Derivation of Web Service Implementation Artifacts from Service Designs Based on SoaML

Michael Gebhart
Gebhart Quality Analysis (QA) 82
Karlsruhe, Germany
michael.gebhart@qa82.de

Jaouad Bouras
ISB AG
Karlsruhe, Germany
jaouad.bouras@isb-ag.de

***Abstract*—The increasing complexity of service landscapes requires a detailed planning that considers wide-spread quality attributes, such as loose coupling between services and their autonomy. To support this planning task, the Object Management Group standardized the Service oriented architecture Modeling Language for designing services and entire service-oriented architectures. In order to use the service designs modeled using SoaML within a model-driven development process, the created service designs have to be used to derive web service implementation artifacts. However, mapping rules described nowadays do not consider the SoaML design artifacts or do not consider service designs as a whole. In this article, mapping rules are identified and enhanced to transform service designs into web service implementation artifacts. The transformation rules are exemplarily applied to implement a service-oriented workshop organization system.**

Keywords-service design; SoaML; web service, implementation; derivation.

I. INTRODUCTION

This article is an extension of the work presented in [1]. Due to the increasing number of applications within Information Technology (IT) landscapes, the integration of these applications is an important success factor when realizing new functionality. For that purpose, service-oriented architecture (SOA) evolved as architecture paradigm [2] to create a flexible and maintainable IT. These strategic goals are massively influenced by the design of the building blocks of an SOA, the services. Quality attributes, such as loose coupling and autonomy [3], have been identified that impact flexibility and maintainability as higher-value quality attributes [4]. In order to ensure their fulfillment, a detailed planning is necessary.

For that purpose and for reducing the complexity when designing services, the Object Management Group (OMG) standardized a new language for designing services and entire service-oriented architectures, the Service oriented architecture Modeling Language (SoaML). The standard is vendor- and tool-independent and provides a meta model and a profile for the Unified Modeling Language (UML). As UML profile SoaML adds several stereotypes that focus on the specifics when designing services. Currently, SoaML is released in version 1.0.1 and is already supported by several tool vendors. Also some vendors already replaced their proprietary UML profiles with SoaML, such as IBM [4].

In order to use SoaML as language within a model-driven development process for services in particular web services as introduced by Hoyer et al. [5], a derivation of web service implementation artifacts from service designs based on SoaML is necessary. For that purpose, mapping rules have to be formalized that describe the relation between constructs of the modeled service designs and the generated final implementation. Furthermore, they constitute the basis for automatic transformations that can be embedded into software development tools. The mapping rules have to consider the underlying concepts so that the characteristics of the service designs are reflected by the web services. This is an important aspect for mapping rules, because for example when quality attributes have been considered during the service design phase, such as introduced by Gebhart et al. [1][6][7], the mapping rules are then expected to create web services that again fulfill these quality attributes.

This article analyses proposed mapping rules for creating web service implementation artifacts from service designs based on SoaML. As languages for web service implementation the Web Service Description Language (WSDL) and XML Schema Definition (XSD) are chosen to describe the service interface and included data types. Furthermore, Service Component Architecture (SCA) as component model, and Business Process Execution Language (BPEL) for the implementation of composed services are considered. In a first step, existing rules are analyzed. Since SoaML is available as a UML profile there exist a lot of rules, for instance to create data types based on XSD from UML Classes that can be reused. Afterwards, these rules are extended to support the service designs as a whole. To illustrate the mapping process introduced above, web service designs describing a workshop organization system have been designed using SoaML regarding wide-spread quality attributes.

The article is organized as follows: Section II introduces the concept of service designs and their creation using SoaML. Furthermore, in this section, existing mapping rules and their applicability for service designs are analyzed. In Section III, the scenario of the workshop organization is illustrated and its functioning especially through the created service designs is described. In Section IV, these service designs are mapped onto web services using the prior created mapping rules. Section V concludes this article and introduces future research work.

II. Related Work

This section describes the fundamental terms and existing work in the context of specifying service designs and their mapping onto web service implementation artifacts based on XSD, WSDL, BPEL, and SCA.

A. Service Design

According to Gebhart et al. [7][8] and Erl [9], a service design consists of a service interface as external point of view and a service component fulfilling its functionality. In order to formalize service designs and to enable their transformation into implementation artifacts, Mayer et al. [9] introduce a UML profile for describing behavioral and structural aspects of service interactions. Similarly, within the SENSORIA project [10] a UML profile for the service interaction is specified. Also IBM [11] introduced a UML profile for modeling software services. Even though all of these UML profiles enable the modeling of services they lack in acceptance as they are not standardized. For that reason the OMG decided to work on a standardized UML profile [12] and a meta model to formalize service-oriented architectures and their services. As a result, SoaML has been created [13]. In this article, SoaML in version 1.0.1 is used.

According to Gebhart [14], in SoaML a service interface is described by a stereotyped UML Class that realizes a UML Interface describing the provided operations. A second UML Interface can be used for specifying callback operations the service consumer has to provide. These are necessary to realize asynchronous operation calls as they are for example required to invoke long-running business processes [7].

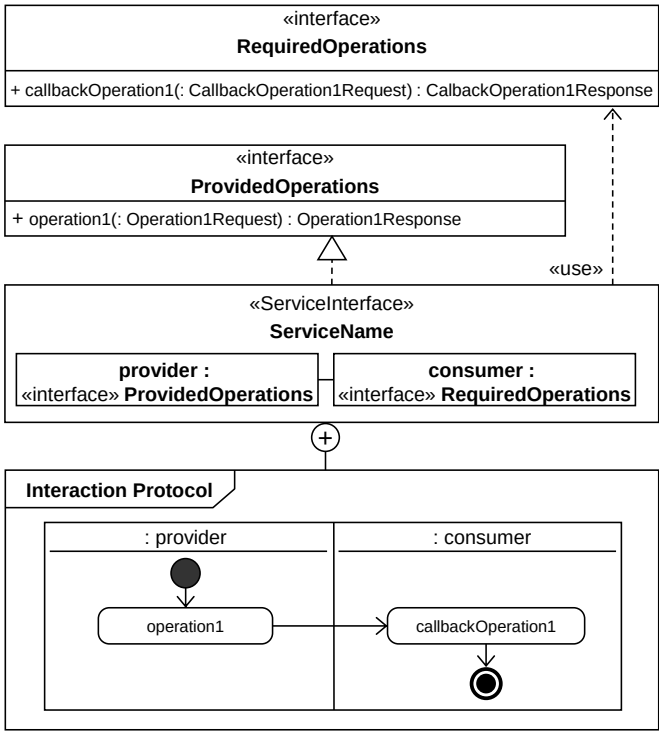


Figure 1. Service interface in SoaML.

An interaction protocol can be added as owned behavior. It is described by means of a UML Activity and determines the valid order of the operation calls. Every call is modeled using a UML Call Operation Action and is assigned to a UML Partition that represents one of the participating roles. Figure 1 shows a service interface in SoaML. In this case, the service interface describes that an operation is provided, namely the operation "operation1". There is one request message expected as input parameter. A response message will be returned as a result of the operation call. Furthermore, one callback operation is expected to be provided by the service consumer. In this case according messages are also included. The service provider is named "provider" and the service consumer is named "consumer". The interaction protocol describes that for a valid result the provided operation has be to be called initially on the part of the provider. Afterwards, the callback operation will be invoked. The messages used as input and output parameters are modeled using UML Classes stereotyped by "MessageType". They can be further refined into more fine-grained data types. Figure 2 shows the modeling of message types.

