

Project Number: **215219**
 Project Acronym: **SOA4All**
 Project Title: **Service Oriented Architectures for All**
 Instrument: **Integrated Project**
 Thematic Priority: **Information and Communication Technologies**

D6.5.2 Advanced Prototype For Adaptive Service Composition Execution

Activity 2:	Core Research and Development
Work Package:	WP6 Service Construction
Due Date:	28/02/2010
Submission Date:	28/02/2010
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	CEFRIEL
Revision:	1.1
Author(s):	Gianluca Ripa, Teodoro De Giorgio (CEFRIEL), Yosu Gorroñoigoitia (ATOS), Adrian Mos, Fy Ravoajanahary (INRIA)
Reviewers:	Maria Maleshkova (OU), Matteo Villa (TXT)

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	25/11/2009	Table of content	Gianluca Ripa (CEFRIEL)
0.2	18/01/2010	First draft	Gianluca Ripa (CEFRIEL)
0.3	25/01/2010	Section 4	Gianluca Ripa, Teodoro De Giorgio (CEFRIEL)
0.4	01/02/2010	Section 3	Gianluca Ripa, Teodoro De Giorgio (CEFRIEL)
0.5	04/02/2010	Integrated comments and contributions	Gianluca Ripa, Maurilio Zuccalà (CEFRIEL), Yosu Gorroñoigoitia (ATOS)
0.6	09/02/2009	Installation and Use	Teodoro De Giorgio (CEFRIEL)
0.7	15/02/2009	Added section on LPML to BPEL transformation	Adrian Mos, Fy Ravoajanahary (INRIA)
0.8	15/02/2009	Integrated comments and contributions	Teodoro De Giorgio (CEFRIEL)
1.0	17/02/2009	Draft for internal review	Gianluca Ripa (CEFRIEL)
1.0.1	22/02/2009	Internal Review	Maria Maleshkova (OU)
1.0.2	23/02/2009	Internal Review	Matteo Villa (TXT)
1.1	25/02/2009	Integrated comments and contributions	Teodoro De Giorgio, Gianluca Ripa (CEFRIEL)

Table of Contents

EXECUTIVE SUMMARY	6
1. INTRODUCTION	8
1.1 INTRODUCTORY EXPLANATION OF THE DELIVERABLE	8
1.2 PURPOSE AND SCOPE	8
1.3 STRUCTURE OF THE DOCUMENT	8
2. THEORETICAL GROUNDING	9
2.1 THE PROBLEM	9
2.2 OUR APPROACH	11
2.3 EXAMPLES OF ANNOTATIONS FOR SERVICE ADAPTATION	15
3. DESIGN AND IMPLEMENTATION	18
3.1 ARCHITECTURE OF THE EE V2	18
3.1.1 <i>Architecture</i>	18
3.1.2 <i>Interfaces exposed</i>	20
3.2 IMPLEMENTATION	22
3.2.1 <i>Automatic Deployment of LPML models</i>	22
3.2.2 <i>Execution and self adaptation</i>	26
4. INSTALLATION AND USE	31
5. CONCLUSIONS	34
6. REFERENCES	35
ANNEX A. RDF EXTRACTED FROM MICROWSMO DESCRIPTION	37
ANNEX B. RESTFUL AND WSDL/SOAP SERVICE DESCRIPTION	38
ANNEX C. WEATHERFORECAST2THEWEATHER MAPPING SCRIPT	40

List of Figures

Figure 1 - SOA4All Studio “Create” buttons	9
Figure 2 - Sample Process	10
Figure 3 - MicroWSMO of the Weather Forecast service as displayed in a browser	13
Figure 4 - Domain ontology for the EE services.....	14
Figure 5 - Runtime service adaptation	15
Figure 6 - Execution Engine v2 component diagram	18
Figure 7 - Execution Engine v2 deployment diagram.....	20
Figure 8 - Sequence Diagram: Automatic Deployment of LPML	25
Figure 9 - Execution and self adaptation sequence diagram	27
Figure 10 - Integration between EE and the DSB	28
Figure 11 – EE v2 SOA4All source code tree	31

List of Tables

Table 1 - Candidate Services for the Sample Process.....	11
Table 2 - Concepts for supporting RESTful service description	12
Table 3 - Operations and parameters of the WeatherForecast service and of the TheWeather services.....	14
Table 4 - Monitoring Events.....	22
Table 5 - Mapping of LPML to BPEL Elements.....	24
Table 6 – EE v2 modules	31

List of Listings

Listing 1 – Mapping script fragment.....	14
Listing 2 – MicroWSMO fragment.....	16
Listing 3 – Description of latitude parameter in RDF format extracted from the MicroWSMO17	
Listing 4 – SAWSDL description of latitude parameter	17
Listing 5 – Example LPMDeployer SOAP Request.....	21
Listing 6 – Generic MicroWSMO description in RDF	37
Listing 7 – WeatherForecast service description in MicroWSMO based on RDF	38
Listing 8 – TheWeather service description in SAWSDL.....	39
Listing 9 – WeatherForecast2TheWeather mapping script	41

Glossary of Acronyms

Acronym	Definition
BPEL	Business Process Executable Language
D	Deliverable
EC	European Commission
EE	Execution Engine
EE v1	Execution Engine version 1
EE v2	Execution Engine version 2
HTS	Human Task Service
IM	Intermediate Model
LPML	Lightweight Process Modelling Language
MSG	Mapping Script Generator
PE	Process Editor
POM	Project Object Model
RDF	Resource Description Framework
RDFa	RDF in attributes
RDFS	RDF Schema
REST	Representational State Transfer
SAWSDL	Semantic Annotations for WSDL and XML Schema
SOAP	Simple Object Access Protocol
SWS	Semantic Web Service
WP	Work Package
WSDL	Web Services Description Language
WSMO	Web Service Modelling Ontology

Executive summary

The Advanced Prototype For Adaptive Service Composition Execution (EE v2) adds further functionalities to the lightweight composition environment presented in the first prototype, the EE v1 presented in [5]. EE v1 is able to adapt service requests to actual service interfaces at execution time, it supports the service selection according to functional, non-functional and contextual information and provide fault handling and dynamic rebinding mechanisms. The new functionality of EE v1 are:

- **RESTful services support:** EE v1 was able to work only with Web Services based on the WSDL and SOAP standards; the EE v2 overcomes this limitation allowing a process to use WSDL/SOAP and RESTful Web services at the same time transparently.
- **Automatic deployment of LPML models:** when the EE v1 was released, the lightweight process modelling language was still under development, now the EE v2 includes full support for processes described in the LMPL as it allows the deployment of LPML models automatically from the Process Editor¹ (described in [24]) by translating them into executable processes/services.
- **Support for Human tasks:** EE v2 include the support for processes that contains activities that are executed by humans such as checking requests for completeness and correctness. These activities are called human tasks.

As for the integration aspects, the EE v2 is ready to be integrated with some of the other SOA4All components, namely:

- **Process Editor:** the EE v2 exposes a Web Service that provides access to the deploy functionality of the EE; this service is invoked by the Process Editor when the user asks for the execution of a newly developed process.
- **Consumption Platform:** when a process is deployed, it can be accessible as an HTTP/SOAP Web Service; the SOA4All component in charge of invoking the executable process is the Consumption Platform.
- **Analysis platform:** during execution, the EE v2 generates events that trigger messages sent to the Analysis Platform, that is the component in charge of displaying these events to the users.

Furthermore, the EE v2 includes some internal improvements, with respect to the previous prototype:

1. decoupling from BPEL engine implementation: while the EE v1 supported only Active BPEL, now the EE v2 supports any BPEL engine implementation²;
2. update to the latest version of Axis2;
3. general refactoring and re-design for simplifying the architecture of the component.

The EE v2 is available with its source code through the SOA4All SVN repository.

Finally, a final release of the EE prototype is planned by the end of August 2010 (month 30 of the SOA4All the project). In the final release further improvements will be implemented, if needed, based on the feedbacks coming from the implementation of the SOA4All use cases,

¹ The component of the SOA4All Studio that allows a user to build processes in a lightweight manner, without requiring heavy technical skills.

² Actually the EE v2 was tested with Active BPEL and Apache ODE.

in addition to some refinements and bug fixes. This document describes in detail all the above-mentioned aspects.

1. Introduction

1.1 Introductory explanation of the deliverable

The first prototype of the composition framework (EE v1) [5], released on February 2009 (month 12 of the SOA4All project) was built on top of SCENE [12] and incorporated a new approach for adapting service requests to actual service interfaces through semantic annotations. The approach was described in detail in [5], [14] and [15].

The Advanced Prototype For Adaptive Service Composition Execution (EE v2) is the new updated version of EE that is in charge of translating the LPML models into executable processes, exposing them as services and executing them. It interacts with the Process Editor, the Consumption Platform and the Analysis Platform.

This document contains the description of the software release of the EE v2. This document highlights also the theoretical background of the new features of the EE v2.

1.2 Purpose and Scope

This document aims to describe the Execution Engine v2 in all its aspects, with specific emphasis on the novelties and on the improvements with respect to the previous release. This document describes the software components developed for the EE v2, the new approach adopted for exploiting semantic annotations in self-adaptation at runtime and the definition of interfaces for the integration in the SOA4All Runtime and SOA4All Studio. To help the description different examples are provided. Such examples are used also as test cases.

Furthermore, this deliverables shows the architecture of the Execution Engine.

1.3 Structure of the document

Section 2 introduces the approach used for allowing the RESTful services support in the context of process execution.

Section 3 describes the design and implementation of the Advanced Prototype For Adaptive Service Composition Execution.

Section 4 depicts the details about the installation and use of the EE v2.

Finally, section 5 draws some conclusion.

2. Theoretical Grounding

In the deliverable [5] we presented SAWSDL as a service description language for supporting the dynamic replacement of Web Services inside a service composition, even in presence of syntactic mismatches between service interfaces. Starting from the results shown in [12] that identify a number of possible mismatches between services, and some basic mapping functions that can be used to solve that mismatches. 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. In [14] and [15] we describe how to use SAWSDL for the automatic generation of the scripts for the mapping between Web Services based on WSDL descriptions. EE v1 includes the implementation of this approach.

During the second year of the SOA4All project, we added to the EE the support for REST services. Indeed, in the last years, a lot of web APIs implementing the REST principles were created as a lightweight alternative to HTTP/SOAP. Moreover, the support for REST services is a SOA4All requirement coming from the SOA4All Use Cases.

Since SOA4All proposes MicroWSMO [4] as semantic service description language for REST services, we implemented the support for REST services grounding on MicroWSMO. Thus, the EE v2 is able to use service descriptions in MicroWSMO and SAWSDL format and to invoke both HTTP/SOAP and REST services. Furthermore, since the Execution Environment (from the first release) is able to react to contextual changes, faults and misbehaviours of the component services of a process by replacing a service with a different one, the EE v2 is able to substitute a HTTP/SOAP service with a REST service as implementation of an activity of a process.

