# Mapping Between Service Designs Based on SoaML and Web Service Implementation Artifacts

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*—Because of the increasing complexity of service landscapes and the requirement to fulfill quality attributes, such as loose coupling and autonomy, services have to be designed in detail before implementing them. For this purpose, the Object Management Group has standardized the Service oriented architecture Modeling Language, which enables the abstract formalization of service designs. However, existing mappings between abstract formalizations and web service implementation artifacts focus on certain modeling elements and do not consider service designs as self-contained development artifacts. In this article existing and applicable mapping rules for transforming service designs based on the Service oriented architecture Modeling Language into implementation artifacts are identified and missing rules are defined. For illustration purposes, service designs of a scenario in the context of a workshop organization system are transformed into web service implementation artifacts.**

***Keywords-service design; SoaML; implementation; mapping; transformation.***

## I. INTRODUCTION

Today, services as methodology to integrate distributed systems become increasingly important. Software vendors enhance their products with service interfaces and enterprises organize their Information Technology (IT) according to service-oriented architecture (SOA) principles [18]. As result, services constitute the building blocks of today's IT and the complexity of the service landscape increases. Additionally, due to their influence, services are required to fulfill certain quality attributes [21], such as loose coupling and autonomy [20]. These attributes have been identified as important as they influence higher-value quality attributes of the entire architecture, such as its flexibility, maintenance, and cost-efficiency. In order to ensure the fulfillment of quality attributes with simultaneous handling of the service landscape complexity, a detailed planning of the services during their development and prior to their implementation is necessary. This phase is called service design phase. As part of a quality assurance process that analyses the quality of services based on abstract formalizations, a design of services has to be derived from implementation artifacts.

In order to enable a vendor-independent formalization of service designs with a common understanding and tool support, the Object Management Group (OMG) decided to work on a standardized meta model and profile for the Unified Modeling Language (UML) that enables the modeling of service-oriented architectures and their elements, the services. The result of this effort is the Service oriented architecture Modeling Language (SoaML), which is currently released in version 1.0. Compared to UML, SoaML adds several stereotypes necessary for the specifics of service-oriented architectures, such as service interfaces and participants. Today, SoaML gains increasing tool support, even IBM decided to replace their proprietary UML profile for software services with SoaML [24].

However, using SoaML in a service development process requires a systematic mapping between formalized service designs and implementation artifacts, such as web service artifacts. Otherwise, the service designs cannot act as development artifact within a model-driven approach as introduced by Hoyer et al. [22]. Furthermore, quality analyses that base on abstract SoaML models, such as introduced by Gebhart et al. in [1][2][16], cannot be applied on already implemented services as there is no way to derive an abstraction with guaranteed correct semantic. Mapping rules that are available today mostly focus on the mapping of certain selected modeling elements, such as the transformation of UML Classes into data types represented by XML Schema Definition (XSD). However, they do not consider a service design as a whole.

In this article, existing mapping rules based on UML and SoaML that are applicable for service designs are identified and summarized. Furthermore, additional mapping rules are defined if necessary. In this context, web services are assumed as implementation using in particular the Web Service Description Language (WSDL) and XSD to describe the service interface, Service Component Architecture (SCA) as component model, and the Business Process Execution Language (BPEL) for the implementation of composed services. As result, the mapping of service designs as self-contained development artifact is described which enables the modeling of service designs using SoaML within model-driven service development and quality assurance processes.

The article is organized as follows: Section II introduces the concept of service designs and analyzes mapping rules in existing work regarding their applicability for service designs. After introducing service designs of a workshop organization system in Section III, in Section IV, these service designs are mapped onto web services. Section V concludes this article and introduces future research work.

 ISBN: 978-1-61208-230-1 

## II. RELATED WORK

This section describes the fundamental terms and existing work in the context of specifying service designs and their mapping onto implementation artifacts.

### A. Service Design

According to Gebhart et al. [15][16] and Erl [19], a service design consists of a service interface as external point of view and a service component fulfilling the functionality. In order to formalize service designs and to enable their transformation into implementation artifacts, Mayer et al. [19] introduce a UML profile for describing behavioral and structural aspects of service interactions. Similarly, within the SENSORIA project [13] a UML profile for the service interaction is specified. Also IBM [23] 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 [26] and a meta model to formalize service-oriented architectures and their services. As result, SoaML has been created [3].

