

An Approach to Enable Replacement of SOAP Services and REST Services in Lightweight Processes

Teodoro De Giorgio, Gianluca Ripa, and Maurilio Zuccalà

CEFRIEL - ICT Institute Politecnico di Milano

Via Renato Fucini 2, 20133 Milano, Italy

{teodoro.degiorgio,gianluca.ripa,maurilio.zuccala}@cefriel.it

<http://www.cefriel.it>

Abstract. In the last few years, several Web APIs implementing the REST principles were created. Process modellers often need to use both SOAP and REST services within a single process. In this paper we present our experience in using MicroWSMO for supporting dynamic replacement of SOAP and REST services inside a service composition, even in presence of syntactic mismatches between service interfaces. We also present a running prototype we implemented in order to apply our approach to lightweight processes execution.

Keywords: Service Adaptation, SOAP Services, REST Services, Semantic Annotations, Service Mediation, Service Composition, Service-Oriented Architectures, Lightweight Processes.

1 Introduction

In a previous paper [1] we exploited SAWSDL [10] 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, in [2] the authors defined a set of basic mapping functions. Since in real cases they often observe combination of mismatches, also the mapping functions can be combined to provide a solution to complex mismatches. These basic mapping functions are defined as follows:

- ParameterMapping: maps abstract service input data on concrete service input data.
- ReturnParameterMapping: maps abstract service output data on concrete service output data.

- `OperationMapping`: maps abstract service operations on concrete service operations.
- `StateMapping`: maps an abstract service state on a concrete service state.
- `TransitionMapping`: maps an abstract service transition on a concrete service transition.

Basic mapping functions can be combined in adaptation scripts, defined in a domain specific language. The language is composed of rules structured in two parts:

- A mismatch definition part that specifies the type of the mismatch to be solved by the rule, and contains two subelements: `input`, specifying the elements of the abstract service that show the given mismatch, and `mapping`, specifying the elements of the concrete service the input elements have to be mapped on.
- A mapping function part that contains the name of the function to be used to solve the mismatch.

For an in depth treatment of mismatches, mapping functions and mapping language see [2]. This solution usually requires 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. The idea behind the approach is that some ontology already exists and that adding semantic annotations to the service description is an easier task for the service provider than manually creating the mapping script for the service integrator. If two service providers use the same ontology for annotating their service descriptions, the definition of the mapping scripts can be automated.

We extended our approach described in [1] for supporting REST services. Indeed, in the last few years, several Web APIs implementing the REST principles were created as a lightweight alternative to SOAP. Since in our previous approach we exploited SAWSDL as a semantic description language for WSDL based services, we investigated some equivalent language in the context of REST services. In [7] MicroWSMO, a semantic annotation mechanism for RESTful Web services is described. It is based on a microformat called hRESTS (HTML for RESTful Services) for machine-readable descriptions of Web APIs, and it is backed by a simple service model which serves as a rough equivalent to WSDL. Our extended approach uses service descriptions written in MicroWSMO and SAWSDL format in order to enable for substituting a SOAP service with a REST one (and viceversa) as implementation of an activity of a process.

The approach was implemented in a prototype that is part of the Execution Engine developed in the SOA4All project [13]. SOA4All concentrates on empowering non-technical users to do simple IT modelling and development work in the area of service construction and composition. In SOA4All is introduced a Lightweight Process Modelling Language (LPML) for supporting users in the lightweight modelling based on three design principles:

1. abstraction of process models;
2. the use of semantic annotations;
3. context-awareness.

LPML reuses concepts mainly defined in BPMN [12] and BPEL [15] and add new modelling concepts for supporting these principles and to support users with advanced guidance during the modelling activities. The output generated by SOA4All tools starting from the business process graphically described by the users is a lightweight process described using BPEL [15] plus the following attributes extensions used to support the dynamic binding and service substitution at runtime:

- replacementCondition: this attribute represents an element in a taxonomy that defines the set of pre-defined replacement conditions. The replacement conditions are the situations in which the engine will try to substitute a service with another one.
- selectionCriteria: The selection criteria is the criteria applied by the engine for selecting a substitute service from a list of alternatives. The selection criteria are for example: concepts like the price of something, response time, service rating, and other.
- alternativeServiceList: this element lists the services that can be used as alternatives in the service substitution. is a list of alternativeService subelements:
- alternativeService: this element is an URI that points to the service description specification.

In this paper we focus on the execution of these lightweight processes. The Execution Engine component delegated to do this is able to analyze the attributes generated and to execute the process that contains in the alternative service list REST and SOAP services. The component is able to replace at runtime a SOAP service with a REST one, and vice versa. In this paper we illustrate the results of our research and the new solution developed for using SAWSDL and MicroWSMO for enabling the execution of self-adaptive processes.