The EE v2, released at month 24 of the SOA4All project, can deploy and execute lightweight processes, developed using the Process Editor, that require to invoke both Web Services and RESTful services. Furthermore, the EE v2 is able to replace at runtime a Web Service with a RESTful one, and vice versa.

In this section 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.

2.1 The problem

In many situations, process modellers can have the need to use both WSDL and REST services within a single process. For the sake clarity in this section we refer to the specific sample process depicted in Figure 2. This process is created by a modeller that has the need to construct a weather forecast service that returns a weather forecast given as input the name of a city. First of all, the modeller uses the SOA4All Studio “Discover” feature (depicted in Figure 1) for searching for a weather forecast service.



Figure 1 - SOA4All Studio “Create” buttons

He 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 search for a service that given a city name returns its geographical coordinates. He finds such service implemented as REST. Combining these two services in a LPML process the modeller is able to build the needed service. He can do this assisted by the SOA4All

Studio Compose [2] functionality.

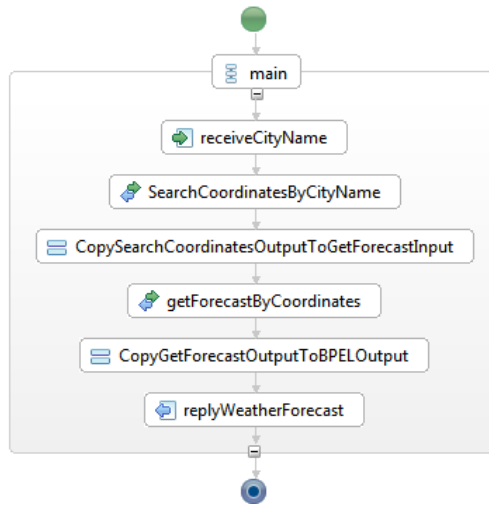



Figure 2 - Sample Process

The result is the sample process in Figure 2, composed by two activities (identified by the  symbol in Figure 2) bounded at design time to two services:

- GeoNames³, a RESTful service that gives access to a worldwide geographical database with a search function,
- ForecastService⁴, a Web Service, based on WSDL and SOAP that provides weather forecasts.

As depicted in Figure 2, the sample process:

1. receives the city name string as input,
2. invokes the search operation of GeoNames 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 implements 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 the GeoNames and/or ForecastService. This can be done without changing the process but adding specific semantic annotations to the activities as described in [6].

Table 1 lists the available candidate services for the implementation of the sample process activities and the set of alternatives selected by the modeler.

³ <http://www.geonames.org/export/geonames-search.html>

⁴ <http://zulu-53.nebula.fi/ws/services/ForecastService?wsdl>

Table 1 - Candidate Services for the Sample Process

Replacement for	WS Name	WS Type	WS Description
-	GeoNames	REST	http://www.geonames.org/export/geonames-search.html
-	ForecastService	WSDL/SOAP	http://zulu-53.nebula.fi/ws/services/ForecastService?wsdl
ForecastService	TheWeatherCRF	WSDL/SOAP	http://inrete.dyndns.info/TheWeather/TheWeatherCRF.asmx?wsdl
ForecastService	GeoNamesWeather	REST	http://www.geonames.org/export/JSON-webservices.html
ForecastService	WeatherForecast	REST	http://demo.cefriel.it/rest/services/WeatherForecast.htm
ForecastService	TheWeather	WSDL/SOAP	http://demo.cefriel.it/axis2/services/TheWeather?wsdl
GeoNames	GeoService	WSDL/SOAP	http://demo.cefriel.it/axis2/services/GeoService

In conclusion, the Sample Process scenario proposed in this section requires to compose heterogeneous SOAP and REST services in a single process definition.

At the time of the writing of this deliverable, other initiatives are facing a similar problem. An example is Apache ODE [17] that propose a RESTful variant of the BPEL invoke activity that replaces the attributes partnerLink/operation with resource and method. At the time of writing, this new Apache ODE feature is not yet implemented and it is in the state of a proposal, subject to changes based. Even if implemented this solution will solve only the first part of our problem because it allows to build a process that can have SOAP services for implementing some activities and REST services for implementing some other activities. This approach does not allow for replacing a failing SOAP service with a REST one or vice versa.

Furthermore, such heterogeneous composition, in our scenario, must be built in a semi-automatic way without the need of heavy technical skills. Indeed we assume, as a requirement for SOA4All, that our modeller is not an IT expert but a business domain expert.

2.2 Our approach

The problem described in the previous section raises some questions:

- How to define guidelines for the semantic annotation of RESTful services?
- How to establish the semantic compatibility between RESTful services and between RESTful services and Web Services? Given that we already solved the problem of the compatibility between Web Services.
- How to adapt at runtime RESTful services and Web Services?

In order to support invocation of RESTful services in the EE v2 and to enable service replacement and service message adaptation between WSDL/SOAP and RESTful services, we use the MicroWSMO service description for RESTful services.

MicroWSMO (described in detail in [4]) is a semantic annotation mechanism for RESTful Web 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. SOA4All Studio provides SWEET, a tool used for annotating Web APIs in MicroWSMO.

Starting from the analysis of the RESTful services and Web Services we derived the minimum information required for describing the characteristics of a RESTful service needed by the EE for allowing the invocation and the replacement at runtime:

- The service description URL;
- The URI template for each operation⁵;
- 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 follows 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

This minimum set of data is in line with the Lightweight RESTful Service Model described in [16] and [4].

In order to support the user in annotating the service descriptions with the needed information, we defined two ontologies:

- <http://www.soa4all.eu/ontology/execution/parameters>,
- <http://www.soa4all.eu/ontology/execution/mimetypes>.

Furthermore, we rely on the XML Schema Datatypes definition. Table 2 shows some concepts used for annotating the service descriptions involved in the sample process introduced in section 2.1.

Model References
http://www.w3.org/TR/xmlschema-2/#integer
http://www.w3.org/TR/xmlschema-2/#double
http://www.soa4all.eu/ontology/execution/parameters#Required
http://www.soa4all.eu/ontology/execution/parameters#Optional
http://www.soa4all.eu/ontology/execution/mimetypes#Text_xml

Table 2 - Concepts for supporting RESTful service description⁶

⁵ In this context the term operation refers to the meaning defined in [4]: “operation: a single action that the client can perform on a service”.

⁶ It has to be noted that the SOA4All WP3 team is defining a common minimal service model, which will detail the two ontologies that we have sketched in Table 2.

In order to better explain why in the EE v2 this information is required, we provide a comparison between two simple compatible services: the first WeatherForecast⁷ described using MicroWSMO annotations and the second TheWeather⁸ described with SAWSDL.

Our approach starting from the two service descriptions is able to establish the semantic compatibility and to automatically provide the service substitution in case of necessity. In order to do this we should define some criteria of compatibility and map information between these two service descriptions.

Weather Forecast service API

This is a service of [Weather Forecast](#)

Operation `getForecastByLocation`

POST <http://demo.cefriel.it/rest/services/WeatherForecast/byLocation?{-join|&|lat,lon}>

Parameters:

([lat](#)) lat (**Required**) [*double*] - the latitude of the location
 ([lon](#)) lon (**Required**) [*double*] - the longitude of the location

Parameters in Post:

([hour](#)) hour (**Optional**) [*integer*]
 ([min](#)) min (**Optional**) [*integer*] - the time of the forecast

Output value: weather and temperature of Location in [xml](#) format type.
[Xml Schema](#)

Figure 3 - MicroWSMO of the Weather Forecast service as displayed in a browser

Figure 3 shows the human readable service description of a sample RESTful service. The source code of this page is a MicroWSMO service description in RDF. As described in D3.4.3, MicroWSMO allows HTML service documentation to be annotated with service semantics in the same way that WSDL is annotated with SAWSDL. In section 4.1 of D3.4.3 the minimal service model is described: the interaction of a service consumer with a RESTful service is a series of operations where the consumer sends a request to a resource (using one of the HTTP methods GET, POST, PUT or DELETE) and receives a response that may link to further resources. In D3.4.3 there is the RDFS realization of this service model and the extension with semantic annotation.

Annex A shows an instance of the service model extracted from the MicroWSMO description of the WeatherForecast RESTful service and the SAWSDL service description of TheWeather service.

Referring to the two services descriptions in Annex A, in Table 3 we show how the EE v2 is able to understand if two services are semantically compatible and adapt the service messages by describing analogies and differences between the two service descriptions.

	WeatherForecast RESTful service		TheWeather WSDL/SOAP service	
	Name	Semantic Annotation	Name	Semantic Annotation
Operation	getForecastByLocation	weather#ByLocation	getForecast	weather#ByLocation
Parameter	lat	gps#Latitude	latitude	gps#Latitude

⁷ <http://demo.cefriel.it/rest/services/WeatherForecast.htm>

⁸ <http://demo.cefriel.it/axis2/services/TheWeather?wsdl>

Parameters	lon hour min	gps#Longitude time#Hours time#Minutes	longitude hour min	gps#Longitude time#Hours time#Minutes
Output Parameters	previsione temperatura	weather#Description weather#Temp	weather temp wind	weather#Description weather#Temp weather#Wind

Table 3 - Operations and parameters of the WeatherForecast service and of the TheWeather services

The semantic annotations shown in Figure 4 refer to a domain ontology shared between services and created to annotate the service interfaces⁹.

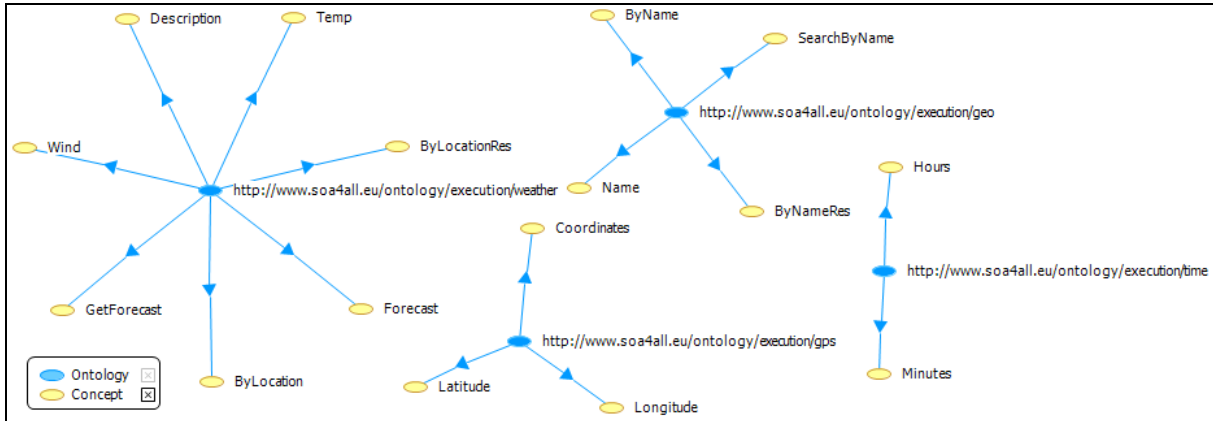


Figure 4 - Domain ontology for the EE services

By the analysis of the semantic annotations, it is possible to establish the semantic compatibility between the two services. In fact, only the 'wind' parameter, that is an optional parameter, cannot be mapped to a parameter of the WeatherForecast service.