According to Gebhart [4], 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. An interaction protocol can be added as owned behavior. It is described by means of a UML Activity and determines the valid order of operation calls. The service component is represented by a UML Component stereotyped by a Participant. Ports with Service or Request stereotype constitute the access points to provided or required functionality and are typed by a certain service interface. An Activity as owned behavior visualized as UML activity diagram enables the specification of the internal logic. Figure 2 and Figure 6 illustrate a service interface and a service component.

### 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 UML profiles applied, or standard UML models. This work is analyzed in order to identify applicable mapping rules to transform service designs.

For the generation of XSD, IBM [6] and Sparx Systems [7] 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 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. [9] discuss the advantages and disadvantages between using WSDL-independent and WSDL-dependent models. Their conclusion is that WSDL-dependent models 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[8] 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. [12] 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 a UML activity diagrams, the authors use a BPMN processes as a 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. [5] 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

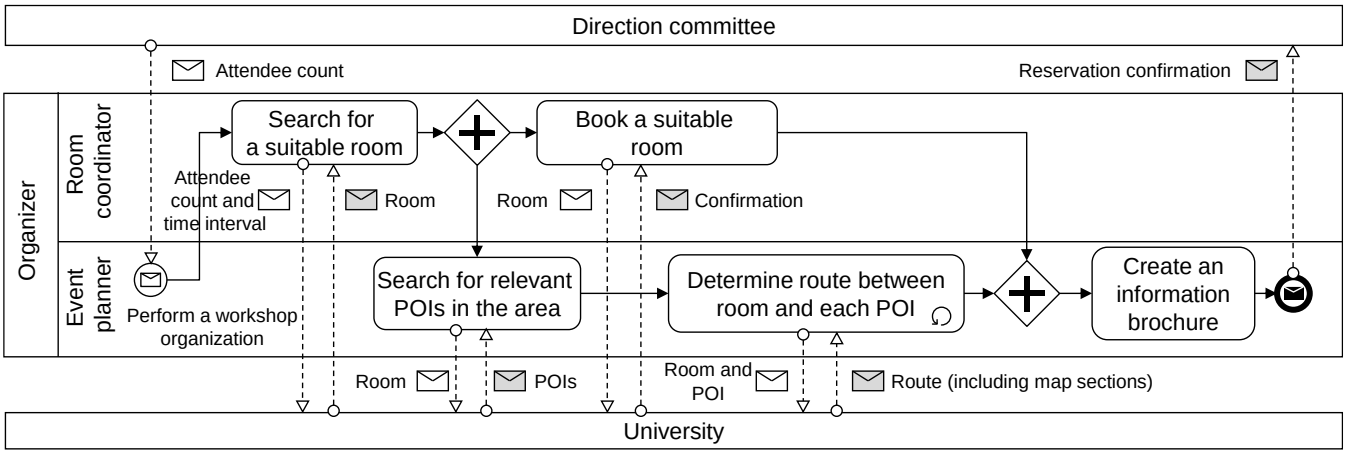


Figure 1. Business process of the workshop organization scenario.

UML profile. Further mapping rules to transform workflows modeled using UML Activity elements onto BPEL artifacts are presented by IBM [14]. 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 new technology for building 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 [10] 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 [11]. 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, in this section the scenario of a workshop organization at a university and the involved systems are introduced. 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 this purpose, beside business use-cases and the domain model as explained in [2] the business process expected to be supported by IT is described using the Business Process Model and Notation (BPMN) [25]. The process for the considered scenario is illustrated in Figure 1. The system based on this process helps visitors and members of the university in organizing a meeting or a workshop at a room located at the university campus. Two existing systems 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) and supports 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 and at the desired time interval. Both systems are provided by the university.

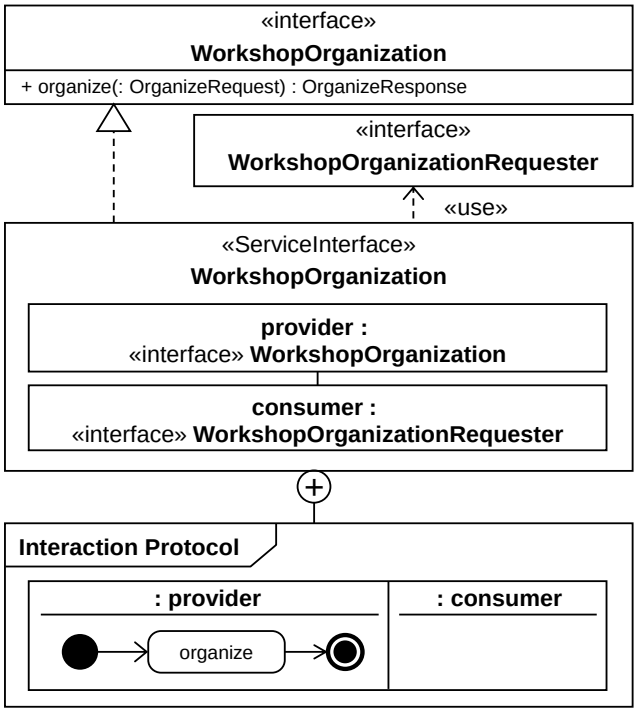


Figure 2. Designed service interface.

