SAWSDL for Self-adaptive Service Composition

Teodoro De Giorgio, Gianluca Ripa, and Maurilio Zuccalà

CEFRIEL - ICT Institute Politecnico di Milano
Via Renato Fucini 2, 20133 Milano, Italy
{teodoro.degiorgio}@students.cefriel.it,
{gianluca.ripa,maurilio.zuccala}@cefriel.it
http://www.cefriel.it

Abstract. In this paper we present our experience in using SAWSDL for supporting the dynamic replacement of services inside a service composition even in presence of syntactic mismatches between service interfaces. We illustrate the guidelines we defined for adding semantic annotations to WSDL based service interface descriptions. We also present the case studies we implemented to apply our approach to some actual service oriented scenarios. Basing on the current results of our work, we finally evaluate the feasibility of using SAWSDL for solving mismatches between service interfaces at different levels and we propose some extensions.

Keywords: SAWSDL, Semantic Annotations, Service Mediation, Service Composition, Service Adaptation, Service-Oriented Architectures.

1 Introduction

In a previous paper [1] we exploited SAWSDL [8] descriptions to support adaptation in the context of service composition. Some sort of adaptation is usually needed when a service to be invoked is not available and therefore should be replaced with an equivalent one: a number of syntactic mismatches can take place in this case, mainly at the interface level (i.e., the two services differ in the names and parameters of the operations exposed) or at the protocol level (i.e., the two services differ in the order in which operations must be invoked), thus preventing the further execution of the overall service composition.

In order to address this problem, a mapping function between the expected service and the available one can be created every time it is needed. Current works adopting this solution usually require the manual definition of proper mapping functions at design time, thus encumbering the work of system integrators in charge of setting up service compositions. In [1] we showed that if service interfaces are semantically annotated, adaptive and dynamic replacement of services in a composition can be achieved in a more effective way. Moreover, thanks to our approach, service integrators do not need to be concerned with mapping services at the syntactical level, since we provide them with a language having a sufficient level of abstraction to define mapping functions in a more effective and user-friendly manner.

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 907–916, 2009.

In this paper we focus on the use of SAWSDL in the perspective of a service provider that aims to semantically annotate its services in order to allow their usage inside a self-adaptive service composition. The paper is organized as follows. Section 2 summarizes the problem. Section 3 presents our approach. Section 4 illustrates the case studies implemented in order to evaluate the approach with some actual service oriented scenarios. Section 5 presents a comparison with respect to other works in the same field. Section 6 draws some conclusions.

2 The Problem

In [2] we identified a number of possible simple mismatches between services, and some basic mapping functions that can be used to solve them. Such mapping functions can be combined in a script to solve complex mismatches. Scripts can be executed by a mediator that receives an operation request, parses it, and eventually performs the needed adaptations. This approach is incorporated in the SCENE [6] framework that we have developed as a part of the SeCSE project and that supports dynamic binding. A service is described by means of an interface and a protocol. Given two services S_1 and S_2, a mismatch occurs when an operation request expressed in terms of the interface of S_1 cannot be understood by S_2 that should handle it. We distinguish between the following classes of mismatches: interface level (i.e., differences between names and parameters of the operations exposed by S_1 and S_2) and protocol level (i.e., differences in the order in which the operations offered by S_1 and S_2 should be invoked). The basic mapping functions we defined to solve these mismatches are the following: *ParameterMapping*, *ReturnParameterMapping*, *OperationMapping*, *StateMapping* and *TransitionMapping*. For further details about mismatches, mapping functions, scripts and mapping language see [2]. This approach shows the following limitations:

- The definition of the script requires some human support to completely understand the mismatches and properly combine the mapping functions.
- An intensive effort is needed from system integrators that, in the worst case, have to specify an adapter for each service that needs to be invoked in the composition.
- It is complicated to add a new service showing mismatches at runtime, since system integrators should develop a new adapter.