To adapt the messages between RESTful and WSDL/SOAP messages as described in [1] we use the same mapping functions used to solve the mismatches identified between WSDL/SOAP services, but taking care of using the right information that distinguishes WSDL/SOAP services from RESTful services. For example, in case the TheWeather is the expected service and the WeatherForecast is the available service, the instance of the operation rename function in the mapping script will be the one depicted in Listing 1.

```

<OperationRename>
  <ExpectedService>
    <OperationName>getForecast</OperationName>
  </ExpectedService>
  <AvailableService>
    <URITemplate>
      http://demo.cefriel.it/rest/service/getForecastByLocation?{-join|&&|lat,lon}
    </URITemplate>
    <Method>POST</Method>
  </AvailableService>
</OperationRename>
    
```

Listing 1 – Mapping script fragment

In Listing 1 we can see the new mapping language keywords introduced in the EE v2, namely: URITemplate and Method. The main difference is that, in order to properly invoke

⁹ It would be also possible to use different ontologies for annotating the services descriptions. If a mediation between the ontology is possible then, at a conceptual level, we are in the same situation of having all the annotations based on one shared domain ontology except from the technical details.

the RESTful services, the EE requires an URITemplate for each operation. In addition, Annex C provides the complete mapping script automatically generated from the EE v2 deployer component starting from the two service descriptions. In Annex C it is possible to see all the information provided in each mapping function. The example shows the simplest case of adaptation between services, while the approach as described in [1] includes other mapping functions used to solve complex situations like protocol mismatches.

We described above how to use SAWSDL and MicroWSMO for supporting the dynamic replacement of RESTful and WSDL/SOAP services inside a service composition. At runtime the behaviour of the EE is exactly the same of the EE v1 as shown in Figure 5, despite the fact that now the candidate services can be both WSDL/SOAP services or REST services.

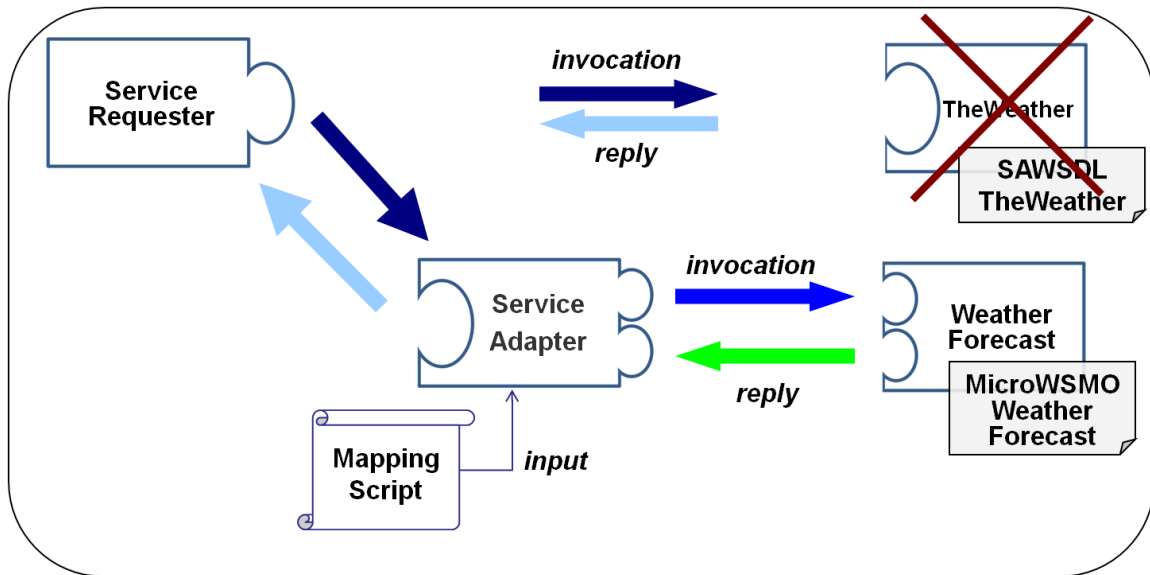


Figure 5 - Runtime service adaptation

2.3 Examples of Annotations for Service Adaptation

SOA4All Work Package 3 (Service Annotation and Reasoning), has defined languages (SAWSDL and MicroWSMO), guidelines and tools for annotating the Web service descriptions of REST and WSDL services. In our Execution Environment we rely on these specifications and it is out of scope of T6.5 to define new languages and tool for annotating Web services descriptions.

In this section, we describe in detail an approach for annotating service description that is a specialization of the approaches defined in the WP3 targeted to the specific elements needed for allowing the self-adaptive execution of the LPML processes.

As already mentioned, our approach is based on MicroWSMO, an extension of hREST that adds semantic annotations to an HTML description like SAWSDL adds semantic annotations to a WSDL service description. Furthermore, MicroWSMO adds to hREST the same attributes that SAWSDL adds to WSDL, namely: modelReference, liftingSchemaMapping and loweringSchemaMapping. Such attributes can be used with elements of the WSDL and hREST definition. It is also possible 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 for the annotation of the service

descriptions:

1. Each operation must be annotated with a modelReference attribute;
2. Each input or output parameter, except from the optional ones, must be annotated with a modelReference attribute;
3. We rely on the data type defined in XML Schema for simple types:
 - Each input or output parameter not defined with an XML Schema¹⁰, must be annotated with a modelReference attribute that points to the relative types defined in "<http://www.w3.org/TR/xmlschema-2/#>";
 - Each input or output parameter of type anyURI¹¹, must be annotated with a modelReference attribute that points to the <http://www.soa4all.eu/ontology/execution/mimetypes> ontology, that contains the concepts represented by all the supported content types (e.g. text/xml);
4. In case an input or output parameter is of type anyURI and content type is text/xml, the description must include the XMLSchema related to it and the XMLSchema must be annotated à la SAWSDL.
5. The optional input or output parameter must be annotated with a modelReference attribute in order to allow the adaptation also of these parameters in case they are present in both the services involved in the adaptation process.

As for SAWSDL service descriptions, we shown different examples of annotation in [1], here we show some examples of annotations for RESTful services descriptions in order to clarify the requirements listed above.

An example of annotation of an input parameter is shown in Listing 2.

```

<span rel="wsl:hasInputMessage">
  <span typeof="wsl:Message">
    <strong>Parameters:</strong><br />
    <code><a rel="sawSDL:modelReference"
      href="http://www.soa4all.eu/ontology/execution/gps#Latitude">lat</a>
    <span property="rdfs:label">lat</span>
    </code><a rel="sawSDL:modelReference"
      href="http://www.soa4all.eu/ontology/execution/parameters#Required">Required</a>
    <i><a rel="sawSDL:modelReference"
      href="http://www.w3.org/TR/xmlschema-2/#double">double</a>
    </i> - the latitude of the location<br />
  </span>
</span>

```

Listing 2 – MicroWSMO fragment¹²

From the MicroWSMO service description it is possible to extract an RDF document. The fragment above is transformed in the following RDF:

```

<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>

```

¹⁰ This may happen with RESTfull services

¹¹ <http://www.w3.org/TR/xmlschema-2/#anyURI>

¹² The semantic annotation format used is included in the MicroWSMO service description using the RDFa (see D3.4.3 for details).

Listing 3 – Description of latitude parameter in RDF format extracted from the MicroWSMO