### B. Service Designs

In the second phase of the development process, the service design phase, a set of service designs has to be designed and modeled using SoaML. Each service design is built according to the understanding introduced in the Fundamentals section. In this article we demonstrate the approach using the "WorkshopOrganization" service that enables the orchestration of involved services. Figure 2 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 3. 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. Additionally, a service component is specified for this service representing the component that fulfills the functionality. The service component and its internal behavior are illustrated in Figure 4 and Figure 5.

## IV. Mapping between Service Designs Based on SoaML and Implementation Artifacts

In this section the steps necessary for mapping service design artifacts onto web service implementation artifacts 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 added as 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.

The service interface in Figure 2 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 [7]. 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 Figure 3. 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.

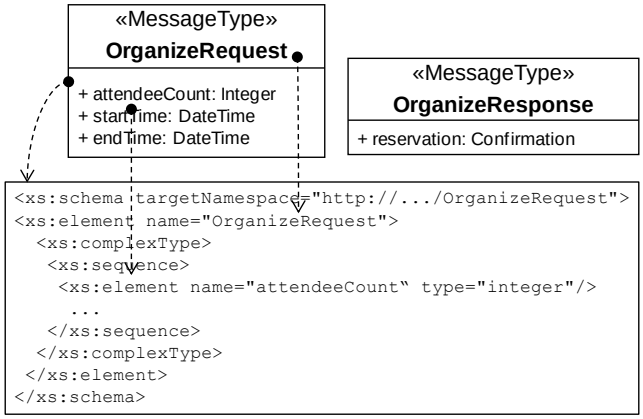


Figure 3. Derived XML Schema Definitions from SoaML messages.

```
<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.

The following table summarizes the transformation, provides more details about the mapping rules, and lists the source and the target elements with necessary attribute configurations. Due to the lack of space, the following transformations are described in textually only.

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) | An attribute 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. |

 ISBN: 978-1-61208-230-1 

### 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 [8], 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 into the same generated port type as stated by Hahn et al. [12]. This enables to overcome 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.

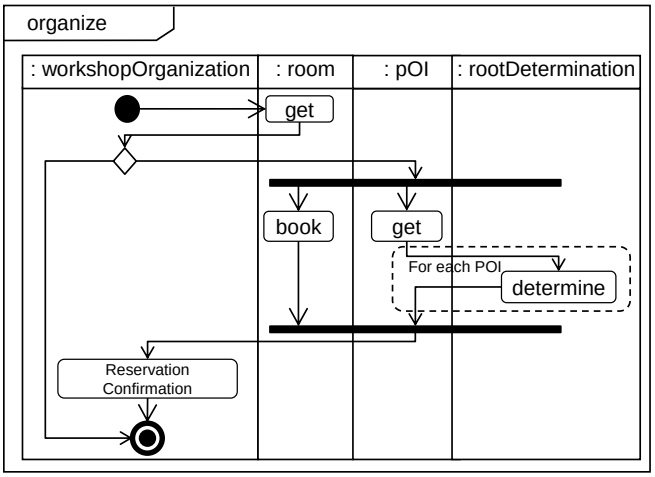


Figure 4. Interaction protocol for the operation "organize".

### C. Derivation of Executable Business Logic

The mapping rules provided by IBM [14] 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 4 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, which exists only once and consists of a sequence of other activities. 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 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".