In order to overcome these limitations and fulfill the requirements defined in the SOA4All project [9], we extended this approach by adding a semantic layer. The semantic layer is composed of the following elements:

- SAWSDL [8] service descriptions. WSMO-Lite [3] is an essential building block for the overall SOA4All architecture. WSMO-Lite realizes the WSMO [11] concepts in a lightweight manner using RDF/S and SAWSDL.
- A reference ontology. Compared to our previous approach in which system integrators have to build a mapping from one service description to another, now each service provider has to annotate a service description using a common domain specific ontology.

– A mapping script generator. An agent that, at pre-execution time, is able to process two SAWSDL descriptions and to automatically derive a mapping script.

3 Extending Our Approach with a Semantic Layer

SAWSDL offers three kinds of annotations: *modelReference*, *liftingSchemaMapping* and *loweringSchemaMapping*. These attributes can be used with elements of the WSDL definition. It is also possibile to attach multiple URIs in a single annotation, but in this case no logical relationship between them is defined by the SAWSDL specification.

While this flexible approach can promote the use of SAWSDL, it allows many degrees of freedom to the user. In order to solve our specific problem and to allow the automatic generation of mapping scripts, we add the following constraints:

1. Each operation must be annotated with a *modelReference* attribute,
2. The WSDL schema must be annotated with a *modelReference* attribute,
3. *loweringSchemaMapping* attribute is not used,
4. Optionally the elements of the WSDL schema can be annotated with a *liftingSchemaMapping* attribute,
5. We rely on the data type defined in XML Schema for simple types.

In the SAWSDL fragment below we show how we annotate operations. We add only a *modelReference* attribute to a *wsdl:operation* element[1]. During the generation process of the mapping script, the mapping script generator elaborates two SAWSDL descriptions. For each operation of the first SAWSDL, it extracts the model reference attribute value and searches in the second SAWSDL an operation annotated to the same URI. If such operation exists, the corresponding mapping function is generated.

```
<wsdl:portType name="portTypeName"><wsdl:operation
name="operationName">
  <sawsdl:attrExtensions
   sawsdl:modelReference="http://www.soa4all.eu/.../...#concept"/>
  <wsdl:input message="messageName"/>
  <wsdl:output message="anotherMessageName"/>
</wsdl:operation></wsdl:portType>
```

We do not annotate the *wsdl:input* and the *wsdl:output* elements. The next fragment identifies a data type definition for an input or output parameter of an operation within a service description.

```
<xs:element name="elementTypeName"><xs:complexType><xs:sequence>
   <xs:element name="elementName1" type="..."
    sawsdl:modelReference="http://www.soa4all.eu/.../...#concept2"/>
```

[1] Our examples refer to WSDL 1.1 but the approach can be equivalently applied to WSDL 2.0.

```
   <xs:element name="elementName2" type="..."
    sawsdl:modelReference="http://www.soa4all.eu/.../...#concept3"
    sawsdl:liftingSchemaMapping="http://www.soa4all.eu/.../name.ml"/>
   <xs:element minOccurs="0" name="elementName3" type="..." />
</xs:sequence></xs:complexType></xs:element>
```

In the listing above, four XML Schema elements are depicted. The first one (*elementTypeName*) and the last one (*elementName3*) are not annotated. The first one is only a container element and its semantics can be derived by the semantic annotations of its component elements. The last one is an optional element. Elements *elementName1* and *elementName2* are annotated with a *modelReference* attribute that contains a URI pointing to a concept of an ontology. *elementName2* is also annotated with a *liftingSchemaMapping* attribute that contains a URI that refers to a lifting schema espressed in a specific lifting schema language (see [1] for details). The presence of the *liftingSchemaMapping* attribute indicates the need for transformation between the WSDL element and the semantic model. An example of SOAP response message body, taken from an actual service invocation, can be found in the listing below.

```
<ServiceResponse xmlns="http://www.soa4all.eu/example.namespace">
 <return>0, Clear</return>
</ServiceResponse>
```

In this example the output parameter contains two data: the temperature and a weather condition label. This information can be represented by two concepts in the semantic model:

- *0* is the concept `http://www.soa4all.eu/models/Weather/weather#temp`,
- *Clear* is the concept `http://www.soa4all.eu/models/Weather/weather#description` in the semantic model.