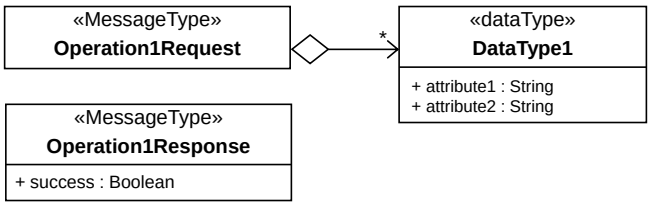


Figure 2. Message types in SoaML.

The service component is represented by a UML Component stereotyped by "Participant". Ports with Service or Request stereotype constitute the access points to the provided or required functionality and are typed by a certain service interface. An Activity as an owned behavior and visualized as UML activity diagram enables the specification of the internal logic.

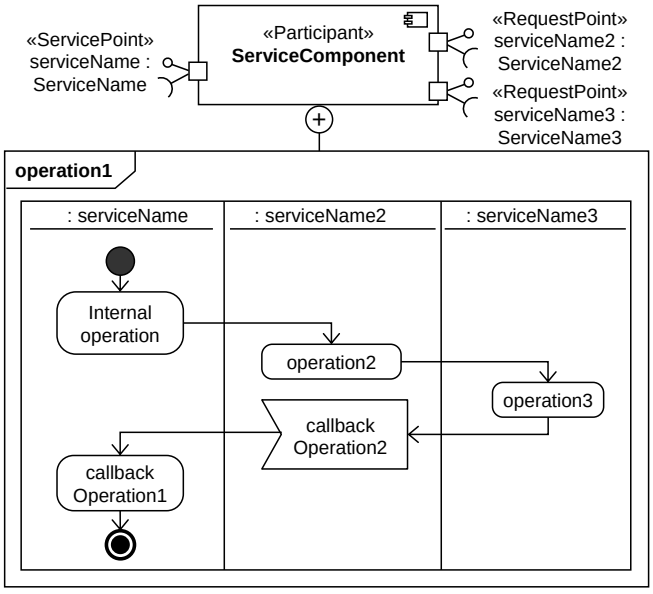


Figure 3. Service component in SoaML.

Figure 3 shows a service component in SoaML. It provides one service specified by the service interface depicted before. In order to fulfill the functionality of the service component, two other services are required. According to the Activity owned by the component, in a first step, an internal operation is performed. This means that this functionality is completely fulfilled by the service component itself. Thus it is modeled using a UML Opaque Action. Afterwards, the operations "operation2" and "operation3", provided by the services "serviceName2" and "serviceName3" respectively, are invoked. These operation calls are modeled using UML Call Operation Actions within according UML Partitions. Next, the service component waits for "callbackOperation2" being invoked. Finally, it invokes the callback operation provided by the initial service consumer.

B. Mapping Rules

In the context of mapping formalized service designs onto web service implementation artifacts based on XSD, WSDL, SCA, and BPEL approaches exist that consider either the derivation from SoaML-based models, UML models with own applied UML profiles, or standard UML models.

For the generation of XSD, IBM [15] and Sparx Systems [16] provide adequate mapping rules that map UML class diagrams onto XSD artifacts and support both the transformation of classes and their relationships like aggregations, compositions, associations, and generalization. Both vendors integrate the mapping rules into their own tools, which enable a model-driven development with a graphical tool support. The transformations are applicable to all UML models without any constraints. The applied rules can be used in our approach to map message types of service designs onto XSD.

Regarding WSDL, Grønmo et al. [17] discuss the advantages and disadvantages between using WSDL-independent and WSDL-dependent models. Their conclusion is that WSDL-dependent models, which are UML models containing WSDL-specific constructs, obscure the behavior and content of modeled services and make service designs incomprehensible. WSDL-independent models in contract simplify building complex web services and integrating existing web services. For that reason, they provide transformations based on UML class diagrams with custom WSDL-independent stereotypes. However, most of the presented transformations are based on standard UML elements and are thus applicable for service designs based on SoaML as it abstracts from WSDL details too. Also IBM [18] introduces mapping rules and an automatic transformation from UML to WSDL in [8]. These rules fully cover the transformation of standard UML elements into WSDL but are not described in detail. Only the relationships between source and target elements can be inferred and used in our work. In contradistinction to the previous related work the transformation generates also needed namespaces not bound to the source models but bound to the project structure used during the transformation. The project structure has the form of a file system containing source models and the relative paths will be used in order to generate namespaces for the target artifacts. This strategy may generate correct namespaces for a simple project. However, when merging the generated artifacts from many projects or changing the project structure during development the resulting namespace changes will make the WSDL files ambiguous.

Hahn et al. [19] present a transformation from a Platform Independent Model (PIM) to a Platform Specific Model (PSM), which converts SoaML to BPEL, WSDL, and XSD artifacts. Compared to our approach requiring a generation of BPEL processes from UML activity diagrams, the authors use BPMN processes as source models for the generation of executable BPEL processes. Even though no detailed mapping rules are provided, a promising and consistent output is generated and the mapping is illustrated using a simple scenario. The approach can be considered as a proof for the possibility of producing web service artifacts from SoaML service designs. The authors restrict that a SoaML service interface is mapped onto one and only one WSDL document containing XSD types that represent the SoaML Messages. A new capability supported by the SoaML to WSDL transformation is the ability to generate Semantic Annotations for WSDL (SAWSDL).

For generating BPEL, Mayer et al. [20] discuss the difficulties when transforming a UML Activity illustrated by means of a UML activity diagram into an executable language, such as BPEL. They introduce two alternatives on generating BPEL constructs. The first alternative is to generate a BPEL process similar to the UML Activity, where control nodes of the UML are replaced with edge and activity guards. The second alternative is to create a BPEL process with constructs in UML converted to their equivalent BPEL constructs. The first alternative is easy to be implemented and results in an unreadable and complex BPEL process, whereas the second one results in a better structured orchestration. The approach presents a robust and promising transformation into BPEL. However, the WSDL artifacts are inferred from elements described by a custom UML profile. Further mapping rules to transform workflows modeled using UML Activity elements onto BPEL artifacts are presented by IBM [21]. The approach handles some constraints of a UML Activity and provides adequate solutions. For example, to specify needed information, as for instance the partner links, the activity diagram should be extended with UML elements, such as input and output pins. Another constraint handled by the authors is how to model loop nodes in an Activity. Here, the authors propose a specific representation in UML to enable an easy and consistent generation of a BPEL loop element. These enhancements among others can be applied to consistently transform a UML Activity as the internal behavior of service components into an executable BPEL process.