```
<bpel:partnerLinks>
 <bpel:partnerLink name="client"
     partnerLinkType="WorkshopOrganization"
     myRole="WorkshopOrganizationProvider"/>
 <bpel:partnerLink name="room"
partnerLinkType="Room"
     partnerRole="RoomProcessProvider"/>
...
</bpel:partnerLinks>
```

Source Code 4. Derived partnerLinks in the BPEL process.

 ISBN: 978-1-61208-230-1 

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. Source Code 4 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. Each activity has the attributes “name” and “operation” set to the name of the action. The attribute “partnerLink” is set to the corresponding partnerLink prior defined. The activities are located within the corresponding scopes of flow elements mapped later. 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”.

### *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 5 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 [10].

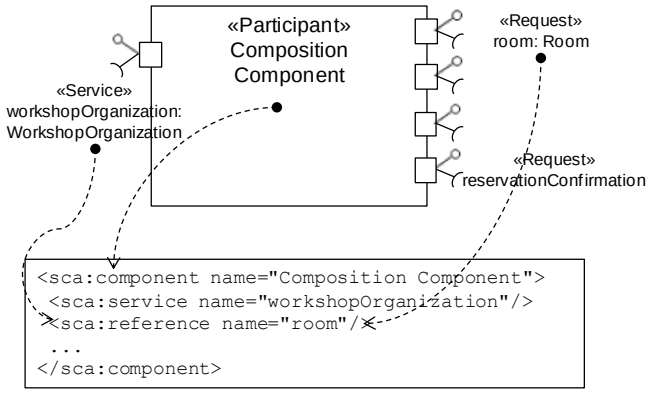


Figure 5. Derivation of SCA component model.

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 SCA Composite is the basic unit of a composition in an 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 6 illustrates the final SCA Composite in a graphical visualization as introduced by the standard.

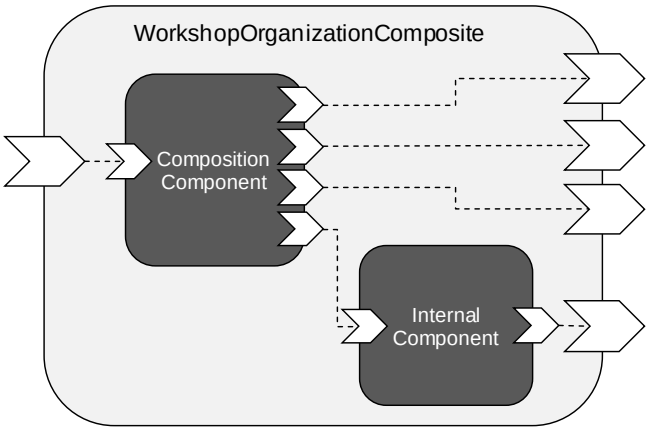


Figure 6. SCA Composite for the workshop organization process.

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 reprsenting 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.

 ISBN: 978-1-61208-230-1 

## V. CONCLUSION AND OUTLOOK

In this article, we illustrated the mapping between service designs that are based on SoaML as standardized modeling language and web service implementation artifacts. As most mapping rules between abstract formalizations and implementation artifacts focus on UML or SoaML in general, we identified mapping rules that can be used for a transformation of service designs as self-contained service development artifacts.

The usage of the mapping rules was illustrated by means of a business process for organizing workshops at a university. In this context several service designs have been created and the most complex one was transformed into implementation artifacts. As implementation technologies web services based on XSD, WSDL, BPEL, and SCA have been chosen as they are most wide-spread today.

By identifying and describing necessary mapping rules, on the one hand this article enables IT architects and developers to systematically transform service designs into implementation artifacts. As result, this supports the usage of SoaML within a model-driven development processes for services as models based on this language can act as valuable development artifacts. On the other hand, the mapping rules help to derive abstract service designs from already implemented web services. As we work on automatic quality analyses of services designs based on SoaML as introduced in [1] and exemplified in [17], a systematic derivation of service designs from implementation artifacts is necessary. As result the mapping rules described in this article support quality analysis processes in the context of service-oriented architectures.

In order to leverage the mapping rules within our QA82 Architecture Analyzer tool [28] that enables the automatic quality analysis of service designs, the rules will be implemented by means of Query Views Transformation (QVT) [27]. This enables the automatic derivation of service designs from implemented web services that can be analyzed regarding wide-spread quality attributes, such as loose coupling and autonomy. As result, IT architects and developers will be able to automatically evaluate, whether developed web services support the flexibility, maintainability, and cost-efficiency of the IT.

## REFERENCES


[1] 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.

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

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

[4] 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.

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

[6] 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]

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

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

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

[10] 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]

[11] 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]

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

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

[14] 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]

[15] 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.

[16] 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.

[17] 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.

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

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

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

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

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

[23] 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]

[24] 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]

[25] OMG, “Business process model and notation (BPMN)”, Version 2.0 Beta 1, 2009.

[26] OMG, “Unified modeling language (UML), superstructure”, Version 2.2, 2009.

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

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


 ISBN: 978-1-61208-230-1 