One solution could be the insertion of these two concepts in the *modelReference* attribute, thus suggesting the content but not the way to extract the concept from the *return* element. So we create a custom lifting file that contains all the information required to solve this kind of mismatches.

We do not use the *loweringSchemaMapping* attribute because we are facing the problem of adapting a SOAP message prepared according to a specific WSDL for invoking a service with a different WSDL description. The *loweringSchemaMapping* attribute could be used for adapting a service request expressed at the semantic layer to a SOAP message, so it is out of scope with respect to our specific problem.

In order to annotate a WSDL service description as described above, a service provider needs an ontology that contains the right concepts with respect to the elements comprised in the WSDL specification. In Section 4 the ontologies used in our specific examples are described. They contain only concepts that we used to annotate WSDL for our examples. The last fragment represents a link between operations and parameters. For each operation it is possible to identify at least one *elementTypeName* of the XML Schema for the input and one for the output parameter using the *element* attribute.

```
 <wsdl:message
name="messageName">
 <wsdl:part element="elementTypeName"/>
</wsdl:message>
```

Once the WSDL description is annotated as described above, it is possible to automatically solve mismatches between services and perform the service interface adaptation.

4 Examples of Annotations for Service Adaptation

We applied our approach to some actual service oriented scenarios. Our case studies differ in the level of adaptation required and in the operational context (the scenarios are described in detail in [1] and [2]).

Figure 1 shows the vocabulary for semantic descriptions of services that we used and we required from the semantic model. Figure 1 contain only concepts that we used to annotate WSDL for our examples. We can use the concepts presented to annotate other services that are compatible to the services used in the case study. It can be extended by adding other elements (e.g., instances, relations, sub concept, axiom, parameter range) to obtain other tasks and use a different terminology, as it is possible to create differences between concepts that represent operations and parameters. It is not an objective of this work to create new ontologies. We derived the concepts needed for implementing the examples starting from some actual service descriptions.

4.1 Weather Forecast

The first example considers two stateless services: *TheWeather* and *Forecast*, each exposing one operation. According to our approach, we annotated the service interfaces using the ontology shown in Figure 1 to automatically solve mismatches between these two services. The elements of the interfaces mapped to

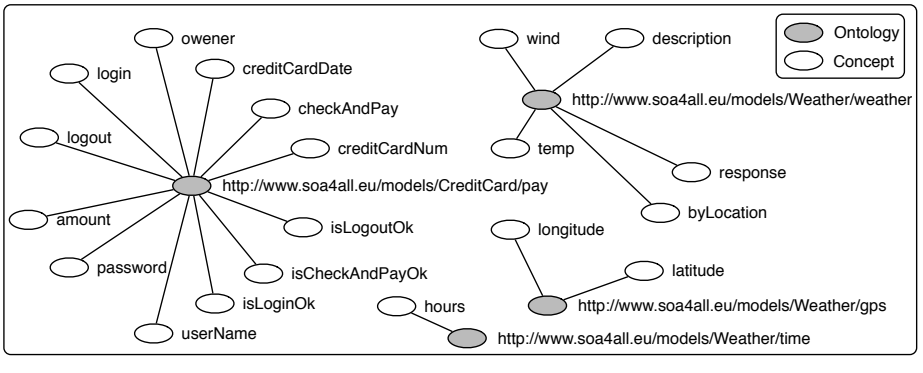

Fig. 1. Shared Domain Ontology

Table 1. Semantic Annotations for TheWeather

	TheWeather Service		Forecast Service	
	Name	**Semantic Annotation**	**Name**	**Semantic Annotation**
Operation	getForecast	weather#byLocation	Forecast	weather#byLocation
Schema element	latitude	gps#latitude	latitude	gps#latitude
	longitude	gps#longitude	longitude	gps#longitude
	hour	time#hours	hours	time#hours
	temp	weather#temp	return	weather#response
	wind	weather#wind		liftingForecast.ml
	weather	weather#description		