SCA is a software technology which provides a model for building and composing applications and systems applying a service-oriented architecture paradigm. Combined with other technologies, such as WSDL and BPEL, SCA provides the underlying component model. In [18], Digre provides mapping rules for SoaML elements and SCA. The transformation is executed manually and the author mentions

that ambiguities in the SoaML model may prevent from producing proper SCA models. This is exactly the reason why a certain self-contained and well-understood design artifact, such as the service design in this article, has to be chosen when describing transformations. Another fully automated and tool-supported mapping of SoaML onto SCA artifacts is proposed by IBM [22]. The tool allows the application of SCA stereotypes to the source models in order to add more details specific to the SCA domain.

III. SCENARIO

In order to illustrate the transformation of service designs based on SoaML into web service implementation artifacts, the scenario of a workshop organization at a university and the involved systems are introduced in this section. The system helps visitors and members of the university in organizing a meeting or a workshop at a room located at the university campus. Additionally, the development steps for creating the required service designs are explained.

A. Business Requirements

In a first step, the business requirements have to be formalized. For that purpose, a domain model, business use cases, and the business processes that are expected to be supported by IT have to be described. These artifacts constitute the basis to create service designs based on SoaML that can be used to derive web service implementation artifacts.

The domain model describes entities and their relation within the considered domain. It is necessary to understand the domain, to unify the terminology, and to avoid misunderstandings. Thus, terms used within business use cases and business processes are expected to follow the domain model. Furthermore, operations and parameters are expected to be named functionally when designing services [1]. This can be only determined when functional terms, such as entities, are documented. To formalize the domain model, there exist several approaches. One alternative is to use UML class diagrams.

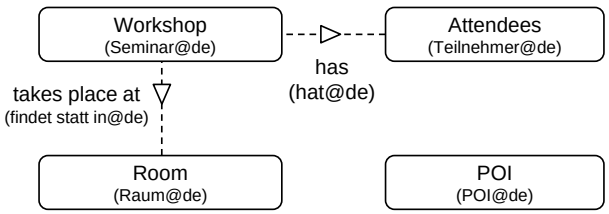


Figure 5. Domain model for workshop organization scenario.

Another alternative that has been chosen in this article is the Web Ontology Language (OWL) [23][24]. One advantage of OWL is that it can be directly referenced by WSDL using the Semantic Annotations language for WSDL (SAWSDL) [25]. By means of labels, OWL allows the description of terms in various languages. This is especially helpful when different languages are used during the requirements, the design, and implementation phases. In this case, the domain model includes the terms in English and German. An excerpt of the domain model for the workshop organization scenario is depicted in Figure 5. The domain model can either be formalized directly using XML or by means of tools, such as Protégé [24].

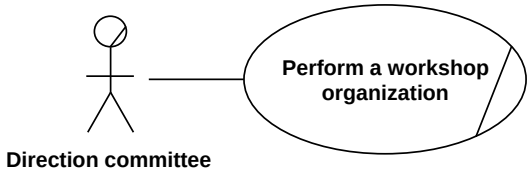


Figure 6. Business use case expected to be supported by IT.

The business use cases are modeled using UML use cases extended by the UML profile for business modeling as introduced by Johnston in [26]. The business use case expected to be supported by IT is illustrated in Figure 6.

Compared to standard UML use cases, a business use case describes the business boundaries instead of system

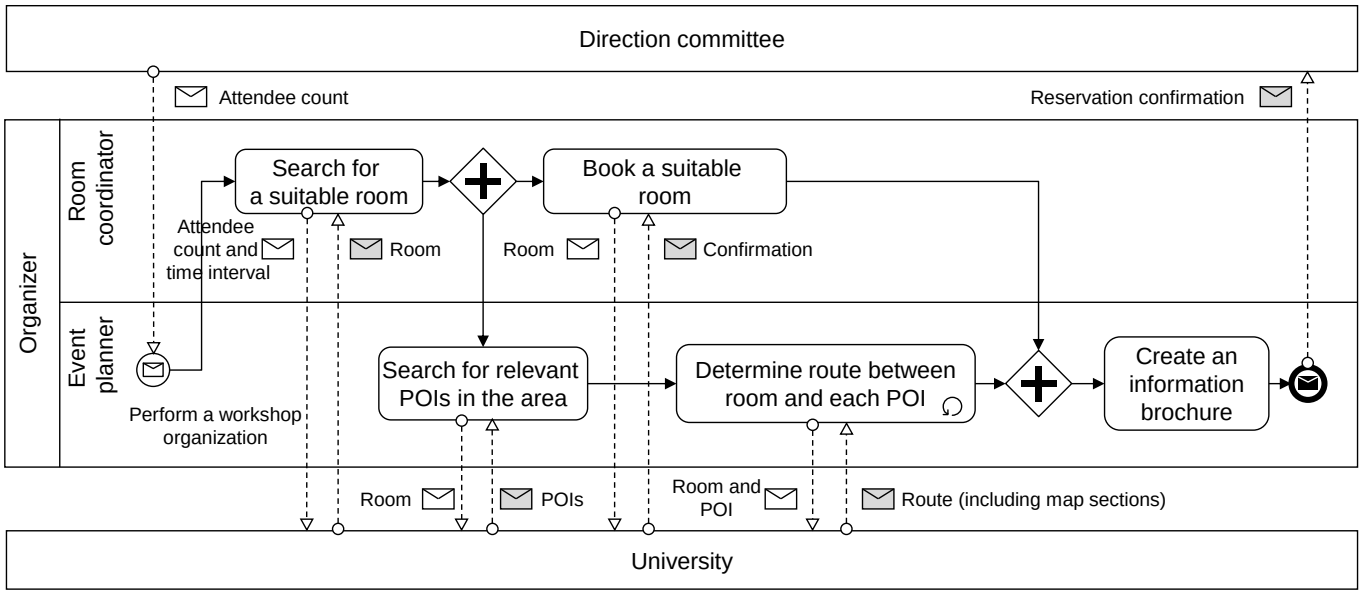


Figure 4. Business process of the workshop organization scenario.

boundaries. This means that the business actor specified by the stereotyped UML Actor is not part of the business represented by the business use case. The business actor, i.e., is an external participant that interacts with the business realizing the business use case. According to the diagram a direction committee is expected to be supported while performing a workshop organization.

A business use case is realized by means of a business process. The business process, i.e., describes the internal behavior of a business use case. Since the business process represents the essential artifact when deriving service designs, the business process for the workshop organization business use case has to be described. For that purpose the Business Process Model and Notation (BPMN) [27] is applied. The process for the considered scenario is illustrated in Figure 4. Two existing systems, provided by the university, are involved in the realization of the business process, namely the KITCampusGuide system and the facility management system. The KITCampusGuide system provides operations to manage Points of Interest (POI) such as the determination of all relevant POIs (Parking, Cafeteria, etc.) in the area surrounding the target and the provision of route guidance to all relevant POIs. The facility management system is concerned with room searches and enables the reservation of a room for a given number of attendees at the desired time interval.

