WSDL version 1.1. These description elements are basically all covered by our SOA service model. Only the *Service Parameter* in fact comprises several WSDL description elements. More precisely, the *Service Parameter* needs to provide means for describing the *Types* definitions referenced by *Elements* that in turn are subsumed to WSDL *Part*. As it extends the UML metaclass *Class*, the *Elements* can be mapped to the *Service Parameter's* attributes and the *Types* definition to the attribute type and its definition. Considering this information, the proposed SOA service model is WSDL compliant.

Concerning IBM’s service model, our model covers the majority of modeling elements. Therefore, only the differences are discussed here. The *Message Attachment* is one modeling element of the IBM model that is not explicitly treated by the proposed service model. If this concept is required, it may easily be included into the *Service Parameter,* which offers a certain degree of flexibility due to its inheritance of the UML metaclass *Class*. Furthermore, the IBM service model introduces the modeling elements *Service Consumer, Service Partition, Service Gateway* and partially *Service Provider*, which are not included in our service model.

WSDL	IBM Service Model [Jo05]	Our Service Model
Service	Service	Service
Port		ServiceEndPoint
Binding	Service Channel	Binding
Porttype	Service Specification	Service Interface
Operation	Operation	Service Operation
Message - input - output - fault	Message - - -	Service Message - hasReqestMessage - hasResponseMessage - hasFaultMessage
Part	-	Service Message Parameter
Element	-	
Types	-	
-	Message Attachment	-
-	Protocol	Service Interaction Protocol
-	Service Collaboration	Orchestration Definition
-	Service Provider	Service Providing Component
-	Service Consumer	-
-	Service Partition	-
-	Service Gateway	-

Table 2: Comparison WSDL / IBM / Own Approach

These modeling elements particularly serve the purpose to support an IBM-specific security model. The basic idea thereby is to assign the services to virtual organizational units (*Service Partitions*). The intersection between these *Service Partitions* is realized using so called *Service Gateways*. As there are other security models that do not require such a partitioning into virtual organizational units, this aspect is not treated within the proposed service model yet. An adequate extension for our service model that also allows the modeling of security-related aspects will be part of our future work.

In summary, the introduced SOA service model covers all relevant aspects of the IBM service model as well as WSDL. In addition, it allows us to draw a distinction between abstract and concrete services and enables modeling the dependencies between services of both kinds.

Current Status and Next Steps

In this paper, we presented a first step towards a comprehensive SOA service metamodel enabling model-driven development of both atomic and composite SOA services. It allows the explicit design of service composition and ensures consistency of functional dependencies within an integrated SOA design. Additionally, it helps bridging the gap between the service model and the business process model as wells as the service's implementation and eventually the deployment model by enriching our SOA service metamodel with deployment information for both atomic and composite services.

However, there are still some remaining issues that are not addressed so far. For instance, the seamless integration of the orchestration definition is not fully solved yet. The employment of BPMN certainly is a feasible approach, but it also causes difficulties regarding the synchronization with the UML-based models. In consequence, we would prefer the integration of BPDM, which at least is build on parts of the UML metamodel. To support a fully automated transformation we furthermore aim at an OCL-based formalization of the presented constraints and the QVT-based specification of transformations of the services along with the service providing components to WSDLs as well as skeletons of the implementing classes. Finally, we plan to enhance our SOA service metamodel by cross-cutting concerns like identity management and process management. This should allow integrating these aspects in the same model-driven way. Hence, we will extend our SOA service metamodel with the additionally needed elements.

References

[AC+04] G. Alonso, F. Casati, H. Kuno, V. Machiraju: Web Services – Concepts, Architectures and Applications, Berlin, Heidelberg, Springer-Verlag 2004.

[AH+03] van der Aalst, Wim M. P.; ter Hofstede, A. H. M.; Weske, M.: Business Process Management: A Survey. In: Lecture Notes in Computer Science Band 2678. Springer-Verlag, Berlin, 2003,

[BM+04] Bernhard Bauer, Jörg P. Müller, and Stephan Roser: A Model-Driven Approach to Designing Cross-Enterprise Business Processes, University of Augsburg, 2004.

[Er04] Thomas Erl: Service-Oriented Architecture: Concepts, Technology and Design, Prentice Hall PTR, ISBN 0-13-185858-0 August 2004

[EW+06] Christian Emig, Jochen Weisser, Sebastian Abeck: Development of SOA-Based Software Systems – an Evolutionary Programming Approach, IEEE Conference on Internet and Web Applications and Services ICIW'06, Guadeloupe / French Caribbean, February 2006.

[HZ05] Henkel, M., Zdravkovic, J., "Approaches to Service Interface Design", Proceedings of the Web Service Interoperability Workshop, First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'2005), Hermes Science Publisher, Geneva, Switzerland, 2005.

[JL+05] Jiang J, Lipponen, J., Selonen, P. & Systa, T. UML-Level Analysis and Comparison of Web Service Descriptions. Ninth European Conference on Software Maintenance and Reengineering (CSMR'05), 2005.

[Jo05] Simon Johnston: UML 2.0 Profile for Software Services, IBM develperWorks, April 2005. http://www-128.ibm.com/developerworks/rational/library/05/419_soa

[KH+05] J. Koehler, R. Hauser, S. Sendall, M. Wahler: Declarative techniques for model-driven business process integration, IBM Research Division, Zurich Research Laboratory, 2005.

[Le03] Frank Leymann: Web Services - Distributed Applications without Limits, Business, Technology and Web, Leipzig, 2003

[LR+02] Frank Leymann, Dieter Roller, M.-T. Schmidt: Web Services and business process management. In: IBM Systems Journal (2002) 41, S. 198-211, 2002.

[MC+03] Esperanza Marcos, Valeria de Castro, Belén Vela: Representing Web Services with UML: A Case Study, In Service-Oriented Computing - ICSOC 2003, pages 17-27, Springer Berlin / Heidelberg.

[MM+07] Christof Momm, Robert Malec, Sebastian Abeck: Towards a Model-driven Development of Monitored Processes, 8. Internationale Tagung Wirtschaftsinformatik (WI2007), Karlsruhe, Februar 2007.

[OASIS-BPEL] Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language (WSBPEL), Version 1.1, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel .

[OASIS-SOA] OASIS Reference Model for Service Oriented Architecture, Committee Specification 1, August 2006. http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

[OMG-BPDM] OMG:Business Process Definition Metamodel, Januar 2004.

[OMG-BPMN] OMG: Business Process Modeling Notation (BPMN), Version 1.0, BPMI.org, February, 2006.

[OMG-Infra] OMG:UML 2.0 Infrastructure Specification, OMG Adopted Specification ptc/03-09-15, September 2003. OMG

[OMG-Super] OMG: Unified Modeling Language: Superstructure version 2.0 formal/05-07-04, OMG, http://www.omg.org/docs/formal/05-07-04.pdf, August 2005

[OMG-UPMS] OMG Request For Proposal: UML Profile and Metamodel for Services (UPMS), OMG Document: soa/2006-09-09. http://www.omg.org/docs/soa/06-09-09.pdf, 2006.

[Ry03] Arthur Ryman: Understanding Web Services, IBM Technical Article, July 2003, http://www-128.ibm.com/developerworks/websphere/library/techarticles/0307_ryman/ ryman.html

[W3C-WSDL] W3C: Web Services Description Language (WSDL), version 1.1, http://www.w3.org/TR/wsdl, May 2001.