Table 2. Lifting Script for Forecast Service

Schema element	**Concept**	**Function**
return	weather#temp weather#description	stringSplitter(,)

the corresponding concepts of the ontology are shown in Table 1. The *Name* column represents the name of one WSDL element, while the *Semantic Annotation* column contains the value of the SAWSDL attributes. To help the visualization of data we omitted the common prefix of all these values, i.e., `http://www.soa4all.eu/models/`. These values represent resources that can be identified via URIs. In particular, we can distinguish the values in two groups: values coming from the attribute *modelReference* and *liftingSchemaMapping*. In Table 1 the only value pointing to the a lifting script is *liftingForecast.ml*. The content of *liftingForecast.ml* is summarized in Table 2 and it solves the same example described in the SOAP body response XML fragment depicted in Section 3. When all the information necessary to establish the semantic compatibility between the annotated element of the two services is provided by annotating the service description, it is possible to start the process to generate the mapping script. Figure 2 is a representation of the mapping between concepts present in the two services. It can be noted that the mapping between operations and input parameters is one to one, while return parameters *temp* and *weather* of *TheWeather* are mapped to the *return* element. Moreover, the *wind* element is not associated but the semantic compatibility is established due to the fact that

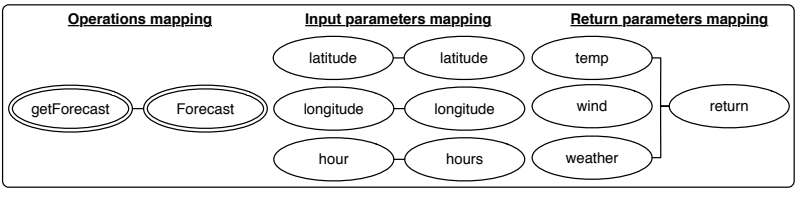

Fig. 2. Weather Mapping

Table 3. Semantic Annotations for PaymentCC

	PaymentCC Service		CCPayment Service	
	Name	**Semantic Annotation**	**Name**	**Semantic Annotation**
Operation	setupConf	pay#login	login	pay#login
	payCC	pay#checkAndPay	checkCC	pay#check
			payByCC	pay#pay
	logout	pay#logout	logout	pay#logout
Schema element	userName	pay#userName	user	pay#userName
	password	pay#password	password	pay#password
	success	pay#isLoginOk	success	pay#isLoginOk
	owner	pay#owner	owner	pay#owner
	CCNumber	pay#creditCardNum	credNum	pay#creditCardNum
	expirDate	pay#creditCardDate	expDate	pay#creditCardDate
	amount	pay#amount	amount	pay#amount
	accepted	pay#isCheckAndPayOk	checkOk	pay#isCheckOk
			completed	pay#isPayOk
	loggedOut	pay#isLogoutOk	loggedOut	pay#isPayOk

parsing the *wind* element shows that it is an optional element, for this reason all the required parameters are mapped each other and for our approach this is enough to establish that the two service interfaces are semantically compatible[2].

4.2 Credit Card

The second example consists of two stateful services: *PaymentCC* and *CCPayment*, composed by different operations. As in the previous example, the first step is to annotate the WSDL description with semantic annotations. We present the results in Table 3. In this case the common prefix is: `http://www.soa4all.eu/models/CreditCard/`.