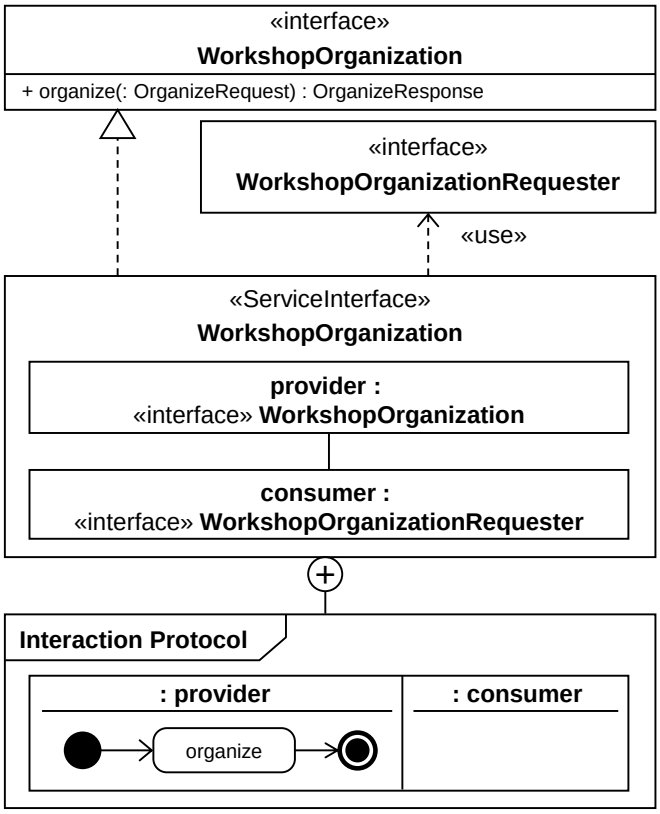


Figure 7. Derived service interface.

B. Service Designs

In the second phase of the development process, the service design phase, a set of service designs have to be designed and modeled using SoaML. Each service design is built according to the understanding introduced in Section II. The service designs can be created systematically as introduced by Gebhart et al. in [28]. In a first step, the service designs are derived from the business requirements. For example, for every pool within the business process, one service interface and one service component is created. All message interactions are used to derive provided and required operations. For example, a message start event in BPMN is transformed into one provided operation. In a next step, the derived service designs are revised regarding quality attributes, such as loose coupling and autonomy, as introduced in [29]. This is important, because these quality attributes influence higher-value ones, such as flexibility and maintainability, which in turn represent essential drivers for service-oriented architectures. For example, in this step naming conventions are considered, operations within service interfaces are split or merged, and it is ensured that long-running operations are provided by means of asynchronous instead of synchronous operations.

The resulting artifact from the design phase which describes the service "WorkshopOrganization" are presented below. This represents the business process and realizes the orchestration of involved services. Figure 7 shows the designed service interface. The UML Interface realized by the ServiceInterface element lists the provided operation "organize" with its input and output parameters. The input and output parameters are defined using the message types "OrganizeRequest" and "OrganizeResponse" described in Figure 9. As the interface associated by means of the usage dependency does not contain any operation, the service consumer does not have to provide callback operations. This corresponds to the interaction protocol. This example also shows the consideration of quality attributes. The operation "organize" represents the "Perform a workshop organization" message start event within the business process modeled in BPMN. However, the quality attribute discoverability describes that operations should be functionally named and should follow naming conventions. Thus, after a systematic derivation of the service interface, the operation is renamed from "Perform a workshop organization" to "organize".

In addition, a service component, representing the component that fulfills the functionality, is specified for this service. The service component and its internal behavior are illustrated in Figure 8. Also in this case, a systematic derivation is first performed. For example, every invoke activity within the business process in BPMN is transformed into a UML Call Operation Action that is assigned to a UML Partition representing a certain system. Flow elements are transformed into equivalents UML constructs. In a next step, quality attributes are considered, i.e., regarding the discoverability, naming conventions are considered and the functional naming is ensured.

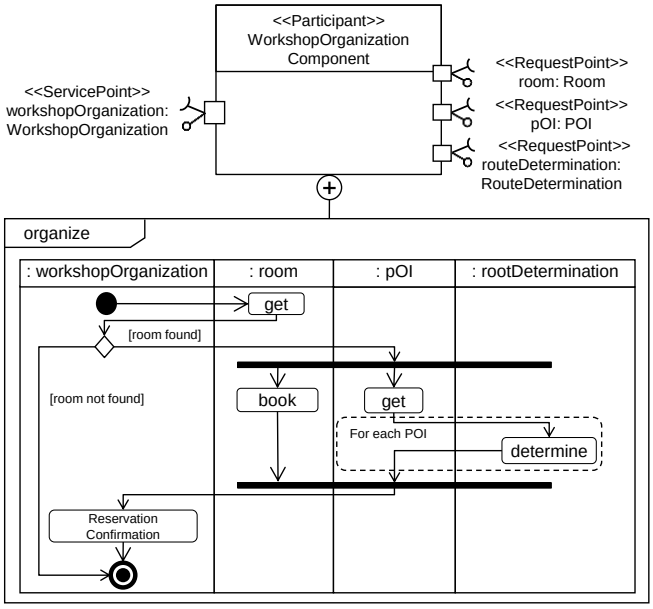


Figure 8. Derived service component.

IV. Derivation of Web Service Implementation Artifacts from Service Designs Based on SoaML

In this section, the steps necessary to derive web service implementation artifacts from service design are illustrated. Divided into four parts, the first subsection targets the derivation of data types and their definitions using XSD. For the provided and required interfaces of the service interface, service interface descriptions based on WSDL with associated message types are generated. For realizing the orchestration of services, BPEL is derived from UML Activity elements and added as the owned behavior of the service component. Finally, a SCA component model describing the structure of the application is derived from the service component. For each step and for each transformation performed existing mapping rules are applied.

A. Derivation of Data Types

Data types contained within the SoaML service designs are expected to be mapped onto XSD to describe request and response messages used within WSDL operations.

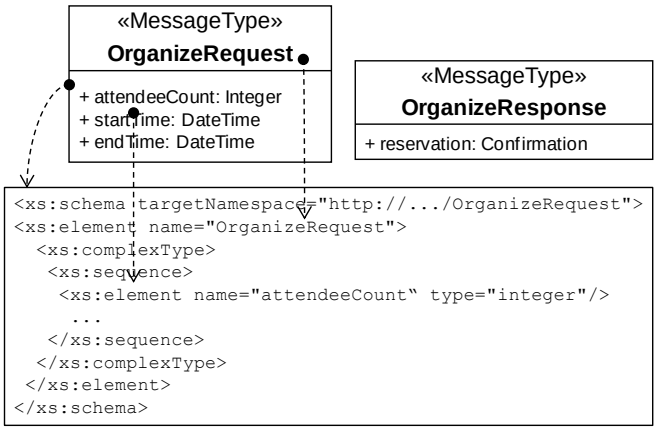


Figure 9. Derived XML Schema Definitions from SoaML messages.

The service interface in Figure 7 provides the operation "organize", which contains input and output messages in the form of UML DataTypes stereotyped by MessageType. They constitute containers for further data types described using attributes or UML Associations to other UML Classes. We follow the mapping rules provided by Sparx Systems [16]. Each input and output parameter is mapped onto an element with a complexType and a sequence of XML elements defining the attributes of the messages as demonstrated in Source Code 1. The XSD descriptions are stored in separate files in order to allow other WSDL documents to reuse the data types.

The separated XSD files are then imported into the WSDL document using an import statement with the corresponding namespace and schema location as shown in Source Code 1.

```
<wsdl:types>
 <xs:import namespace="http://.../OrganizeRequest"
   schemaLocation="http://.../organize.xsd"/>
</wsdl:types>

<wsdl:message name="OrganizeRequestMessage">
 <wsdl:part name="body" element="OrganizeRequest"/>
</wsdl:message>
```

Source Code 1. Derived WSDL message types.

Table I summarizes the transformation, provides more details about the mapping rules, and lists the source and the target elements with necessary attribute configurations.

TABLE I. SoaML artifacts to XML Schema Definition

SoaML Artifact	XML Schema Definition
Package	A schema element with the "targetNamespace" attribute to identify and reference the XSD is generated.
Class (MessageType)	An element as a root element and a complexType definition containing a sequence of child elements are generated. The "name" attribute corresponds to the name of the class.
Attributes (ownedAttributes)	Is mapped onto an element with the "name" and "type" attributes set to the same as in the source.
PrimitiveType, Datatype and MessageType	Are mapped onto the "type" attribute of an element generated while mapping the member attributes of a class. For each referenced data type an import element is used to add the corresponding external schema.
Association	An element is declared for each association owned by a class. The "name" attribute is set to the one of the association role. The "minOccurs" and "maxOccurs" reflect the cardinality of the association.
Generalization (Inheritance)	An extension element is generated for a single inheritance with the "base" attribute set to the base class name. The UML Attributes of the child class are then appended to an "all" group within the extension element.

B. Derivation of Service Interfaces

After generating data types, the operation definitions and their parameters can be derived from the SoaML service interface and its realized interface.

According to IBM [18], a port type acting as container for the operations is generated and each parameter is mapped onto a part element as shown in Source Code 1. The name of the port type is derived from the name of the realized interface in the SoaML service design and enhanced with the suffix "PortType". The WSDL operation element includes the attribute "name", which corresponds to the operation name within the service design. Additionally, the previously derived input and output messages are associated. In case of service inheritance the operations of the parent interface are copied to the same generated port type as stated by Hahn et al. [19]. This allows overcoming the not supported WSDL inheritance limitation.

```
<wsdl:portType name="WorkshopOrganizationPortType">
 <wsdl:operation name="organize">
  <wsdl:input message="OrganizeRequestMessage"/>
  <wsdl:output message="OrganizeResponseMessage"/>
 </wsdl:operation>
</wsdl:portType>
```

Source Code 2. Derived port type in WSDL.

Till now, the abstract part of a WSDL was generated. The concrete part encompasses deployment-specific details about how and where to access a service. A binding definition specifying the communication technology that can be used by the consumer is generated. The binding is named as a combination of the interface name and the suffix "SOAP". Additionally, it is associated with the prior defined port type by setting the attribute "type" to the name of the interface including the suffix "PortType". The messaging protocol binding and the transport protocol binding are set to Simple Object Access Protocol (SOAP) and Hypertext Transfer Protocol (HTTP). In this work, we use SOAP as a default protocol. The final part focuses on the physical endpoint of the service. The endpoint is specified by a URL that has to be specified by the developer.

```
<wsdl:binding name="WorkshopOrganizationSOAP"
              type="WorkshopOrganizationPortType">
 <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="organize"/>
</wsdl:binding>

<wsdl:service name="WorkshopOrganization">
 <wsdl:port binding="tns:WorkshopOrganizationSOAP"
            name="WorkshopOrganizationSOAP">
  <soap:address location="<server>:<port>"/>
 </wsdl:port>
</wsdl:service>
```

Source Code 3: Derived binding and service definition.

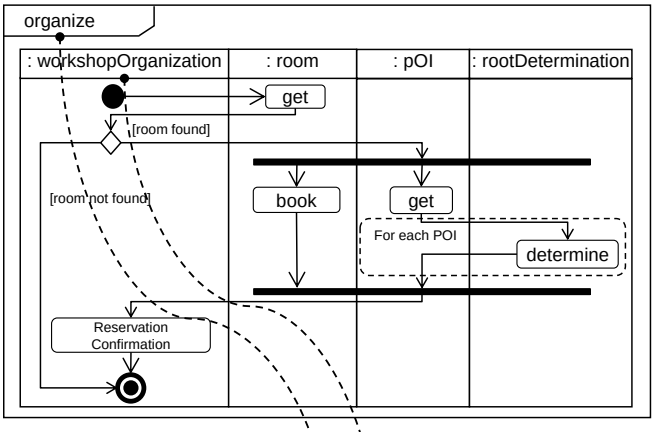
TABLE II. SOAML ARTIFACTS TO WSDL

SoaML Artifact	WSDL
Interface realized by a ServiceInterface	WSDL PortType that will be named according to the interface. It represents provided operations.
Interfaces used by a ServiceInterface	WSDL PortType that will be named according to the interface. It represents callback operations.
Input / Result / Exception parameters in a service interface	WSDL Messages that can be used within the operations.
Parameters	Message Parts that reference the WSDL Messages.
Parameter types	Types, they will be defined in a separate *.xsd document

C. Derivation of Executable Business Logic

The mapping rules provided by IBM [21] cover all UML artifacts of a UML Activity involved in the derivation of control flow elements of a BPEL process. Additionally, new mapping rules to set attribute values were identified in this article and are also mentioned in the following transformation description.

The UML activity diagram in Figure 8 describes the internal behavior of a service operation "organize" and is considered to demonstrate the transformation for most often used control flow elements of a UML activity diagram. The first generated fragment for the BPEL process is the main scope. It exists only once and consists of a sequence of other activities.




```
<bpel:process>
…
<bpel:sequence name="main">
      <bpel:receive name="organize" operation="organize"
       partnerLink="workshopOrganization"
       portType="tns:WorkshopOrganization"
       createInstance="yes" variable="input"/>
…
</bpel:sequence>
…
</bpel:process>
```

Figure 10. Derivation of main scope.

The first partition in the activity diagram contains an initial node which is mapped onto a receive activity with the attribute "partnerLink" set to the label of the partition, namely "workshopOrganization". The attribute "operation" corresponds to the operation name in the interaction protocol. This activity is located at the top of the main scope and waits for an arriving message. The derivation is shown in Figure 10. The mapping rules are not summarized within a table, as according tables are directly available in [21].

The involved web services are specified by separate WSDL definitions containing partnerLink definitions. In order to call these web services, the BPEL process sets a partnerLink for each invoke activity. The partnerLinks are derived from the label of the partitions, such as "room" or "rootDetermination".

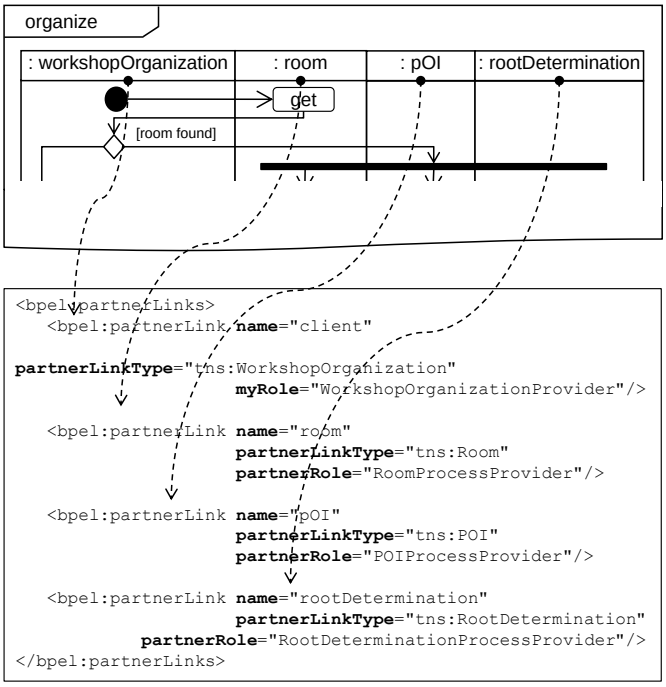


Figure 11. Derivation of partner links.

The partition containing the initial node is mapped onto a partnerLink definition with the attribute "name" set to the value "client" representing the BPEL process itself. For the other partitions, the attribute "name" is equal to the label of the respective partition. Moreover, the partnerLink defining the process itself has the attribute "myRole" whereas other partnerLinks have an attribute "partnerRole" representing the role of an invoked web service. Figure 11 shows the derived partnerLinks for the considered service operation and the invoked service "Room". After defining the partnerLinks, which belong to the abstract part of a BPEL process, the actions within the partitions are mapped onto invoke activities as illustrated in Figure 12.

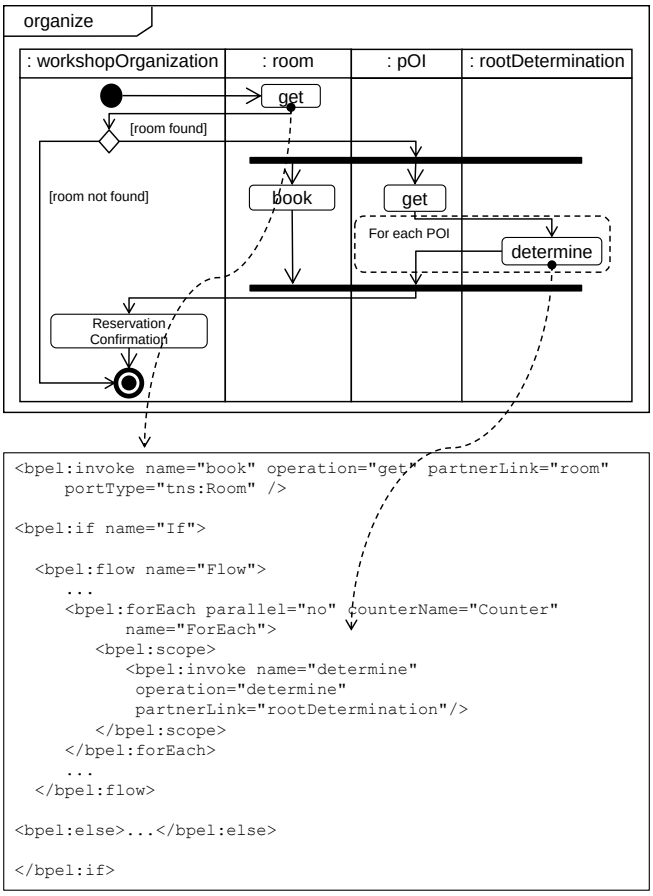


Figure 12. Derivation of activities.

Each activity has the attributes "name" and "operation" set to the name of the action. The attribute "partnerLink" is set to the corresponding partnerLink defined earlier. The activities are located within the corresponding scopes of flow elements mapped later. Compared to the other activities, the action "ReservationConfirmation" in the first partition is an opaque action executed by the BPEL process itself and thus is not mapped onto an invoke activity. After a skeleton for the BPEL process has been created, the control flow elements are derived from corresponding UML elements. The decision node is mapped onto a BPEL if-else construct. The condition of the node has to be added manually by the developer. The black bar representing a fork node and a parallel execution of the contained action is mapped onto a BPEL flow construct. The black bar representing a join node with incoming arrows is implicitly included in the earlier derived BPEL flow construct. The loop node is illustrated using a dashed area and is mapped onto a forEach construct with the attribute "parallel" set to the value "no". If the loop node in UML contains a fork and a join node, the attribute "parallel" is set to "yes". The derivation of flow elements is depicted in Figure 13.

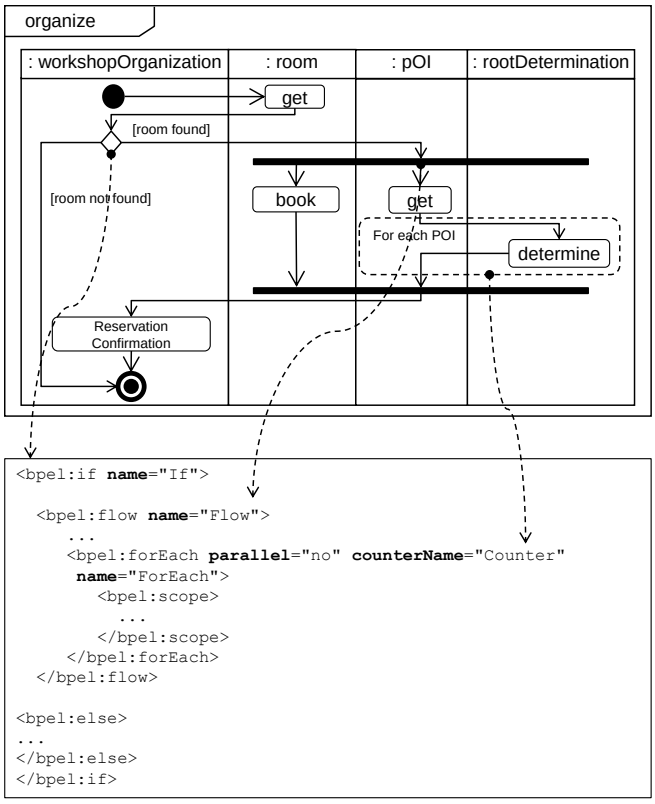


Figure 13. Derivation of flow elements.

D. Derivation of Component Models

In order to embed the already generated artifacts into an entire component model, SCA elements are derived from the service designs. Figure 14 illustrates the mapping between service components described by SoaML Participants and SCA elements, such as SCA Composites, Components, Services, References, and Wires, using mapping rules provided by Digre et al. in [18].

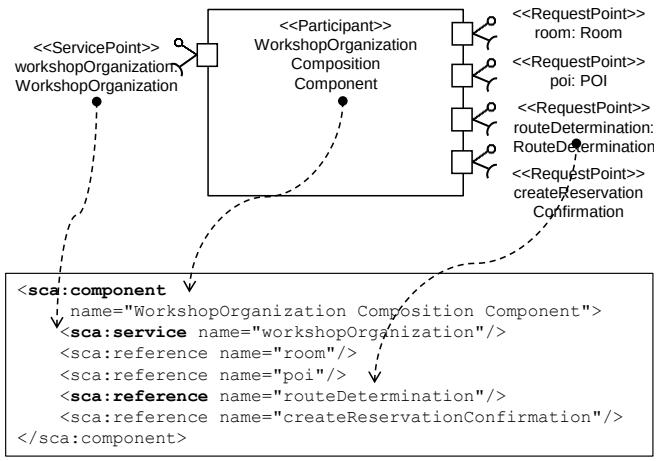


Figure 14. Derivation of SCA component model.

Regard naming conventions, each Participant is mapped onto a SCA component with name set to the label of the Participant. Since each SoaML Participant contains Services and Requests representing provided and required services, SCA Services and SCA References are generated. The names of these elements are set to the names of the ports within the SoaML Participant. The derivation is shown in Figure 14.

The SCA Composite is the basic unit of a composition in a SCA Domain and is an assembly of SCA Components, Services, References, and Wires. The service component presented earlier deals with the orchestration of external services and contains also a reference to an internal component for creating the reservation confirmation. These two components are to be grouped into an SCA Composite, whereas SoaML service channels wiring the Services to Requests are mapped onto SCA Wire elements. Additionally, if two Services or two Requests are wired together to delegate service calls, a promote element is added. Figure 15 illustrates the final SCA Composite in a graphical visualization as introduced by the standard.

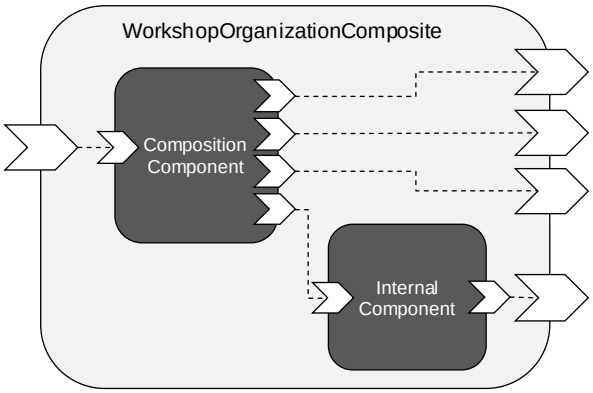


Figure 15. SCA Composite for the workshop organization process.

SCA requires that Service and Reference elements are compatible. The compatibility is assured by means of the assigned interfaces. The interfaces used in this context can be derived from service interfaces in SoaML as illustrated in Section B. The resulting service interface descriptions based on WSDL can be embedded into the SCA Composite. For this purpose, based on the realized and used UML Interfaces representing provided and required interfaces within the service designs, a bidirectional service interface description using WSDL with a base and a callback interface is generated. An “interface.wsdl” element is added to the Service element with the attribute “interface” set to the URL of the WSDL service representing the provided service interface “WorkshopOrganization”. The “callbackInterface” attribute of the Service element is set to the port type representing the “WorkshopOrganizationRequester”. For the corresponding SCA Reference, the assignment is reversed, i.e., the attribute “interface” of the interface element within the SCA Reference is set to the required interface and the attribute “callbackInterface” is set to the provided interface. The systematical derivation is depicted in the following figure.

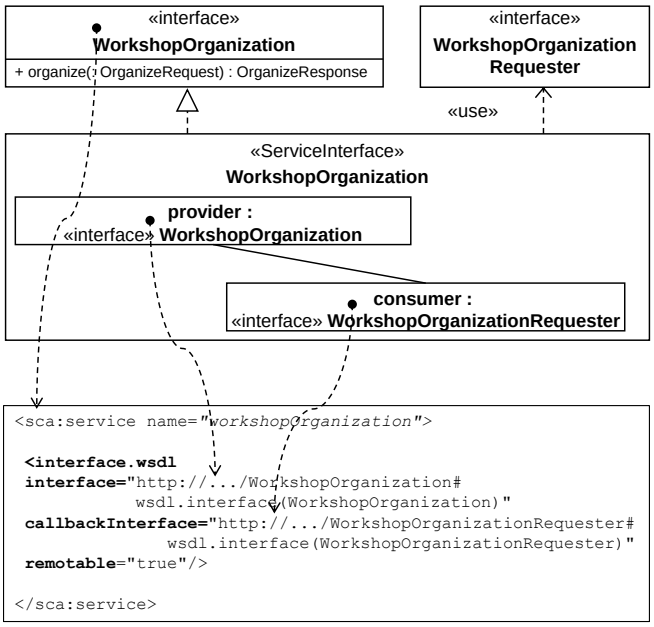


Figure 16. Integration of WSDL into SCA.

When a service component in SoaML consists of further service components, these refinements are also transformed into equivalents in SCA. Figure 17 illustrates the mapping of composite service components.

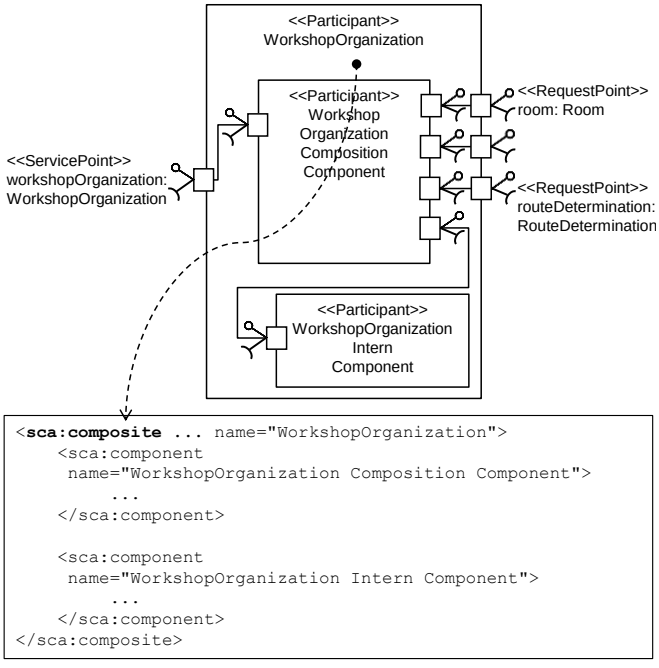


Figure 17. Transformation of composite service components.

As a result, the entire service components including their implementation and refinements into further service components can be mapped into SCA. The following table summarizes the mapping rules.

TABLE III. SOAML ARTIFACTS TO WSDL

SoaML Artifact	SCA
Service Component / Participant	Composite that is named according to the Participant.
Service	Service that is named according to the Service in SoaML. As interface in SCA the mapped service interface the Service is typed by is referenced.
Request	Reference that is named according to the Request in SoaML. Also in this case the Reference has an interface that is derived by the service interface the Request is typed by.
ServiceInterface	The service interface a Service or Request is typed by is transformed into a WSDL according to the rules described before. The service interface is transformed into a interface in SCA that is used to describe Service and References.
OwnedBehavior	An owned behavior that can be transformed into a BPEL process is set as implementation for a certain component in SCA.
Internal Participant	Component within the SCA composite. The component is named according to the internal participant.
Service Channels	Wiring between component within the SCA composite.

V. CONCLUSION AND OUTLOOK

In this article, we illustrated the derivation of web service implementation artifacts from prior created service designs that base on SoaML as standardized modeling language. For that purpose, existing mapping rules that in particular focus on UML as source artifacts have been analyzed and enriched with details that aim at supporting service design specifics. As a result, the mapping rules could be identified to enable the systematic derivation of web services based on XSD, WSDL, BPEL, and SCA as wide-spread technologies.

This systematic derivation is especially necessary in model-driven development approaches for web services. As SoaML is a language standardized by the OMG it is the preferred language when modeling services. Due to the complexity of today's software, a detailed planning and thus modeling before implementation is recommended. During the design phase it is easier to focus on architecture-relevant issues, such as a loose coupling between services. The mapping rules enable a systematic derivation of web services so that prior considered quality attributes are also fulfilled by the implementation artifacts.

The created mapping rules have been exemplified by means of a service-oriented workshop organization system. The system has been created using a model-driven approach. After capturing the requirements, the service designs have been created and aligned with wide-spread quality attributes as introduced by Gebhart et al. [7][29] using the QA82 Analyzer [32]. The methodology has been described in [33]. Afterwards, the service designs have been used to derive web services using the extended mapping rules.

The rules on the one hand help IT architects to understand the relation between service designs, the language SoaML, and web services as implementation, which allows IT architects to reduce the impact of service design changes on the final implementation. On the other hand, the mapping rules constitute the conceptual basis for automatic transformation as they can be realized using languages, such as Query Views Transformation (QVT) [34]. This will increase the significance of SoaML in model-driven development processes as it represents a full-fledged development artifact.

In the past, we especially focused on the creation of quality attributes and metrics for service designs based on SoaML. Also our tool, the QA82 Analyzer that enables automatic quality analyses aimed at the analyses of SoaML models. The conceptual understanding about how characteristics of service designs are reflected within web service implementations enables us to transform our existing SoaML metrics into metrics for web services. Thus, in the future our QA82 Analyzer will also be able to analyze web services regarding wide-spread quality attributes, such as loose coupling and autonomy.

REFERENCES

[1] M. Gebhart and J. Bouras, "Mapping between service designs based on soaml and web service implementation artifacts", Seventh International Conference on Software Engineering Advances (ICSEA 2012), Lisbon, Portugal, November 2012, pp. 260-266.

[2] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA – Service-Oriented Architecture Best Practices, 2005. ISBN 0-13-146575-9.

[3] T. Erl, SOA – Principles of Service Design, Prentice Hall, 2008. ISBN 978-0-13-234482-1.

[4] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, Addison-Wesley, 2003. ISBN 978-0321154958.

[5] P. Hoyer, M. Gebhart, I. Pansa, S. Link, A. Dikanski, and S. Abeck, "A model-driven development approach for service-oriented integration scenarios", 2009.

[6] M. Gebhart and S. Abeck, "Quality-oriented design of services", International Journal on Advances in Software, 4(1&2), 2011, pp. 144-157.

[7] M. Gebhart, M. Baumgartner, S. Oehlert, M. Blersch, and S. Abeck, "Evaluation of service designs based on soaml", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 7-13.

[8] M. Gebhart, M. Baumgartner, and S. Abeck, "Supporting service design decisions", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 76-81.

[9] T. Erl, Service-Oriented Architecture – Concepts, Technology, and Design, Pearson Education, 2006. ISBN 0-13-185858-0.

[10] SENSORIA, "D1.4a: UML for Service-Oriented Systems", http://www.sensoria-ist.eu/, 2006. [accessed: July 11, 2012]

[11] S. Johnston, "UML 2.0 profile for software services", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/05/419_soa/, 2005. [accessed: July 11, 2012]

[12] OMG, "Unified modeling language (UML), superstructure", Version 2.2, 2009.

[13] OMG, "Service oriented architecture modeling language (SoaML) – specification for the uml profile and metamodel for services (UPMS)", Version 1.0, 2012.

[14] M. Gebhart, "Service Identification and Specification with SoaML", in Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments, Vol. I, A. D. Ionita, M. Litoiu, and G. Lewis, Eds. 2012. IGI Global. ISBN 978-1-46662488-7.

[15] IBM, Generating XSD Schemas from UML Models, Rational Systems Developer Information Center. http://publib.boulder.ibm.com/infocenter/rsdvhelp/v6r0m1/index.jsp. [accessed: July 11, 2012]

[16] Sparx Systems, XML Schema Generation, http://www.sparxsystems.com.au/resources/xml_schema_generation.html, 2011. [accessed: July 11, 2012]

[17] Roy Grønmo, David Skogan, Ida Solheim and Jon Oldevik, Model-driven Web Services Development, SINTEF Telecom and Informatics, 2004.

[18] IBM, Transforming UML models into WSDL documents, Rational Software Architect. http://publib.boulder.ibm.com/infocenter/rsahelp/v7r0m0/index.jsp. [accessed: July 11, 2012]

[19] Christian Hahn, David Cerri, Dima Panfilenko, Gorka Benguria, Andrey Sadovykh and Cyril Carrez, Model transformations and deployment, SHAPE 2010.

[20] Philip Mayer, Andreas Schroeder and Nora Koch, MDD4SOA Model-Driven Service Orchestration, 2008.

[21] IBM: Transforming UML models to BPEL artifacts, Rational Software Architect. http://publib.boulder.ibm.com/infocenter/rsahelp/v7r0m0/index.jsp, 2010. [accessed: July 11, 2012]

[22] IBM, Transforming UML models to Service Component Architecture artifacts, Rational Software Architect. http://publib.boulder.ibm.com/infocenter/rsahelp/v7r0m0/index.jsp. [accessed: July 11, 2012]

[23] W3C, "OWL 2 web ontology language (OWL)", W3C Recommendation, 2009.

[24] M. Horridge, "A practical guide to building owl ontologies using protégé 4 and co-ode tools", http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/, Version 1.2, 2009. [accessed: January 04, 2011]

[25] W3C, "Semantic Annotations for WSDL and XML Schema (SAWSDL)", W3C Recommendation, 2007.

[26] S. Johnston, "Rational uml profile for business modeling", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/5167.html, 2004. [accessed: March 04, 2013]

[27] OMG, "Business process model and notation (BPMN)", Version 2.0 Beta 1, 2009.

[28] M. Gebhart, S. Sejdovic, and S. Abeck, "Case study for a quality-oriented service design process", Sixth International Conference on Software Engineering Advances (ICSEA 2011), Barcelona, Spain, October 2011, pp. 92-97.

[29] M. Gebhart and S. Abeck, "Metrics for evaluating service designs based on soaml", International Journal on Advances in Software, 4(1&2), 2011, pp. 61-75.

[30] Tom Digre, ModelDriven.org, http://lib.modeldriven.org/MDLibrary/trunk/Applications/ModelPro/docs/SoaML/SCA/SoaML to SCA.docx, May 2009. [accessed: July 11, 2012]

[31] J. Amsden, "Modeling with soaml, the service-oriented architecture modeling language – part 1 – service identification", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html, 2010. [accessed: July 11, 2012]

[32] Gebhart Quality Analysis (QA) 82, QA82 Analyzer, http://www.qa82.de. [accessed: July 11, 2012]

[33] M. Gebhart and S. Sejdovic, "Quality-oriented design of software services in geographical information systems", International Journal on Advances in Software, 5(3&4), 2012, pp. 293-307.

[34] OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification", Version 1.1, 2011. [accessed: July 11, 2012]