The paper is organized as follows. Section 2 summarizes the problem. Section 3 presents our approach. Section 4 presents a comparison with respect to other work in the same field. Section 5 draws some conclusions.

2 The Problem

In many situations, process modellers need to use both SOAP and REST services within a single process. For the sake of clarity, in this section we refer to the specific sample process depicted in Fig. 1. This process is created by a modeller who needs to construct a weather forecast service that returns a weather forecast given the name of a city as input. The modeller, who can be supported by tools offered also from SOA4All project, searches for a weather forecast service.

She finds a WSDL/SOAP service that implements the required functionality and that is available for use with the required level of quality. The only problem

is that the service requires as input the geographical coordinates of a location and not the city name. Thus, the modeller searches for a service that, given a city name, returns its geographical coordinates. She finds such service implemented as a REST service. Combining these two services in a BPEL [15], LPML [14] or another process language, the modeller is able to fulfill the needed service.

The result is the sample process in Fig. 1, composed by two activities bounded at design time to two services, e.g., a REST service that gives access to a world-wide geographical database with a search function, and a SOAP service that provides weather forecasts. As depicted in Fig. 1, the sample process:

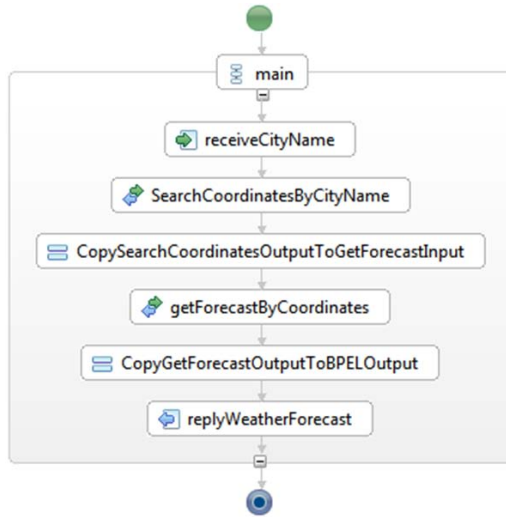


Fig. 1. Sample Process

1. Receives the city name string as input;
2. Invokes the search operation passing as input the name of the city, and obtaining as output the coordinates of the city, i.e., latitude and longitude;
3. The output of the previous activity is passed to the following activity as input;
4. Invokes the ForecastService, service passing as input the coordinates of the city, i.e., latitude and longitude, and obtaining as result the weather forecast for the city;
5. The output of the previous activity is passed to the following activity as input;
6. Returns the weather forecast.

During the modelling phase, the modeller discovers also some possible alternative services (that implement the same functionalities but with a lower level of quality) that can be used in order to successfully execute the composition in case a fault occurs during the invocation of one of the selected services. The Sample Process scenario proposed in this section requires to compose SOAP

and REST services in a single process definition. Aim of our approach is to allow the replacement of a failing SOAP service with a REST one or vice versa. Furthermore, such heterogeneous composition, in our scenario, can be built in a semi-automatic way without the need of heavy technical skills. Indeed we assume that the modeller is not an IT expert but a business domain expert.

3 Our Approach

The problem described in the previous section raises some questions:

- How to establish the semantic compatibility between REST services, and between REST services and SOAP services (given that we already solved the problem of the compatibility between SOAP services)?
- How to adapt REST services and SOAP services at runtime?

In order to support invocation of REST services and to enable service replacement and service message adaptation between SOAP and REST services, we use MicroWSMO service description for REST services. MicroWSMO, described in detail in [7], is a semantic annotation mechanism for REST services, based on a microformat called hRESTS (HTML for RESTful Services) for machine-readable descriptions of Web APIs. MicroWSMO adds SAWSDL-like annotations to hRESTS service descriptions. The SOA4All Studio provides SWEET [4], a tool used for annotating Web APIs in MicroWSMO. Starting from the SOA4All results and from the analysis of REST services and SOAP services we derived the information required for describing the characteristics of a REST service needed to allow the invocation and the replacement at runtime:

- The service description URL
- The URI template for each operation
- The name of the operations
- A model reference for each operation that refers to the corresponding service operation concept
- The name of the method (i.e., GET or POST)
- The list of the input parameters with the following information:
 - Parameter name
 - Parameter type (as XML Schema datatypes)
 - A model reference URI for each parameter
 - (If necessary) a reference to a lifting/lowering Schema for each parameter
- The output parameter with the following information:
 - MIME-type (default text/XML)
 - If the mime type is text/XML the annotated XML schema, otherwise the model reference URI
 - (If necessary) a reference to a lifting/lowering Schema

It is worth to be noted that the lifting and lowering schemas, that are part of both SAWSDL and MicroWSMO specifications, have to be used only in case of complex mapping logic, where some computation is necessary for translating

from one concept to another. Thus in our approach we rely mainly on the model-Reference attribute that we consider as a more lightweight means for annotating the service descriptions.