In Listing 3, it is possible to see the description of an input parameter of a service operation for a RESTful service, in Listing 4 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>
```

Listing 4 – SAWSDL description of latitude parameter

As shown in Listing 3 and Listing 4, the parameters in the MicroWSMO and in the SAWSDL descriptions are compatibles. Thus, the EE v2 is able to adapt one service to the other.

3. Design and Implementation

This section depicts the internal architecture of the component and implementation details.

3.1 Architecture of the EE v2

3.1.1 Architecture

This section describes the architecture and implementation details of the Execution Engine advanced prototype. Figure 6 depicts the component diagram of the EE v2. The EE v2 is composed of three main subcomponents, described in the following sections.

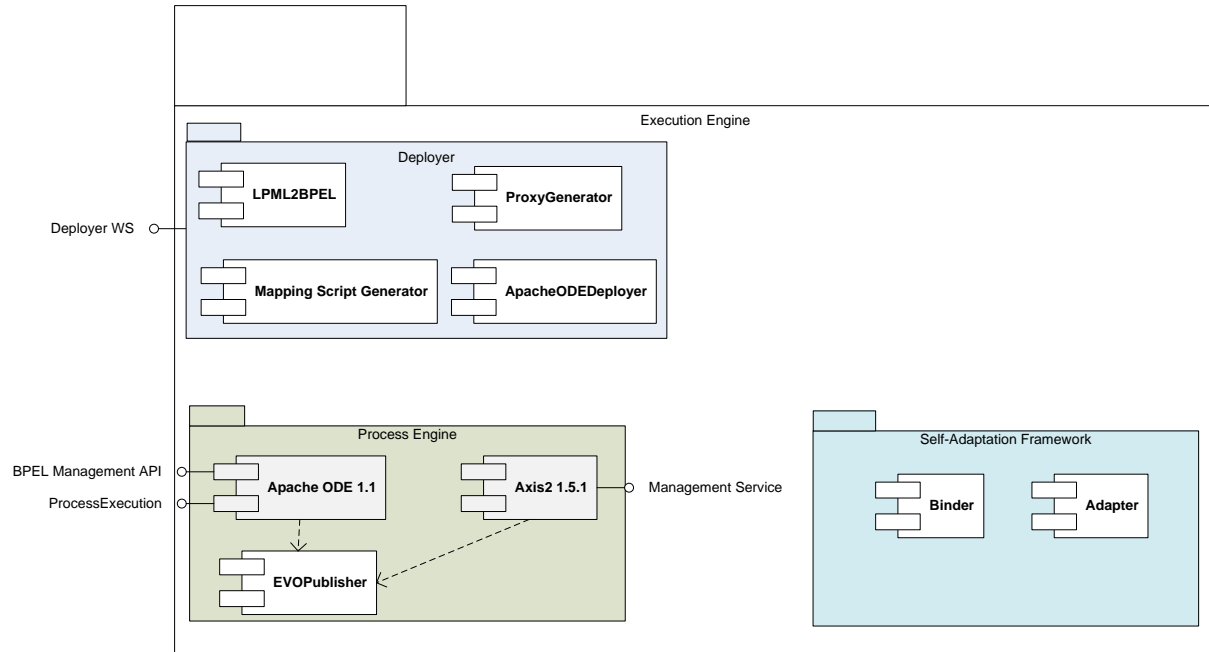


Figure 6 - Execution Engine v2 component diagram

EE v2 is part of the SOA4All architecture described in [1]; it is connected, interacts and exchange information mainly with the follows components briefly described:

- **Process Editor:** it allows users to create, modify, share, and annotate process models based on the LPML.
- **Consumption Platform:** it is the gateway for users to the service world when they act as consumers. The platform allows them to formalise their desires in several ways, defining and refining goals that can be used to discover and invoke the services that fulfil their needs.
- **Analysis Platform:** it obtains information (monitoring events) from the monitoring subsystem (e.g., EE v2) and performs processing in order to extract meaningful information.
- **SWS Repository:** it is a common SWS description repository that contains the SWS descriptions used by the SOA4All component, in this case the EE v2.

The following sections fulfil also the integration aspects between the EE v2 and of the other SOA4All components.

3.1.1.1 Deployer

The Deployer is the component that more than the others makes use of the semantic annotations derived from the LPML and from the services descriptions. It uses this semantic information for generating:

- a BPEL process definition from the LPML coming from the Process Engine;
- the BPEL package to be deployed in the Process Engine under Apache ODE;
- a Java proxy for each activity in the BPEL, to be deployed in the Process Engine under Axis2;
- a mapping script to be deployed in the Adapter that in this way is instructed for adapting the proxy requests for the use with all the candidate services identified for each activity.

3.1.1.2 Process Engine

The Process Engine is composed by third party components that can be plugged and unplugged in the architecture.

It is composed of the following sub-components:

- a standard BPEL 2.0 engine (Apache ODE v1.1) responsible of executing BPEL processes generated by the LPML2BPEL transformation at deploy time;
- a SOAP server Axis2 v1.5.1, responsible of executing service proxies auto-generated by the deployment process,
- the EVOPublisher: an extension of the component developed in the SUPER IP project, responsible of sending monitoring events to the Analysis Platform.

It is worth to note that in the previous version of the EE none of the components of the Process Engine v2 was present. In fact, the previous version was based on Active BPEL 2.1 and Axis2 1.1, and the EVO publisher was absent.

3.1.1.3 Self-Adaptation Framework

The Self-Adaptation Framework is the core part of the EE since it implements all the self-adaptation mechanisms described in section 3 and in [5]. The sub-components of the Self-Adaptation Framework are:

- The Binder: responsible for executing binding (and re-binding) actions at runtime based on the directions defined by rules. This component is able to execute various policies for selecting the candidate services. For instance, the services could be selected from a predefined list. The selection could be based both on functional and non-functional attributes expressed as Goals in the SOA4All terminology.
- The Adapter: responsible for executing adaptation actions at runtime based on the mapping scripts auto-generated at deploy time exploiting the semantic descriptions of the Web Services and of the RESTful services, as described in detail in section 2.2.

3.1.1.4 Runtime View

At runtime (see Figure 7) the EE v2 interacts with the Process Editor through the Deployer WS interface, with the Consumption Platform through the ProcessExecution interface, with the Analysis Platform through the EVOPublisher and with the third party Web Services and RESTful services invoked directly by the service proxies.

Internally, at runtime, besides the components already described in Figure 6 there are some other components. These components are the ones automatically generated, compiled and deployed at deploy-time by the Deployer. They are:

- DeployableBPELProcesses: they are BPEL processes that implements the same control flow and data flow of the original LPML model and that are exposed as Web Services by the Apache ODE BPEL Engine; the DeployableBPELProcesses are instructed for invoking the GeneratedProxies instead of the actual services.

- **GeneratedProxies:** they implements all the runtime self-adaptation characteristics of the EE¹³; they are exposed as Web Services by the Axis2 SOAP server and they are invoked by the BPEL process instance running in Apache ODE;
- **BindingRules:** the binding rules are used by the Binder for selecting a concrete service implementation for executing an activity;
- **Mapping scripts:** they are XML scripts used by the Adapter when a proxy has to invoke a service and to adapt its request/response messages.

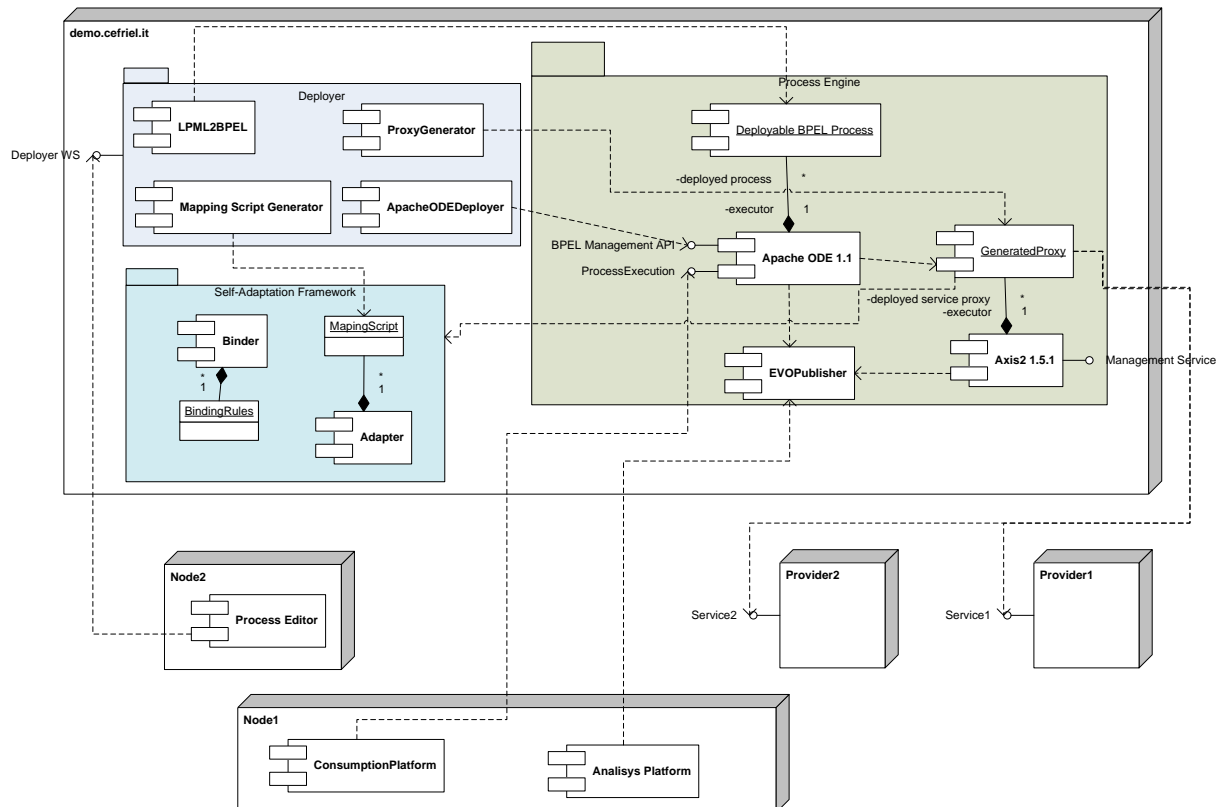


Figure 7 - Execution Engine v2 deployment diagram

3.1.2 Interfaces exposed

3.1.2.1 Deployment Interface

The deployer component expose the LPMDeployer Web Service as interface for the other components. LPMDeployer Web Service is reachable at:

<http://demo.cefril.it:8084/axis2d/services/LPMDeployer?wsdl>.¹⁴

The service has one operation (named deployServiceLPM) used to send the location of the

¹³ As described in [5], at runtime, when the execution of the process reaches the invocation of an external service, a proxy operation is actually called. If the proxy does not refer to any concrete service, it activates the Binder that solve the missing binding. The control is then passed to the proxy that, possibly activating an adapter, invokes the proper operation on the bound service, and then passes the control back to the BPEL execution environment.

¹⁴ At this URI the LPMDeployer Web Service SAWSDL description is available. The SAWSDL is also on the SOA4All iServe Repository.

LPM to the deployer. It is composed by the parameters of input:

- LPMName: required parameter, if only the LPMName is provided the service will look at the default EE repository in order to retrieve the LPM process.
- LPMURI: optional parameter, if the LPMURI is provided the deployer will find the LPM serialized as an XML file at that URI.
- LPMVersion: optional parameter, it is used in case of future extensions of the language for back compatibility.
- LPMFileContent: optional parameter, if the LPMFileContent is provided as an XML file encoded as Base64Binary.
- deploymentOption: optional parameter, for use in case of future extensions of the language for back compatibility.

One sample SOAP Request for the LPMDeployer Service is shown below:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:lpm="http://www.soa4all.eu/execution/services/LPMDeployer">
  <soapenv:Header/>
  <soapenv:Body>
    <lpm:deployServiceLPM>
      <lpm:LPMName>LPMTest</lpm:LPMName>
      <lpm:LPMURI>
        http://www.soa4all.eu/execution/LPM/LPMTest.xml
      </lpm:LPMURI>
    </lpm:deployServiceLPM>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 5 – Example LPMDeployer SOAP Request

The LPMURI parameter refers to the LPM process model URI.

The result of the deployServiceLPM operation is a WSDL/SOAP ready for being invoked (or a SOAP FAULT in case of errors during the deployment). This WSDL/SOAP service, that is the result of the deploy operation execution, exposes at least one operation (e.g. "executeProcess").

3.1.2.2 Consumption Interface

In order to deploy a process the user invokes the executor public WSDL/SOAP interface described in the previous section. The output of this invocation is a WSDL/SOAP ready for being invoked that includes the end-point of the deployed process. The process can be launched through this WSDL/SOAP interface.

Thus, the SOA4All Consumption Platform will invoke the deployed processes using the end-point returned by the deployServiceLPM operation.

In the beginning, using a fresh EE installation, the only available service is the deploy service. After the SOA4All user deploys its first process, the EE will expose the deploy process and the new developed process as a WSDL/SOAP service. When you deploy your second process, the EE will expose 3 services, the deploy service plus the two new WSDLs/services of the two new developed processes. And so on with any new deployed service.

3.1.2.3 Monitoring and Analysis interface

The EE v2 communicates with the SOA4All Monitoring and Analysis Platform through asynchronous events generated by the EE v2.

The events generated are:

Monitoring Events	
ActivityAborted	BusinessActivityUnsuccessfullyFinished
ActivityAssigned	ProcessAborted
ActivityAutomaticallySkipped	ProcessCompleted
ActivityCompleted	ProcessInstantiated
ActivityManuallySkipped	ProcessMonitoringEvent
ActivityReassigned	ProcessResumed
ActivityRelieved	ProcessStarted
ActivityResumed	ProcessSuspended
ActivityScheduled	ProcessTerminated
ActivityStarted	VariableChangedEvent
ActivitySuspended	VariableEvent
ActivityWithdrawn	VariableReadEvent
BusinessActivityClosed	BusinessActivityStarted
BusinessActivityNotStarted	BusinessActivitySucessfullyFinished
BusinessActivityOpen	

Table 4 - Monitoring Events

Details about the events listed in Table 4 are in [2] and [10]. Indeed, the implementation of the EVO publisher component of the EE is done reusing the EVO publisher developed in the FP6 SUPER IP project¹⁵.

The events are sent to the Monitoring Platform through an Active MQ message queue on a topic named: “soa4all.events.execution_engine_v2”.

The Monitoring Platform will be subscribed to this topic in order to consume the events.

3.2 Implementation

3.2.1 Automatic Deployment of LPML models

3.2.1.1 LPML to BPEL transformation

In this section, we describe the advanced prototype of LPML to BPEL transformation as well as the mapping of LPML elements to their BPEL counterparts.

As previously indicated in [6], the transformation of LPML models to BPEL is realised using

¹⁵ <http://www.ip-super.org>

the Intermediate Model (IM) project¹⁶, part of the Eclipse SOA Tools Platform Project¹⁷. The IM uses a SOA meta-model that is well suited to express in a generic form different service-related notions as well as process descriptions. Using transformations to and from the IM, consumers and producers can generate and respectively extract SOA artefacts shared between different platforms and technologies. In SOA4All, we have implemented a transformation from LPML to the IM, as well as reused and updated a transformation from the IM to BPEL. These transformations working in conjunction realise the complete LPML to BPEL transformation. The overall procedure has been outlined in [6] and the rest of this section discusses the current implementation.

The starting point of the transformation is the API defined in D6.3.2, more specifically the *LPMLExporter* interface, implemented by the *BPELExporter* class. The EE Deployer invokes the *exportProcess* method of this class passing it the top-level element of the LPML process to transform (i.e. the LPML *Process* instance). The method generates the BPEL content and returns it to the Deployer.

LPML to IM