Basing on the rules currently implemented in our approach, the two services are compatible and Figure 3 illustrates the final mapping. In Figure 3 the *payCC* operation of the *PaymentCC* service is mapped to two operations of the *CCPayment* service: *checkCC* and *payByCC*. In order to adapt the service request and to invoke the *CCPayment* service a missing information is needed: the sequence of the invocations required by the *CCPayment* service. This is a protocol mapping issue, as described in [2]. In detail the operations of the *CCPayment* service are not indipendent but some relationships exist between them. In this case the current approach requires the intervention of a human being. A possible solution could be to describe relationship between operations in the semantic model as described in [4] about behavioural semantics. Our approach is complementary to this one and considers a situation where the service provider does not develop a complete semantic model for its service but uses a categorization of possible

[2] This is an assumption of compatibility valid at pre-execution time. The actual invocation of the service can fail or produce unexpected results. This is managed by other components of our architecture that are able to react to these situations.

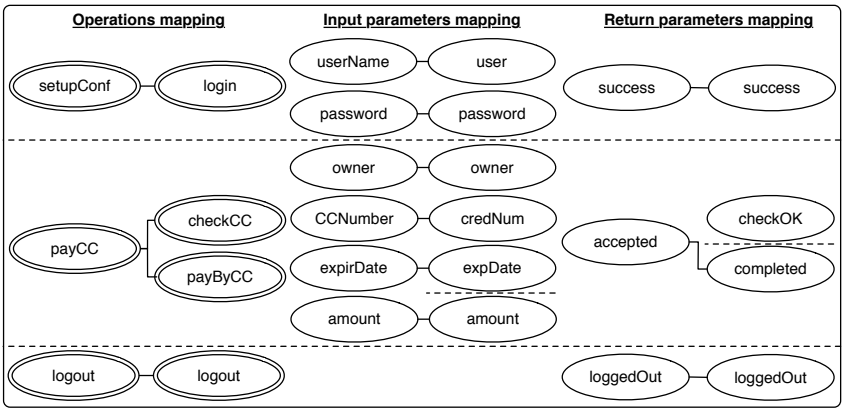

Fig. 3. Credit Card Mapping

relationships for annotating the operations listed in a WSDL. Thus another option could be to add a new attribute to the SAWSDL specification for enabling such kind of annotations and enabling the description of the order in which a client should invoke the operations of the Web Service.

5 Related Work

Service adaptation is a hot research topic in Service Oriented Architectures and Semantic Web Services. Here follows a brief description of some approaches related to SAWSDL that are particularly close to the topic of this paper. Other related work with respect to service adaptation are described in [1].

In [7] an approach for the semi-automatic creation of schema mappings based on a central ontology is developed. This approach aims to create mapping rules to transform an instance of the source schema to an instance of the target schema for solving data migration problems in an industrial scenario. The same approach can be applied also for solving the problem of adapting two Web Services, since the WSDL specification includes the use of XML Schema for the schema definition. This approach is similar to ours but it does not use directly SAWSDL as an annotation language because it is not specific for the Web Service adaptation scenario.

In [5] the SIROCO middleware platform is presented. It enables the dynamic substitution of stateful services that become unavailable during the execution of service orchestrations. It assumes that the service state description is provided with the service interface descriptions according to the standard WS-Resources Framework [12]. The information managed by the SIROCO Reconfiguration Management System consist of BPEL descriptions of service orchestrations, SAWSDL descriptions of each service used in the orchestration and SAWSDL descriptions of services that may be used for the reconfiguration of the orchestration. The reconfiguration process present in the SIROCO infrastructure is based on discovering candidate

substitute services, identifying semantically compatible services and synchronizing the state of the resources used by the services. In SIROCO candidate substitute services, whose state descriptions exactly match the state descriptions of unavailable services, are selected. These descriptions must be provided according to the WS-ResourceProperties standard specification.

6 Conclusions

Our semantically extended approach allows the invocation of services whose interface and behavior differ from each other by means of semantic annotations. The approach addresses the following requirements defined in the SOA4All project:

- Adaptation, since it enables the automation of the creation of adaptation scripts, thus overcoming some of the current limitations and issues connected to lack of standardization in service centric systems.
- User-friendly composition, since it allows the development of new added-value services in a lightweight and effective manner and it moves some tasks, in the process of developing added-value services, from the system integrator (e.g., manual mapping for adapting service requests to actual service interfaces) to each service provider (e.g., annotation of service descriptions).
- System self-reconfiguration, since it eliminates the complexity of adding new mismatching services at runtime to the service composition by developing service adapters.

The main advantage of our solution is the possibility to bring the service mapping language to a higher level of abstraction by means of semantic annotations of service interfaces: we achieve service composition by building service descriptions exploiting shared domain ontologies, instead of describing service mappings at a syntactical level. This approach greatly simplifies the work of SOA4All users, since it simplifies adaptive and dynamic service composition also in an open world setting.

It is possible to raise the level of abstraction reached by our approach using an additional semantic layer. An example of a semantic model that we are evaluating is WSMO-Lite [3], one of the main results of the SOA4All project [10]. In our approach we assume that all the services can be annotated by one domain ontology. There are many common scenarios for which this is applicable and semantic annotations already exists. In the examples proposed, this annotation is manually done to the ontology shown in Figure 1, but we assume that in the SOA4All project proper tools will be created for the annotation phase. Furthermore, we are working for enabling the mapping script generator to generate rules (i.e., state mapping and transition mapping rules) able to solve also protocol level mismatches. One example of protocol level mismatch is related to the credit card example in Section 4.2. These sample services list operations for enabling the payment: *login*, *check*, *pay*, *logout*. Strong dependencies exist between these operations and these dependencies are not described inside a WSDL. It

is possible to derive these dependencies from the SAWSDL annotations if the dependency relationships between concepts are included in the semantic model. However, in our approach we would allow the service provider to specify these dependencies by means of semantic annotations instead. Thus a possible extension to SAWSDL could be to add a new attribute for annotating the operation with this kind of information.

Acknowledgments

Parts of this work were sponsored by the European Commission in course of FP7 project SOA4All. The opinions expressed represent the authors' point of view and not necessarily the one of the project participants or of the EC.

References

1. Cavallaro, L., Ripa, G., Zuccalà, M.: Adapting Service Requests to Actual Service Interfaces through Semantic Annotations. In: PESOS Workshop. IEEE Computer Society Press, Vancouver (2009)
2. Cavallaro, L., Di Nitto, E.: An Approach to Adapt Service Requests to Actual Service Interfaces. In: SEAMS Workshops. ACM Press, Leipzig (2008)
3. Vitvar, T., Kopecký, J., Viskova, J., Fensel, D.: WSMO-Lite Annotations for Web Services. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 674–689. Springer, Heidelberg (2008)
4. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Computing (2007)
5. Fredj, M., Georgantas, N., Issarny, V., Zarras, A.: Dynamic Service Substitution in Service-Oriented Architectures. In: IEEE Congress on Services - Part I (2008)
6. Colombo, M., Di Nitto, E., Mauri, M.: SCENE: A service composition execution environment supporting dynamic changes disciplined through rules. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 191–202. Springer, Heidelberg (2006)
7. Drumm, C.: Improving Schema Mapping by Exploiting Domain Knowledge. Universität Karlsruhe, PhD Thesis (2008)
8. W3C: Semantic Annotations for WSDL and XML Schema. W3C Recommendation (2007), `http://www.w3.org/TR/sawsdl/`
9. SOA4All: Project Website, `http://www.soa4all.eu`
10. Ripa, G., Zuccalà, M., Mos, A.: D6.5.1. Specification and first prototype of the composition framework. SOA4All project deliverable (2009)
11. ESSI WSMO working group, `http://www.wsmo.org/`
12. Web Services Resource Framework. OASIS Standard (2006)