This minimum set of data is in line with the Lightweight RESTful Service Model described in [7] and [4].

This way of annotating can be used also for enhancing the discovery of a service by means of semantic search engines. In our experience, the opposite, i.e. using the annotation commonly created for enhancing the discovery for automating the service invocation/substitution at runtime, is very often not possible.

An example of a two fragments of service annotations for a REST and a SOAP service are given below. The following RDF fragment is the MicroWSMO description of the input parameter “latitude” of a service operation for a REST service.

```
<wsl:hasInputMessage>
  <wsl:Message>
    <sawsdl:modelReference
      rdf:resource="http://www.soa4all.eu/ontology/execution/parameters#Required"/>
    <sawsdl:modelReference
      rdf:resource="http://www.w3.org/TR/xmlschema-2/#double"/>
    <sawsdl:modelReference
      rdf:resource="http://www.soa4all.eu/ontology/execution/gps#Latitude"/>
    <rdfs:label xml:lang="en">lat</rdfs:label>
  </wsl:Message>
</wsl:hasInputMessage>
```

Below the same parameter is described in SAWSDL.

```
<xs:complexType>
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="1" name="latitude" type="xs:double"
      sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/gps#Latitude"/>
    ...
  </xs:sequence>
</xs:complexType>
```

The parameters in the MicroWSMO and in the SAWSDL descriptions are compatible. Thus, the Execution Engine starting from a set of annotations of the two service descriptions similar to the one in the example above, is able to establish the semantic compatibility and to automatically provide the service substitution in case of necessity.

Now we describe what happens at runtime. We supposed that a process is deployed and all the artefacts required to execute a process and to provide the self-adaptation at runtime are generated. We assume also that the process of execution starts and the executor of processes sends the input required for the invocation of the services expected from the process.

During the execution of the process the role of the Execution Engine is to check the result of the service invocation and in case of necessity adapt the request/response. The Service Adapter is the component delegated to adapt the messages of the real services provided by third party. During the execution of the process, the Execution Engine is able to bind to the expected service suggested in input and invoke it. As shown in Fig. 2, it is possible that the response of the Service A invoked is a fault message. In this case, by the analysis of the selection criteria and based on the alternative service list, Service B is selected

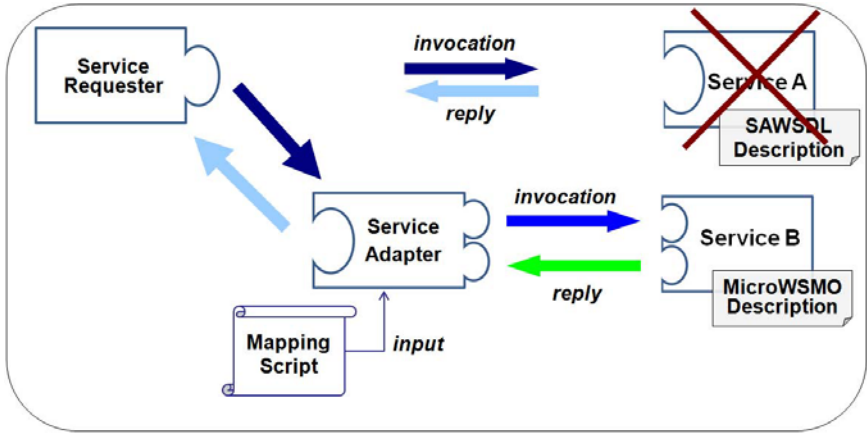


Fig. 2. Runtime Service Adaptation

as the new service to be invoked. In the case of the scenario in the sequence, the expected service is a WSDL/SOAP service and the alternative service is a REST service. The role of the Service Adapter in this second invocation is crucial for the success of the invocation. It consists of the following steps:

1. Receive from the Execution Engine the expected request and the reference to the alternative service;
2. Select the mapping script related to the adaptation from the expected service to the alternative service;
3. Execute the script and adapt the request to the alternative service using the mapping script selected;
4. Invoke the alternative service;
5. Receive the response from the alternative service;
6. Adapt the response to the expected service using the mapping script selected;
7. Send the adapted response to the Execution Engine.

Then, it is possible to continue the execution of the process, because the service replacement is transparent to the final user and it is internal to the Execution Engine.

4 Related Work

At the time of the writing of this paper, other initiatives are facing a similar problem. An example is Apache ODE [11] that proposes a REST variant of the BPEL invoke activity that replaces the attributes partnerLink/operation with resource and method. This new Apache ODE feature is not yet implemented and it is in the state of a proposal, subject to changes.