The first part of the transformation involves the generation of the IM instance. This is accomplished through a complete 2-phase parsing of the LPML elements referenced by the top-level Process element. The first phase involves the extraction of activities and gateways from the LPML process and creating the corresponding elements in the IM metamodel instance. The second phase involves obtaining the flows present in the LPML process and creating their equivalent entities in the IM instance. The two phases are coordinated through the use of correlated data structures that ensure the consistency of the generated IM model.

IM to BPEL

The second part of the transformation involves the generation of the BPEL process from the IM instance. This transformation reuses existing open-source code available in the Eclipse IM2BPEL plugin¹⁸ and it depends on BPEL-related Eclipse plugins available in the Eclipse BPEL Project¹⁹. This phase is performed through a process consisting of three main steps. First, a BPEL skeleton resource is created as a stream, containing all the basic mandatory information to get an empty executable process. This includes the reference to the client WSDL, an empty list of services participating in this BPEL process aka "PartnerLinks", a list of messages used within the process including "request" and "response" message types, and finally the orchestration logic in a sequence containing the "receive" and "reply" requests. A file is created as well, on which this skeleton stream is associated. Then, the BPEL generation really starts with the IM model on the one hand, and the BPEL skeleton file on the other hand. The IM model is loaded and parsed in order to realise a matching between each of its entities and the corresponding BPEL constructs. Owing to the preparation phase where BPEL-specific structures were put in the IM model during the LPML to IM transformation, the identification of either a basic or a structured BPEL activity in an IM is straightforward. The generation of BPEL 2.0 compliant constructs is facilitated by helper classes provided by a library from the Eclipse BPEL project. Finally, the BPEL skeleton is enriched with all these generated constructs to achieve the complete BPEL process reflecting the model expressed in LPML. This IM to BPEL transformation also includes the generation of the client WSDL file enabling the exposure of the BPEL process as a Web Service.

¹⁶ <http://www.eclipse.org/stp/im/>

¹⁷ <http://www.eclipse.org/stp/>

¹⁸ Contribution under review: https://bugs.eclipse.org/bugs/show_bug.cgi?id=233412

¹⁹ <http://www.eclipse.org/bpel/>

LPML Element	BPEL Element	SOA4All BPEL extensions	Comments
Process	Process		
Activity	Invoke	AdaptiveInvoke	In case and LPM activity has a replacement condition and/or a selection criteria associated it must be mapped on an AdaptiveInvoke.
Parallel Gateway	Flow		+ required Sequence elements
Exclusive Gateway	If		+ required Sequence elements
Flow	Implicit ordering in Sequence		
Service (serviceReference)	Partner Link		+ required corresponding element in the WSDL of the BPEL
Service List		alternativeServiceList	
humanTask		humanTask	
SemanticAnnotation (type=replacementCondition)		replacementCondition	
SemanticAnnotation (type=selectionCriteria)		selectionCriteria	

Table 5 - Mapping of LPML to BPEL Elements

3.2.1.2 Executable artefacts generation process

In this section, we describe the generation of proxies and of the adaptation rules.

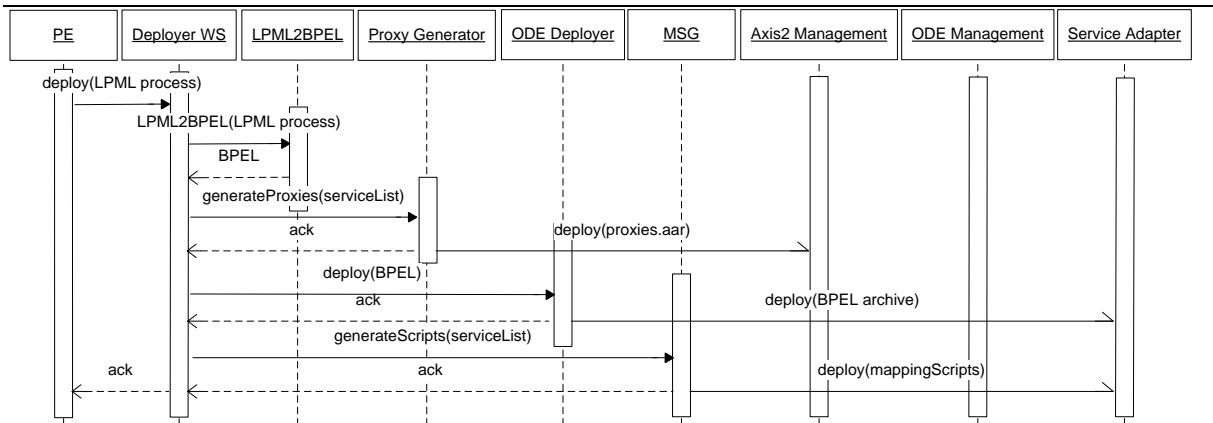


Figure 8 - Sequence Diagram: Automatic Deployment of LPML

The deployment of the LPM process consists of several interrelated activities that make the process ready to be executed. In Figure 8 are shown all the components involved and the relationships between them and the artefacts generated and deployed to the respective component able to process them.

All the components in the figure are included into the EE v2, except for the Process Editor that invokes the EE v2 by using the LPMDeployer Web Service. LPMDeployer is a service with all the capabilities to manage the EE v2 in the deployment phase of the process. It is directly linked to the Deployer WS component that manages and uses the other components involved.

By using the LPML2BPEL transformation (see section 3.2.1.1) the LPM model is translated in BPEL 2.0. The reason is that the EE v2 use Apache ODE as system for executing the process and BPEL 2.0 is the standard specification supported.

After the SOAP Request and the LPML2BPEL transformation, the artefact generated is the SOA4All Extended BPEL 2.0 (for details see [6]) of the process that contains all the information derived from the LPM process models transformed for satisfying the execution purposes.

The Deployer WS component receives the Extended BPEL of the Process from the LPML2BPEL transformation and performs three main steps:

- Proxies generation
- Deployment of the process in Apache ODE
- Mapping script generation and adaptation rules

For each service involved in the process deployed, a specific dedicated proxy is generated at runtime. The proxy exposes a similar interface exposed by the service provider and uses the information extracted by the Extended BPEL in order to allow the dynamic replacement of services and in reaction to contextual situations and to manage situations that requires human tasks. In case of replacement, the proxy is able to select and bind to the right service thanks to the selecting criteria and the list of the possible alternative services provided during the deployment from the process.

The second step is the deployment of the process in Apache ODE, as described in detail in [17] it consists in:

- Creating a temporary service unit directory for the BPEL processes
- Placing the relevant .bpel, .wsdl and .xsd files into the temporary directory
- Creating an ODE deployment descriptor (deploy.xml) and placing it in the temporary

directory

- Zipping up the contents of the temporary directory into a service unit archive
- Creating a temporary service assembly directory
- Placing the service unit archive in a temporary service assembly directory
- Updating the service assembly's jbi.xml descriptor
- Zipping up the contents of the service assembly directory into a service assembly archive
- Copying the service assembly archive into the appropriate deploy directory

The current version of Apache ODE does not support the invocation of RESTful services, but is planned (as anticipated in [17]) to extend the invoke activity to handle RESTful Web Services. The invoke activity replaces the attributes partnerLink/operation with resource and method. The method attribute identifies the HTTP method. All HTTP methods are supported, although this spec is only concerned with GET, POST, PUT and DELETE. The resource attribute identifies a BPEL variable of a simple type (xsd:uri, xsd:string etc.) used as the URL of the actual resource. The resource element can be used instead of the resource attribute to calculate the URL using an expression.

Even if Apache ODE right now is not able to invoke RESTful services, EE v2 is able to support the invocation of RESTful service thanks to the approach described in detail in section 2 and implemented in the EE v2.

The last step is the mapping script generation. For each service present in the process and starting from the list of the possible alternative services it is possible to generate mapping scripts between:

- WSDL/SOAP services;
- RESTful services;
- WSDL/SOAP and RESTful services.

All the mapping scripts generated are available to the service adapter and used in case of need during the execution of the process at runtime. EE v2 could use the SWS Repository introduced in 3.1.1 to retrieve the semantic service descriptions.

3.2.2 Execution and self adaptation

In this section, we describe what happens at runtime.

3.2.2.1 Processes execution and self-adaptation

The deployment of a process described above generates all the artefacts required to execute a process and to provide the self-adaptation at runtime. The artefacts are available inside the EE v2 and the deployed processes are ready for the activation. It is possible to see the list of the processes and to activate them by using the Web Services described at the following web address: <http://demo.cefriel.it:8085/ode/services/listServices>

Using these services, the process of execution starts when the Consumption Platform executes the process and sends the input required for the invocation of the services expected from the process.

In the sequence diagram in Figure 9 it is described a possible scenario of service invocation and rebinding. The activation of the process permits to Apache ODE the execution of the compiled BPEL process selected.

The role of the adapter during the execution is to check and in case of necessity adapt the

request/response during the invocation of a service. Service adapter is the component in the middle between proxies and real services provided by third part.

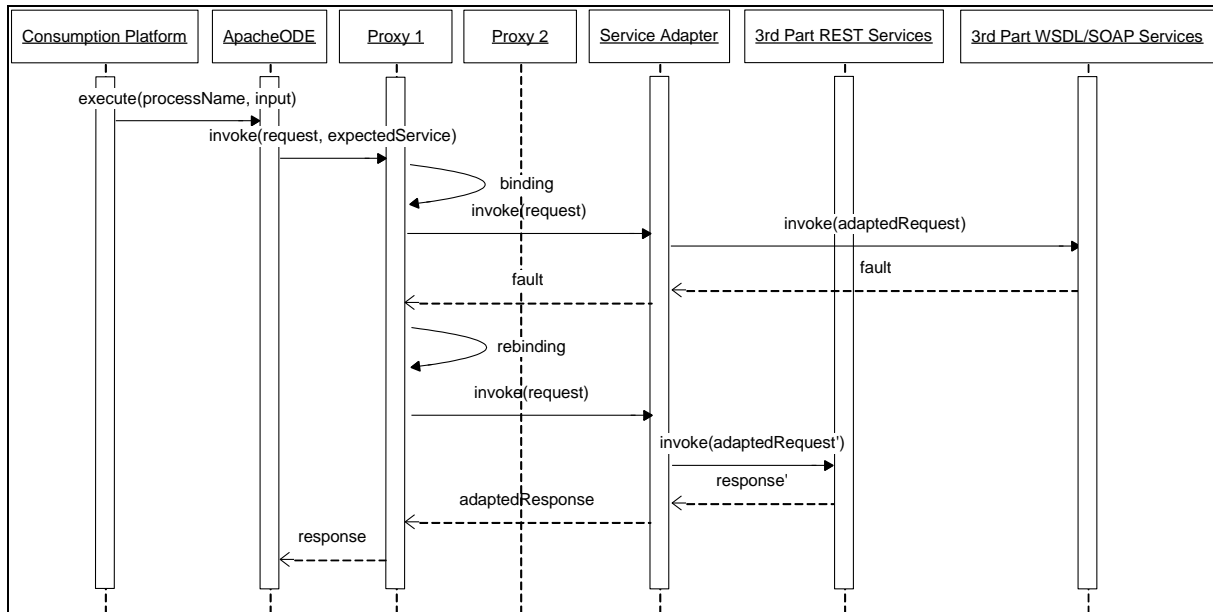


Figure 9 - Execution and self adaptation sequence diagram

During the execution of the process, Apache ODE performs the service invocation to the corresponding Proxy. The proxies bind to the expected service suggested in input and invoke it. As shown in the figure, it is possible that the response of the service invoked is a fault message. In this case by the analysis of the selection criteria and based on the alternative service list Proxy 1 selects the new service to be invoked and invokes it. 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 Proxy1 the request from 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. 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 Proxy 1.

Then, Proxy 1 forwards the response to Apache ODE that continues the execution of the process.

3.2.2.2 WebService Invocation through the SOA4All DSB

The EE can directly invoke WSDL and REST services through their actual addresses/endpoints. In SOA4All some services are exposed on the SOA4All DSB as described in [1]. The EE should be able to send requests for services invocation through the DSB. Some adapter and specific interface have to be developed.

SOA4ALL third party services are accessible for any potential consumer through the SOA4ALL Distribute Service Bus. Within a SOA4ALL process executed by the EE, an activity

requires to be executed by one of those third party services. EE supports direct invocation through the generated proxies, hosted by the Axis 2 engine and generated during the deployment phase, but the invocation through the DSB have additional benefits, as described in detail in [1]:

- optimal communication performance,
- high scalability,
- DSB monitoring.

EE can be modified in order to use the DSB infrastructure to route activity invocation within a running process to the third party service bound to that activity. The approach is shown in Figure 10.

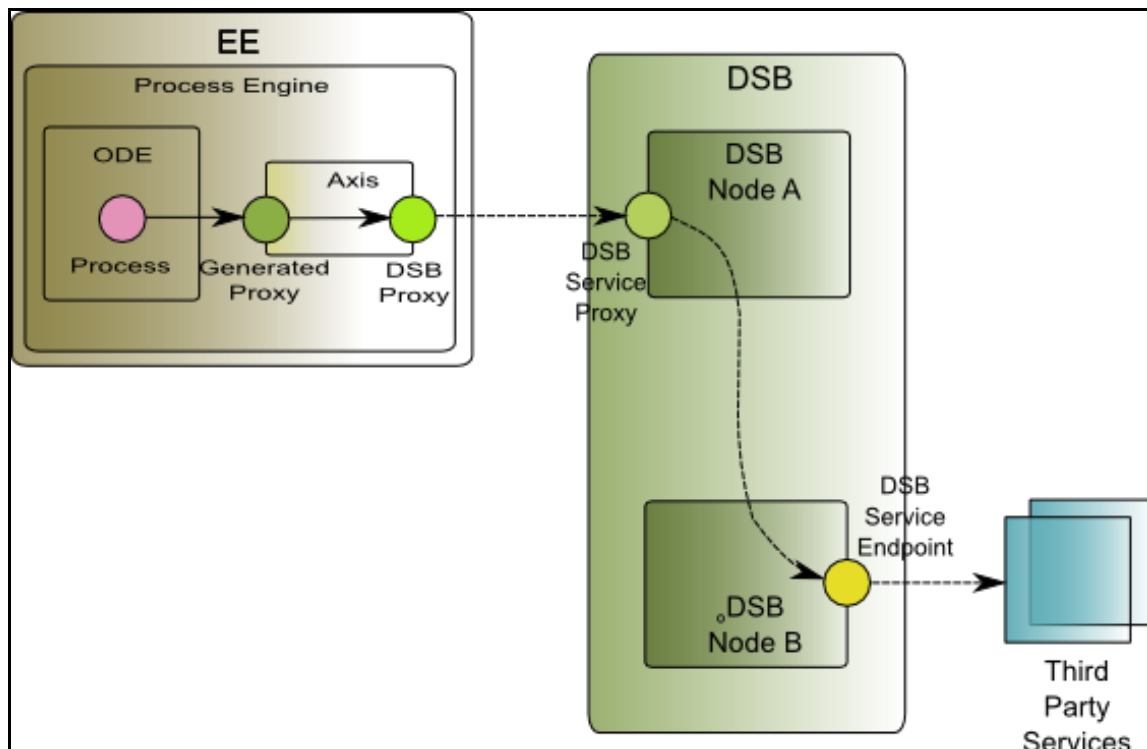


Figure 10 - Integration between EE and the DSB

Each node of a DSB offers a Petals ESB facility named Proxy Component. Currently a REST Proxy Component is available for routing messages to REST third party services. A similar WSDL Proxy Component is under development. This component forwards the incoming message to the DSB endpoint that is closer to the third party service. In turn, this DSB endpoint invokes the actual third party service and routes back the response message to the calling DSB service proxy, which returns the response to the external caller. This approach supports distributed DSB nodes.

In this picture, the caller, which invokes the closer DSB node, is the Execution Environment. Concretely, the generated proxy (hosted by the Axis 2 component and associated to a concrete activity within the running process) forwards the message to another Axis 2 proxy service, the Axis 2 DSB proxy service, which dispatches it to the DSB proxy service of the closer DSB node.

The Axis 2 proxy service mediates between every EE generated proxy and the bound service. The endpoint of the bound service is encoded within the actual message and routed by the Axis 2 DSB proxy service to the DSB Node Service Proxy, which forwards the message to the correct third party service.

Axis 2 only contains one instance of the DSB proxy, which forwards all the messages dispatched by each generated proxy. Axis 2 DSB proxy supports both available DSB Service proxies: REST-proxy and WSDL-proxy.

3.2.2.3 Human tasks invocation

Most of the activities that are to be found in a process are automatic ones. However, there are other activities that must be executed by people. These are known as human tasks. Human tasks are a requirement from the Service Delivery Platform case study (WP7), see 5.2 in [7] and [8].

In order to handle these tasks at a process level, two complementary specifications have been proposed to become standards: BPEL4People [23] and WS-HumanTask [18]. BPEL4People introduces an extension for BPEL that address human interactions. Meanwhile, WS-HumanTask defines the concept of a human task, and the operations to manipulate it. BPEL4People will also support other features such as role-based access control (RBAC), claim or revocation of a specific task, delegation or escalation of tasks, or chained execution.

For the purpose at hand, we need a workflow engine supporting human tasks. Intalio Tempo [14] is a set a components that supports human workflow within a SOA environment. Tempo aims to be fully BPEL4People compliant, but for now, it does not use any BPEL extensions. It actually works as a standalone human task Web Service. It exposes its functionality as both REST and WSDL-based interfaces. Most of the operations defined in WS-HumanTask are already exposed. Apache ODE, the SOA4ALL BPEL orchestration engine of choice, is also supported. A more thorough description of Tempo and how it fits into the human task requirement can be found in [9].

From the WP6 perspective, since the EE is largely based on ODE, a technical decision has been taken to integrate Tempo within the EE. Tempo contains the definition of a specific BPEL process that performs all the necessary steps to execute a human task (possibility of approval, rejection, decisions on deadlines, etc.). Tempo offers this process as an asynchronous service to be invoked.

The human task, being defined as an asynchronous service, needs a correlation ID, in order for the HTS to uniquely identify the appropriate process to call back. The other input parameters that must be sent along with the request are:

- Task Metadata, defines the model for the human task. The possible attributes for this object are:
 - Role Owner, the role that is liable to complete a specific task.
 - User Owner, a specific user to whom this task is assigned
 - Deadline
 - Title
 - Description
 - User Process Endpoint, in this case it is the same as the Execution Engine
- Task Input, other information required in order to complete the task

The scenario to handle human tasks is as follows:

- At design time, a user of the Process Editor defined in WP2 adds a human task activity to her process of choice.
- The user also defines the different input parameters
- The user and roles are managed outside SOA4All within the case studies scenarios

in [11]. Consistency between SOA4All users and case study roles should be maintained.

- The human task is defined as an asynchronous Web Service.
- WP6 offer the means to translate an human task into a BPEL invoke and receive activities to and from the HTS that handle the task object defined above²⁰.

²⁰ At the time of writing this feature is not yet implemented as an automatic translation but it possible to manually define a BPEL able to invoke the HTS for supporting human tasks.

4. Installation and Use

The EE v2 uses Subversion²¹ SOA4All repository to manage its source code and Apache Maven²² as software project management. The component described in Figure 6 and Figure 7 represents the modules of the soa4all-execution-engine part of the SOA4All project that are described in Table 6.

Apache ODE Deployer <ul style="list-style-type: none"> Artifact Id: apache-ode-deployer Group Id: eu.soa4all.execution.ode Packaging: jar 	Binder <ul style="list-style-type: none"> Artifact Id: binder Group Id: eu.soa4all.execution.binder Packaging: jar
Evo Publisher <ul style="list-style-type: none"> Artifact Id: evo-publisher Group Id: eu.soa4all.execution.evo Packaging: jar 	LPM2BPEL <ul style="list-style-type: none"> Artifact Id: lpm2bpel Group Id: eu.soa4all.execution.lpm2bpel Packaging: jar
Mapping Script Generator <ul style="list-style-type: none"> Artifact Id: mapping-script-generator Group Id: eu.soa4all.execution.mapping Packaging: jar 	Proxy Generator <ul style="list-style-type: none"> Artifact Id: proxy-generator Group Id: eu.soa4all.execution.proxygen Packaging: jar
Service Adapter <ul style="list-style-type: none"> Artifact Id: service-adapter Group Id: eu.soa4all.execution.adapter Packaging: jar 	LPM Deployer <ul style="list-style-type: none"> Artifact Id: lpm-deployer Group Id: eu.soa4all.execution.lpm Packaging: aar

Table 6 – EE v2 modules

The organization of the modules follows the standard directory layout for the Maven project. The dependencies between modules and external libraries are described and configured in the POM file. Figure 11 and shows the EE v2 source code tree.

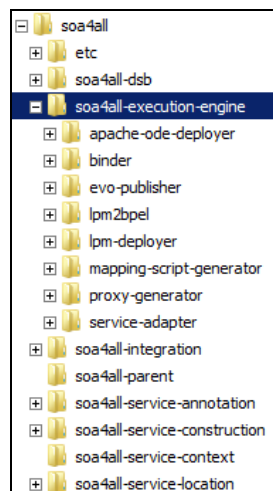


Figure 11 – EE v2 SOA4All source code tree

²¹ <http://subversion.apache.org>

²² <http://maven.apache.org>

All the artefacts generated from the source code must be deployed in Axis2 and Apache ODE configured and located in a specific directory called EE v2.

The EE v2 directory contains for the Windows users a preconfigured installation of Tomcat²³ that contains the software required for the deployment:

- Apache ODE 1.1
- Axis2 1.5.1

A working installation of the EE v2 is reachable in the machine called demo.cefriel.it; the EE v2 is composed of three main components: Deployer, ProcessEngine and Self-Adaptation Framework.

The LPM Deployer (that is released as an axis2 archive that contains LPM2BPEL, Mapping Script Generator, Proxy Generator and Apache ODE Deployer as jar files) depends on the following modules:

- Axis2 1.5.1

For installing the LPM Deployer you will need to install Axis2 on the same machine in which the Execution Engine will be installed and to put the LPMDeployer.aar in the axis2\WEB-INF\services folder.

The ProcessEngine depends on the following modules:

- Axis2 1.5.1,
- Apache ODE 1.1.

For installing the ProcessEngine:

1. Install Apache ODE 1.1;
2. Install Axis2 1.5.1;
3. Put the EVOPublisher.jar in the ode\WEB-INF\lib folder
4. Add the following line in the ode\WEB-INF\conf\ode-axis2.properties
 - a. `ode-axis2.event.listeners= eu.soa4all.execution.evo.notification.EVOPublisher`

The Self-Adaptation Framework is provided with an installer. For installing it is needed to launch the setup.exe provided.

For the development of the EE v2 components in Eclipse it is suggested to use the Maven Eclipse Plugin:

1. Download of the current version of the source code from the SVN Repository and executing the command *“mvn install”*,
2. Execute the *“mvn eclipse:eclipse”* command in the soa4all-execution-engine directory,
3. Add a class path variable in eclipse called *“M2_REPO”* that point to the Local Maven Repository,
4. Open the projects in eclipse.

The installation of the modules share the maven lifecycle phases with the following activities:

- validate: validate the project is correct and all necessary information is available;

²³ <http://tomcat.apache.org>

-
- compile: compile the source code of the project;
 - test: test the compiled source code using a suitable unit testing framework;
 - package: take the compiled code and package it in its distributable format;
 - verify: run any checks to verify the package is valid;
 - install: install the package into the local repository.

5. Conclusions

In the context of the SOA4All project, the Execution Engine, together with the other WP6 components and the Process Engine, should enable different groups of end users to build new services and processes according to their specific needs in a lightweight manner.

In this deliverable, we describe our methodology and approach for adaptive process execution and for the generation of runtime artefacts starting from the modelling language defined in [6].

At the time of writing, the implementation of a second advanced prototype is under way. We will release it at the end of February 2010. In this 2nd prototype we develop:

- A new approach for supporting both WSDL/SOAP services and RESTful services based on SAWSDL and MicroWSMO;
- Automatic deployment of LPML models by translating them into executable processes/services.
- Support for human tasks inside a process.

Furthermore, the EE v2 is already integrated with the Process Editor, with the Consumption Platform and with the Analysis and Monitoring Platform.

During the second year of the project we already developed a demonstrator based on the WP9 scenario. During the third year of the project we plan to develop new demonstrators of the EE v2 based on the use cases of SOA4All.

6. References

- [1] SOA4All Deliverable D1.4.1A SOA4All Reference Architecture Specification
- [2] SOA4All Deliverable D2.3.2 Service Monitoring and Management Tool Suite First Prototype
- [3] SOA4All Deliverable D2.6.2 SOA4All Composer
- [4] SOA4All Deliverable D3.4.3 MicroWSMO and hRESTS
- [5] SOA4All Deliverable D6.5.1 Specification and First Prototype of Composition Framework
- [6] SOA4All Deliverable D6.3.2 Advanced Specification Of Lightweight, Context-aware Process Modelling Language
- [7] SOA4All Deliverable D7.3 End User Service Design.
- [8] SOA4All Deliverable D7.4 End User Service Implementation - First Prototypes
- [9] SOA4All Deliverable D7.5 End User Service Implementation (Final Versions)
- [10] Project IST 026850 SUPER Deliverable 5.7 Semantic Reverse Business Engineering
- [11] SOA4All Deliverable D7.6 End User Service Annotation and Context Descriptions
- [12] M. Colombo, E. Di Nitto, and M. Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In ICSSOC, pages 191–202, 2006
- [13] Cavallaro, L. and Di Nitto, E., “An approach to adapt service requests to actual service interfaces”, in SEAMS '08, Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, New York, NY, USA, ACM Press, 2008, pp. 129–136.
- [14] Cavallaro, L., Ripa, G. and Zuccalà, M., “Adapting Service Requests to Actual Service Interfaces through Semantic Annotations”, in PESOS Workshop, Vancouver, IEEE Computer Society Press, 2009.
- [15] De Giorgio, T., Ripa, G., and Zuccalà, M., “SAWSDL for Self-adaptive Service Composition”, in Beyond SAWSDL 2009, Proceedings of the OTM 2009 workshops, Vilamoura, Springer Press, 2009, pp. 907–916.
- [16] Maleshkova, M., Kopecky, J., Pedrinaci, C., “Adapting SAWSDL for Semantic Annotations of RESTful Services”, in Beyond SAWSDL 2009, Proceedings of the OTM 2009 workshops, Vilamoura, Springer Press, 2009, pp 917–926.
- [17] Apache ODE web site, <http://ode.apache.org>
- [18] Intalio Tempo web site, <http://tempo.intalio.org>
- [19] World Wide Web Consortium (W3C). Semantic Annotations for WSDL and XML Schema, <http://www.w3.org/TR/sawSDL>
- [20] World Wide Web Consortium (W3C). RDFa in XHTML, <http://www.w3.org/TR/rdfa-syntax>
- [21] World Wide Web Consortium (W3C). RDFa Distiller and Parser, <http://www.w3.org/2007/08/pyRdfa>
- [22] WS-HumanTask specification,

<https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/a0c9ce4c-ee02-2a10-4b96-cb205464aa02>

[23] BPEL4People specification:
<https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/30c6f5b5-ef02-2a10-c8b5-cc1147f4d58c>

[24] SOA4All Deliverable D2.6.2 SOA4All Composer

Annex A. RDF extracted from MicroWSMO description

A generic RDF extracted from MicroWSMO service description in RDFa.

```

<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:hr="http://www.wsmo.org/ns/hrests#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:sawSDL="http://www.w3.org/ns/sawSDL#"
  xmlns:wsl="http://www.wsmo.org/ns/wsmo-lite#" xmlns:xhv="http://www.w3.org/1999/xhtml/vocab#"
  xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <!--Service Description URL-->
  <wsl:Service rdf:ID="serviceID">
    <rdfs:isDefinedBy
      rdf:resource="http://www.soa4all.eu/services/ServiceDescriptionPage.htm" />
    <rdfs:label>serviceName</rdfs:label>
    <sawSDL:modelReference rdf:resource="http://www.soa4all.eu/ontologies/ServiceOntology" />
    <!--The name of the operations for each operation-->
    <wsl:hasOperation>
      <wsl:Operation rdf:ID="operationID"><rdfs:label>operationName</rdfs:label>
      <!--The base URI for each operation-->
      <hr:hasAddress rdf:datatype="http://www.wsmo.org/ns/hrests#URITemplate">
        http://www.soa4all.eu/services/ServiceLocation?{-join|&|parameterName1,parameterName2,...}
      </hr:hasAddress>
      <!--The name of the method (i.e. GET, POST), default is GET--><hr:hasMethod>GET</hr:hasMethod>
      <!--A model references for each operation--><sawSDL:modelReference
        rdf:resource="http://www.soa4all.eu/ontologies/ServiceOntology#OperationConcept" />
      <!--The list of the input parameters with the follows information-->
      <wsl:hasInputMessage>
        <wsl:Message rdf:ID="inputParameterID">
          <!--A model reference URI for each parameter-->
          <sawSDL:modelReference
            rdf:resource="http://www.soa4all.eu/ontologies/ServiceOntology#ParameterConcept" />
          <!--(If necessary) a reference to a lifting/lowering Schema for each parameter-->
          <sawSDL:liftingSchemaMapping rdf:resource="http://www.soa4all.eu/ontologies/Lifting" />
          <sawSDL:loweringSchemaMapping rdf:resource="http://www.soa4all.eu/ontologies/Lowering" />
          <!--Parameter type (as XML Schema datatypes)-->
          <sawSDL:modelReference rdf:resource="http://www.w3.org/TR/xmlschema-2/#integer" />
          <!--Parameter name--><rdfs:label>parameterName</rdfs:label>
          <!--Parameter required/optional--><sawSDL:modelReference
            rdf:resource="http://www.soa4all.eu/ontology/execution/parameters#Required" />
          </wsl:Message>
        </wsl:hasInputMessage>
      <wsl:hasInputMessage>
        <wsl:Message rdf:ID="inputParameterID2">
          <!--A model reference URI for each parameter--><sawSDL:modelReference
            rdf:resource="http://www.soa4all.eu/ontologies/ServiceOntology#ParameterConcept" />
          <!--(If necessary) a reference to a lifting/lowering Schema for each parameter-->
          <sawSDL:liftingSchemaMapping rdf:resource="http://www.soa4all.eu/ontologies/Lifting" />
          <sawSDL:loweringSchemaMapping rdf:resource="http://www.soa4all.eu/ontologies/Lowering" />
          <!--Parameter type (as XML Schema datatypes)-->
          <sawSDL:modelReference rdf:resource="http://www.w3.org/TR/xmlschema-2/#integer" />
          <!--Parameter name--><rdfs:label>parameterName</rdfs:label>
          <!--Parameter required/optional--><sawSDL:modelReference
            rdf:resource="http://www.soa4all.eu/ontology/execution/parameters#Required"/>
          </wsl:Message>
        </wsl:hasInputMessage>
      <!--The output parameter-->
      <wsl:hasOutputMessage>
        <wsl:Message>
          <!--Mime type (default text/xml)--><sawSDL:modelReference rdf:resource=".../...#text/xml" />
          <!--If the mime type is text/xml the annotated xml schema--><sawSDL:modelReference
            rdf:resource="http://www.soa4all.eu/serviceID/OperationIDOutputSchema.xsd"/>
          <!--otherwise the model reference URI--><sawSDL:modelReference
            rdf:resource="http://www.soa4all.eu/ontologies/ServiceOntology#ParameterConcept"/>
          <!--(If necessary) a reference to a lifting/lowering Schema-->
          <sawSDL:liftingSchemaMapping rdf:resource="http://www.soa4all.eu/ontologies/Lifting" />
          <sawSDL:loweringSchemaMapping rdf:resource="http://www.soa4all.eu/ontologies/Lowering" />
          </wsl:Message>
        </wsl:hasOutputMessage>
      </wsl:Operation>
    </wsl:hasOperation>
  </wsl:Service>
</rdf:RDF>

```

Listing 6 – Generic MicroWSMO description in RDF

Annex B. RESTful and WSDL/SOAP Service description

MicroWSMO service description based on RDF model of the WeatherForecast RESTful Service.

```

<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:hr="http://www.wsmo.org/ns/hrests#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:sawsdl="http://www.w3.org/ns/sawsdl#"
  xmlns:wsl="http://www.wsmo.org/ns/wsmo-lite#" xmlns:xhvh="http://www.w3.org/1999/xhtml/vocab#"
  xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <wsl:Service rdf:about="#svcl">
    <rdfs:isDefinedBy rdf:resource="http://demo.cefriel.it/rest/services/WeatherForecast.htm" />
    <rdfs:label xml:lang="en">Weather Forecast</rdfs:label>
    <wsl:hasOperation>
      <wsl:Operation rdf:about="#op1">
        <hr:hasAddress rdf:datatype="http://www.wsmo.org/ns/hrests#URITemplate">
          http://demo.cefriel.it/rest/services/WeatherForecast/byLocation?{-join|&|lat,lon}
        </hr:hasAddress>
        <sawsdl:modelReference rdf:resource="http://www.soa4all.eu/ontology/execution/weather#ByLocation"/>
        <rdfs:label xml:lang="en">getForecastByLocation</rdfs:label>
        <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>
        <wsl:hasInputMessage>
          <wsl:Message>
            <sawsdl:modelReference
              rdf:resource="http://www.soa4all.eu/ontology/execution/parameters#Optional"/>
            <sawsdl:modelReference rdf:resource="http://www.w3.org/TR/xmlschema-2/#integer"/>
            <sawsdl:modelReference rdf:resource="http://www.soa4all.eu/ontology/execution/time#Minutes"/>
            <rdfs:label xml:lang="en">min</rdfs:label>
          </wsl:Message>
        </wsl:hasInputMessage>
        <wsl:hasInputMessage>
          <wsl:Message>
            <sawsdl:modelReference
              rdf:resource="http://www.soa4all.eu/ontology/execution/parameters#Optional"/>
            <sawsdl:modelReference rdf:resource="http://www.soa4all.eu/ontology/execution/time#Hours"/>
            <sawsdl:modelReference rdf:resource="http://www.w3.org/TR/xmlschema-2/#integer"/>
            <rdfs:label xml:lang="en">hour</rdfs:label>
          </wsl:Message>
        </wsl:hasInputMessage>
        <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#Longitude"/>
            <rdfs:label xml:lang="en">lon</rdfs:label>
          </wsl:Message>
        </wsl:hasInputMessage>
        <wsl:hasOutputMessage>
          <wsl:Message>
            <sawsdl:modelReference
              rdf:resource="http://www.soa4all.eu/ontology/execution/mimetypes#Text_xml"/>
            <sawsdl:modelReference
              rdf:resource="http://demo.cefriel.it/rest/services/OperationIDOutputSchema.xsd"/>
          </wsl:Message>
        </wsl:hasOutputMessage>
      </wsl:Operation>
    </wsl:hasOperation>
  </wsl:Service>
</rdf:RDF>

```

Listing 7 – WeatherForecast service description in MicroWSMO based on RDF

SAWSDL of the WSDL/SOAP TheWeather Service.

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.soa4all.eu/TheWeather/"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  name="TheWeather" targetNamespace="http://www.soa4all.eu/TheWeather/"
  xmlns:sawsdl="http://www.w3.org/ns/sawsdl/">
  <wsdl:types>
    <xs:schema elementFormDefault="qualified"
      targetNamespace="http://www.soa4all.eu/TheWeather/">
      <xs:element name="getForecast"
        sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/weather#ByLocationReq">
        <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:element minOccurs="1" maxOccurs="1" name="longitude" type="xs:double"
              sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/gps#Longitude" />
            <xs:element minOccurs="1" maxOccurs="1" name="hour" type="xs:int"
              sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/time#Hours" />
            <xs:element minOccurs="1" maxOccurs="1" name="min" type="xs:int"
              sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/time#Minutes" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="getForecastResponse"
        sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/weather#ByLocationRes">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1" name="getForecastResult" type="tns:Forecast" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:complexType name="Forecast"
        sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/weather#Forecast">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="1" name="weather" type="xs:string"
            sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/weather#Description" />
          <xs:element minOccurs="0" maxOccurs="1" name="temp" type="xs:string"
            sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/weather#Temp" />
          <xs:element minOccurs="0" maxOccurs="1" name="wind" type="xs:string"
            sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/weather#Wind"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="getForecastSoapIn">
    <wsdl:part name="parameters" element="tns:getForecast" />
  </wsdl:message>
  <wsdl:message name="getForecastSoapOut">
    <wsdl:part name="parameters" element="tns:getForecastResponse" />
  </wsdl:message>
  <wsdl:portType name="TheWeatherSoap">
    <wsdl:operation name="getForecast">
      <sawsdl:attrExtensions
        sawsdl:modelReference="http://www.soa4all.eu/ontology/execution/weather#ByLocation" />
      <wsdl:input message="tns:getForecastSoapIn" />
      <wsdl:output message="tns:getForecastSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="TheWeatherSoap" type="tns:TheWeatherSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getForecast">
      <soap:operation
        soapAction="http://www.soa4all.eu/ontology/execution/weather#GetForecast" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="TheWeather">
    <wsdl:port name="TheWeatherSoap" binding="tns:TheWeatherSoap">
      <soap:address location="http://demo.cefriel.it/axis2/services/TheWeather" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```

Listing 8 – TheWeather service description in SAWSDL

Annex C. WeatherForecast2TheWeather mapping script

Mapping Script that contains the information required to adapt messages from WeatherForecast to TheWeather service.

```

<?xml version="1.0" encoding="utf-8"?>
<Mapping xmlns:tns="http://www.soa4all.eu/MappingLanguageSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ExpectedNameSpace>http://it.crf.TheWeather/</ExpectedNameSpace>
  <ExpectedEndpoint>http://inrete.dyndns.info/TheWeather/TheWeatherCRF.asmx</ExpectedEndpoint>
  <AvailableDescription>http://demo.cefriel.it/rest/services/WeatherForecast.htm</AvailableDescription>
  <InterfaceRule ID="weather2previsioni">
    <OperationRename>
      <ExpectedService>
        <OperationName>getForecast</OperationName>
      </ExpectedService>
      <AvailableService>
        <URITemplate>
          http://localhost:3000/rest/previsioni/getForecastByLocation?{-join|&|lat,lon}</URITemplate>
        <Method>POST</Method>
      </AvailableService>
    </OperationRename>
  </InterfaceRule>
  <InterfaceRule ID="weather2previsioni_0">
    <ReferenceRuleID>weather2previsioni</ReferenceRuleID>
    <DataTypeRename>
      <Wrapper>
        <ExpectedService>
          <Name>getForecast.latitude</Name>
          <Type>double</Type>
        </ExpectedService>
        <AvailableService>
          <Name>lat</Name>
          <Type>double</Type>
        </AvailableService>
      </Wrapper>
      <Wrapper>
        <ExpectedService>
          <Name>getForecast.longitude</Name>
          <Type>double</Type>
        </ExpectedService>
        <AvailableService>
          <Name>lon</Name>
          <Type>double</Type>
        </AvailableService>
      </Wrapper>
      <Wrapper>
        <ExpectedService>
          <Name>getForecast.hour</Name>
          <Type>int</Type>
        </ExpectedService>
        <AvailableService>
          <Name>hour</Name>
          <Type>int</Type>
        </AvailableService>
      </Wrapper>
      <Wrapper>
        <ExpectedService>
          <Name>getForecast.min</Name>
          <Type>int</Type>
        </ExpectedService>
        <AvailableService>
          <Name>min</Name>
          <Type>int</Type>
        </AvailableService>
      </Wrapper>
    </DataTypeRename>
    <ReturnMappingRuleID>weather2previsioniRet</ReturnMappingRuleID>
  </InterfaceRule>
  <InterfaceRule ID="weather2previsioniRet">
    <DataTypeRename>
      <Wrapper>
        <ExpectedService>
          <Name>getForecastResponse</Name>
          <Type>getForecastResponse</Type>
        </ExpectedService>
        <AvailableService>
          <Name>previsioniMeteo</Name>
          <Type>previsioniMeteo</Type>
        </AvailableService>
      </Wrapper>
    </DataTypeRename>
  </InterfaceRule>
  <ExpectedService>

```



```
<Name>getForecastResponse.getForecastResult.temp</Name>
<Type>string</Type>
</ExpectedService>
<AvailableService>
  <Name>previsioniMeteo.temperatura</Name>
  <Type>string</Type>
</AvailableService>
</Wrapper>
<Wrapper>
  <ExpectedService>
    <Name>getForecastResponse.getForecastResult.weather</Name>
    <Type>string</Type>
  </ExpectedService>
  <AvailableService>
    <Name>previsioniMeteo.previsione</Name>
    <Type>string</Type>
  </AvailableService>
</Wrapper>
</DataTypeRename>
</InterfaceRule>
</Mapping>
```

Listing 9 – WeatherForecast2TheWeather mapping script