A similar attempt is the one described in [8] where the REST2SOAP framework is described. REST2SOAP aims at integrating SOAP services and RESTful

services in order to create a BPEL service that combines SOAP, REST services and user interfaces simultaneously. REST2SOAP leverages WADL specification, and can wrap RESTful services into SOAP services semi-automatically.

In [9] the author presents a hybrid approach that makes use of SOAP services and REST services. The system contains both a BPEL engine and a REST orchestration engine. A big workflow is divided into several sub-workflows according to their intrinsic architectural style. Then, each resulted sub-workflow is handled in a native way. SOAP-based orchestration is conducted in the BPEL engine, and REST services are orchestrated in the REST orchestration engine. Invocations help assemble the whole business process. Interactions between heterogeneous services are minimized.

All the above mentioned approaches enable the creation of orchestrations that combine SOAP and REST services but do not enable the dynamic substitution of each other at runtime.

In [3] an approach for replacement of services inside a service composition that focuses on conversational REST services is presented. The approach uses model checking techniques for automatically identifying the interaction protocols mapping. This approach can be combined with ours for enabling the mapping script generator to generate rules (i.e., state mapping and transition mapping rules) able to solve also protocol level mismatches.

In [5] an extension to the WS-BPEL standard process modeling language is proposed to support the native composition of RESTful services. The interaction primitives (GET, POST, PUT, and DELETE) stemming from the REST uniform interface principle can be directly used from within a BPEL process as new service invocation activities. The same authors, in [6], applied the notion of composition to RESTful services and derived a set of language features that are required by composition languages for RESTful services: dynamic late binding, dynamic typing, content-type negotiation, state inspection, and compliance with the uniform interface principle. They also presented a case-study using the JOpera visual composition language.

5 Conclusions

In this paper we described how to use SAWSDL and MicroWSMO for supporting the dynamic replacement of REST and SOAP services inside a service composition. The approach developed can help different groups of end users to build new services and processes according to their specific needs in a lightweight manner, since it hides a lot of technical details related to hybrid and heterogeneous orchestration, and to self-adaptiveness. The approach heavily relies on semantic annotation approaches for the generation of runtime artefacts starting from a process description. There are many common scenarios for which this is applicable and semantic annotations already exists. In the examples proposed, this annotation is realized, using the SOA4All project tools.

The approach and the prototype support both SOAP services and REST services described by using SAWSDL and MicroWSMO, and it is able to replace at runtime

a SOAP services with a REST one, and vice versa. 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.

At the time of writing we are facing the problem of supporting services that work with content types that are not XML-based.

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 projects participants or of the EC. We would like to thank Alessandro Marasco who is contributing to the implementation of the prototype.

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. Cavallaro, L., Di Nitto, E., Pradella, M.: An Automatic Approach to Enable Replacement of Conversational Services. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 159–174. Springer, Heidelberg (2009)
4. Maleshkova, M., Kopecky, J., Pedrinaci, C.: Adapting SAWSDL for Semantic Annotations of RESTful Services. In: OTM Workshops. Springer, Vilamoura (2009)
5. Pautasso, C.: RESTful Web service composition with BPEL for REST. *Data and Knowledge Engineering Journal* 68, 851–866 (2009)
6. Pautasso, C.: Composing RESTful services with JOpera. In: Bergel, A., Fabry, J. (eds.) *Software Composition*. LNCS, vol. 5634, pp. 142–159. Springer, Heidelberg (2009)
7. Kopecky, J., Gomadam, K., Vitvar, T.: hRESTS: an HTML Microformat for Describing RESTfulWeb Services. In: *Web Intelligence*. IEEE, Los Alamitos (2008)
8. Peng, Y.-Y., Ma, S.-P., Lee, J.: REST2SOAP: A framework to integrate SOAP services and RESTful services. In: *Service-Oriented Computing and Applications SOCA* (2009)
9. He, K.: Integration and orchestration of heterogeneous services. In: *Joint Conferences on Pervasive Computing JCPC* (2009)
10. W3C: Semantic Annotations for WSDL and XML Schema. W3C Recommendation (2007), <http://www.w3.org/TR/sawSDL/>
11. Apache ODE: Project Website, <http://ode.apache.org>
12. Business Process Modeling Notation BPMN: Project Website, <http://www.bpmn.org>
13. SOA4All: Project Website, <http://www.soa4all.eu>
14. SOA4All project deliverable: D6.3.2 Advanced Specification of Lightweight, Context-aware Process Modelling Language, <http://www.soa4all.eu/file-upload.html?func=showdown&id=127>
15. OASIS Web Services Business Process Execution Engine (BPEL): TC Website, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel