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

D6.5.3 Final Prototype For Adaptive Service Composition Execution

Activity N:	2
Work Package:	6
Due Date:	31/08/2010
Submission Date:	31/08/2010
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	CEFRIEL
Revision:	1.0
Author(s):	Gianluca Ripa, Teodoro De Giorgio (CEFRIEL), Fy RAVOAJANAHARY (INRIA)
Reviewers:	Maurilio Zuccalà (CEFRIEL), Yosu Gorroñogoitia (ATOS)

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	08/07/2010	TOC defined	Gianluca Ripa
0.2	14/07/2010	First draft contribution	Teodoro De Giorgio
0.3	15/07/2010	Executive Summary, TOC refined	Gianluca Ripa
0.4	26/07/2010	First contributions	Gianluca Ripa, Teodoro De Giorgio
0.5	31/07/2010	Merged contributions, added Section 1 and Section 5	Gianluca Ripa
0.6	02/08/2010	Transformation from LPML to BPEL	Fy RAVOAJANAHARY
0.7	23/08/2010	Merged contributions, document finalized	Teodoro De Giorgio
0.8	27/08/2010	Internal review	Maurilio Zuccalà
0.9	30/08/2010	Internal review	Yosu Gorroñogoitia
1.0	31/08/2010	Final version	Gianluca Ripa

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms

Acronym	Definition
BPEL	Business Process Executable Language
D	Deliverable
DTC	Design Time Composer
EC	European Commission
EE	Execution Engine
EE v1	Execution Engine version 1
EE v2	Execution Engine version 2
EE v3	Execution Engine version 3
HTS	Human Task Service
IM	Intermediate Model
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
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 Final Prototype for Adaptive Service Composition Execution (EE v3), released at the end of August 2010, is the final prototype including all the planned functionalities as described in [6] and [9]. In the final version the prototype adds no new functionalities: existing ones were enhanced or extended taking into account:

- The feedbacks from the industrial partners of SOA4All in charge of validating the technology through real world use case scenarios;
- The evolution of the Lightweight Process Modeling Language (LPML) [1].

The most relevant improvements in the latest version are:

- **Hybrid processes support**: EE v2 already allowed a process to mix WSDL/SOAP Web services, RESTful Web services and Human tasks realizing a hybrid service composition. This was realized by exploiting semantic annotations as described in [2]. The EE v3 extended the approach thus supporting also services described using both the model reference attribute and the liftingSchemaMapping and loweringSchemaMapping ttributes defined in SAWSDL and MicroWSMO specifications.
- **Support for the latest version of LPML**: in the latest version of the LPML the possibility of specifying the data flow between two activities and the conditions on outgoing flows from gateways as XPATH or SPARQL queries was added. The EE v3 supports the execution of the latest version of LPML.

As for the integration aspects, the EE v2 was ready to be integrated with some of the other SOA4All components (i.e., the **Process Editor**, the **Consumption Platform**, the **Analysis platform**). The integration was completed in the EE v3. The EEv3 is available with its source code through the SOA4All SVN repository.

1. Introduction

1.1 Introductory explanation of the deliverable

The first prototype of the composition framework (EE v1) [5], released on February 2009, 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 [3] and [4].

The Advanced Prototype For Adaptive Service Composition Execution (EE v2), released on February 2010, was the new updated version of EE that was in charge of translating the LPML models into executable processes, exposing them as services and executing them. The EE v2 was extended in order to support REST services invocation and the support for the adaptation of SOAP service requests to REST service interfaces through semantic annotations as described in [2].

This document contains the description of the software release of the EE v3.

1.2 Purpose and Scope

This document aims to describe the Execution Engine v3, with specific emphasis on the realization of some example based on the SOA4All Use Case scenarios.

1.3 Structure of the document

Section 2 briefly recaps the use case requirements and scenarios that drove the design and implementation of the prototype. Section 3 describes the main functionalities of the Execution Engine V3 focusing on the new aspects. Section 4 describes the realization of the use cases scenarios focusing on the execution aspects. Finally, section 5 draws some conclusion.

2. Design, deployment and execution of lightweight processes

This section describes the composition, deployment and execution of hybrid processes involving REST services, SOAP services and Human tasks as required by the three use case scenarios described in the followin section 3.

The lifecycle of the process is depicted in Figure 1.

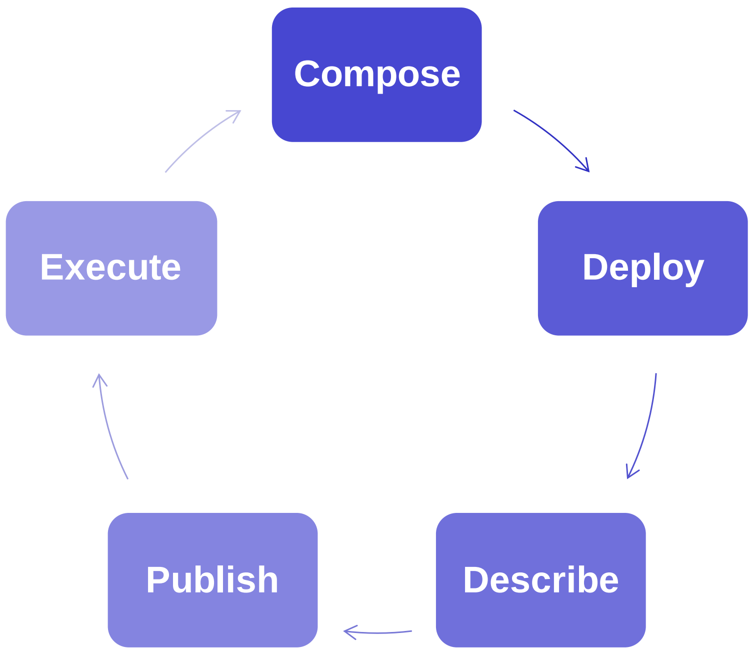


Figure 1 - Lightweight process life cycle

2.1 Compose

At design time, SOA4All offers a complete suite for Service Composition and Adaptation that is also known as Design Time Composition Environment (DTCE), and whose research achievements and technical design have been reported in [13] and [14]. [15] provided a holistic functional description of the final DTCE prototype, intended for end modellers.

SOA4All Service Composition at design time follows a Lightweight Process Modelling methodology and principles that are intrinsically adopted by the Lightweight Process Modelling Language (LPML). LPML is the language intensively used by the DTCE tooling throughout the whole design and deployment life cycle. [10], [11], [12] report the LPML metamodel, methodology and principles. LPML design principles are: a) light semantics as the basis for the description of processes and their elements, b) coarse-grain description of the requirements for process and activities, c) extensive reuse of abstract modelling blocks, such as process fragments and templates, d) contextual adaption, e) semantic enabled data flow, f) multiple activity bindings, etc.

LPML offers a very concise modelling language intended to offer a simple graphical vocabulary to model business process. The SOA4All Studio Process Editor (PE) offers the graphical layout of the language. DTCE tools complement Process Editor, by automating some tedious and cumbersome modelling tasks at design time.

During the design time phase, a modeller produces an optimal executable process model ready for deployment. In this phase, two DTCE tools are involved: Design Time Composer

(DTC) and Optimizer. This phase includes two parts:

- An iterative semi-assisted process-modelling phase, supported by the Process Editor tool and the DTC service. During this phase, the modeller produces a suboptimal executable process model, since she focus on the modelling aspects: work and data flow. DTC service automates some cumbersome modelling tasks, such as binding SWS to activities, expanding unbound activities with matching sub-processes, creating the data flow connectors, checking the semantic I/O compatibility, adapting the process model to contextual information, etc.

- A process-optimization phase, supported by the Process Editor and the Optimizer service. During this phase the modellers optimizes the process model created in the previous phase. Optimizer service replaces the current binding set for each activity within the process with a new one that offers a better global cost function for the process according to both the semantic quality between the I/O of correlated activities and some global KPI specified by the modeller.

The optimized process model is ready for deployment within the SOA4All Execution Environment (EE) using the Deployer service through the Process Editor.

2.2 Deploy

The deployment phase translates the process graphically described using the Process Editor in combination with the DTCE tools, which is not immediately executable, to an Executable Process. Thus, the process is translated into an extended BPEL, defined in [10], through a transformation chain based on the Eclipse project called Mangrove[1]. Leveraging this project allows us to realize a complete transformation from an LPML process to an executable BPEL2.0 process (described in detail in [9]). The serialization purpose is to identify the specificity of each LPML step and generate its BPEL counterparts. That includes basic service call and more conceptual and complex activities such as semantic activities or human tasks. The transformation transports, on the one hand, static attributes:

- Activity name: The name of the activity on the LPML process
- Activity Id: The unique activity identifier
- RoleOwner: The Role of the user which asks for the execution of the process
- Parameters: All the static value which will be used as input on the BPEL process
- Service reference for conversation: The endpoint of the process

On the other hand, it transports dynamic attributes (described in detail in [1] and [10]) 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 execution engine will try to substitute a service with another. Examples of

[1] http://www.eclipse.org/mangrove/

	replacement conditions are: fault and timeout.
selectionCriteria	The selection criteria is the criteria applied by the engine for selecting a substitute service from a list of alternatives. Examples of selection criteria are: price, response time and service rating.
alternativeServiceList	This element lists the services that can be used as alternatives in the service substitution. It is a list of alternativeService subelements: • This element is an URI that points to the service description specification.

Table 1 - Extended BPEL attributes

In detail, an activity defined by the process editor during the composition specifies the execution of some unit of work and will be bound to concrete services. The concrete service is selected by analysing the semantic annotations, and the selectionCriteria.

While the selectionCriteria defines the ranking of services in the service list, the replacementCondition defines when to replace a selected service.

During the deployment phase a lot of tasks are performed automatically for translating the process model in LPML into an executable BPEL document.

1. Human tasks

Human tasks in the LPML are translated into a BPEL fragment as depicted in Figure 2.

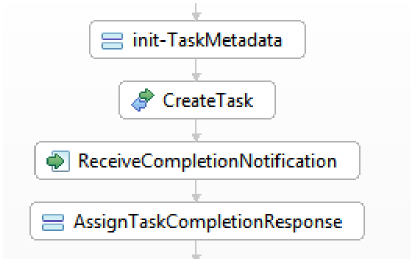


Figure 2 - BPEL fragment for human tasks

2. Activities

Process activities are translated into a BPEL fragment as depicted in Figure 3.

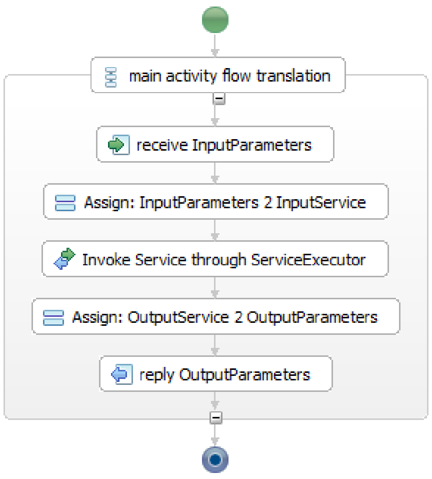

main activity flow translation
receive InputParameters
Assign: InputParameters 2 InputService
Invoke Service through ServiceExecutor
Assign: OutputService 2 OutputParameters
reply OutputParameters

Figure 3 - BPEL fragment for activities

For each activity in the LPML, the following parameters are reported into BPEL variables.

- **serviceRequest**: represents the input of the activity in RDF; the value of this variable is assigned at runtime, and at deploy time it is initialized with an empty value.
- **serviceDescription**: all the information required by the execution engine for the invocation of the service bound by the DTC or by the Optimizer for the execution of the activity:
    - serviceDescriptionURL;
    - serviceName;
    - serviceOperationName;
    - liftingSchemaURL: is the URL to the specifying lifting mappings between semantic data and service data;
    - loweringSchemaURL: is the URL to the specifying lifting mappings between XML and service data and semantic data.
- alternativeServiceList: a list of alternative services for this activity that contains for each alternative service the same elements of the serviceDescription element; this service can be bound to the activity in case a replacementCondition occurs;
- replacementCondition, e.g. fault, no response;
- selectionCriteria, e.g. rating.

3. **Mapping script**

   In case the serviceDescription and/or an alternativeService inside the “alternativeServiceList” does not have a lifting and lowering schema URL, a mapping script is generated in order to allow service substitution at runtime through the service adapter as depicted in Figure 8 and described in [9];

4. **Data flow**

   The mechanisms for managing the data flow inside the LPML, described in [18], [13] and [11], are translated into a standard BPEL assign element. In case of activities bounded to services that manage RDF through the lifting and lowering mechanism the BPEL fragment depicted in Figure 4 is generated.

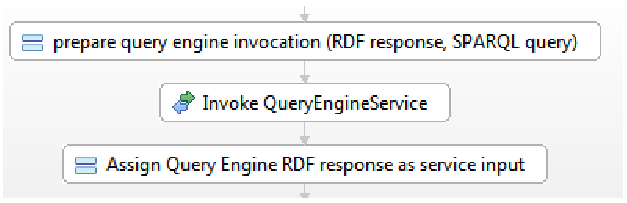

prepare query engine invocation (RDF response, SPARQL query)
Invoke QueryEngineService
Assign Query Engine RDF response as service input

Figure 4 - BPEL fragment for data flow translation

5. Thus, in case of RDF mapping it is required to invoke the QueryEngine SPARQL-based [26] able to understand and manipulate the RDF parameters.**Control flow**

   Standard control flow mechanisms (e.g., gateways, conditions) described in [18] and [12] are translated into a standard BPEL fragment. When services bound to the activities manage RDF as input and output, the conditions are resolved using the SPARQL Query engine. In such settings, each flow in the LPML is annotated with a SPARQL query. The corresponding BPEL fragment generated in this situation is the one depicted in Figure 5.

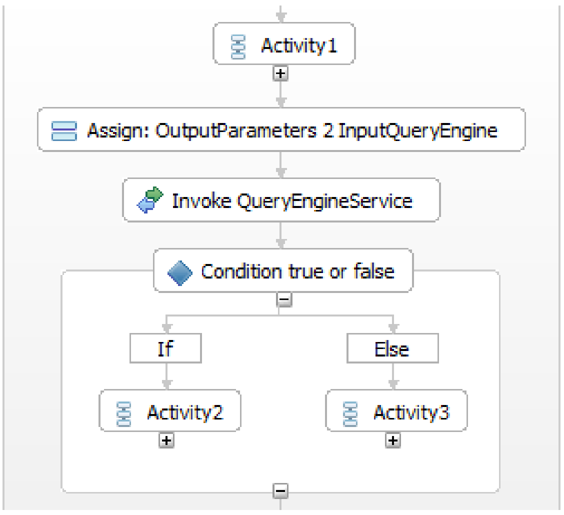

Activity1
Assign: OutputParameters 2 InputQueryEngine
Invoke QueryEngineService
Condition true or false
If
Else
Activity2
Activity3

Figure 5 - BPEL fragment for Control flow condition translation

6. **Loops**

   Another important aspect associated with Data flow is loop, in case of loops in the process by executing a SPARQL Query it is possible to repeat a sequence until the result of the query is false, the input of the query can be both received as input from the process or as output parameters from an activity execution. A typical loop translation is represented in Figure 6.

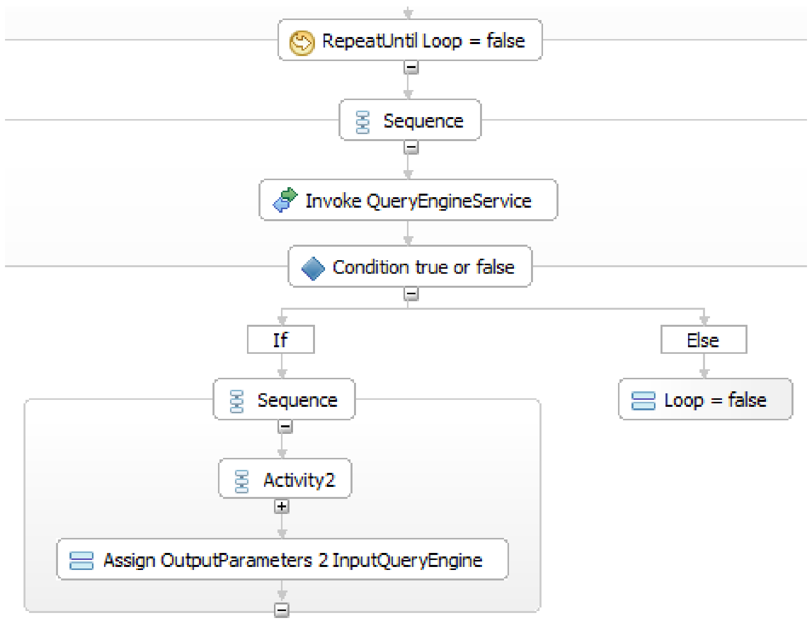


Figure 6 - BPEL fragment for data flow loop translation

The process is finally exposed as service, so the process can be used by invoking it as a normal service by the Consumption Platform through the conceptual layer. Indeed, EE v3 allows to hide the service implementation and to dialogue with a process at an higher level of abstraction by means of semantic descriptions. In order to achieve this goal the modeler has to add further semantic annotation to the WSDL as described in the next section.

2.3 Describe

During the description phase the process interface is described to enable the reuse of the process for other composition in the SOA4All Composer. The deployed lightweight process is encapsulated in a standard WSDL/SOAP interface and available as service. An example of a possible interface of the executable process service is shown in Figure 7. For each process the corresponding WSDL is automatically generated.

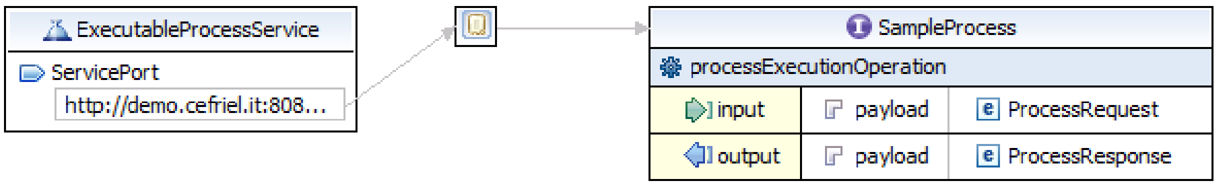


Figure 7 – Executable Process Service interface

Input and output of the service process can be both RDF and XML data types.

In case of RDF, as for every other web service, the WSDL of the process must be annotated with two attributes:

- http://www.w3.org/ns/sawsdl#liftingSchemaMapping for the input parameter;
- http://www.w3.org/ns/sawsdl#loweringSchemaMapping for the output parameter.

The two attributes point respectively to the lifting and lowering mapping URI. The mapping for the input parameter is between the semantic data represented in RDF in the input of the process and the input service parameters of the first activity inside the process. The mapping for the output parameter is between the output service parameters of the last activity inside the process and the semantic data represented in RDF for the output of the process.

In general, the WSDL of the process can be semantically annotated in conformity to the SAWSDL recommendation and it is ready for the publication. Thus each element of the WSDL of the process can be also annotated with a modelReference[2] attribute for facilitating the reuse of the process in other contexts. For this purpose two different SOA4All Studio tools are available:

- the SOWER Editor, that could be used to annotated WSDL and
- the data grounding editor to create lifting and lowering schemas.

2.4 Publish

The Executable Process Service interface, that is a Semantically Annotated WSDL, is published by the modeler on iServe[3], the SOA4All tool in charge of acting as service repository.

2.5 Execute

After the publishing the process is accessible as a Web Service. Through the Consumption Platform it is possible to invoke the executable process. Service selection and replacement at runtime is performed assuming that all implementations of an abstract service have different interfaces or adopt different communication protocols.

2.5.1 Process invocation

The invocation of the services inside the process is provided with an exchange of messages, typically in XML as the traditional SOAP services, but they might also be through invocation of URLs and involving other format of responses such as JSON or other in REST services. To start the execution of the process the Consumption Platform invokes the appropriate operation in the executable process exposed as a service. The Consumption Platform handles input and output data at a conceptual-level and is able to lower and lift to and from invocation-level messages, thanks to the grounding information attached to the services description. Since, as described above, deployed processes are exposed as WSDL services that can be annotated with lifting and lowering schemas, the Consumption Platform is able to invoke the processes as any other service.

2.5.2 Process execution

This section describes what happens during the execution of the executable process described with the extended BPEL described in [11]. The execution can proceed in two different ways depending on the kind of services involved in the process.

The first approach, already described in [2] and [9], is depicted in Figure 8. The service adapter is the component delegated to invoke the services. It receives an abstract service request inside a SOAP message; the request is based on an abstract service interface. Service adapter is able to analyze the request and adapt the abstract request to the concrete service request for performing the service invocation to the appropriate service type and return as the service response of the real service invocation in the adapted SOAP message.

[2] http://www.w3.org/ns/sawsdl#modelReference

[3] http://iserve.kmi.open.ac.uk/

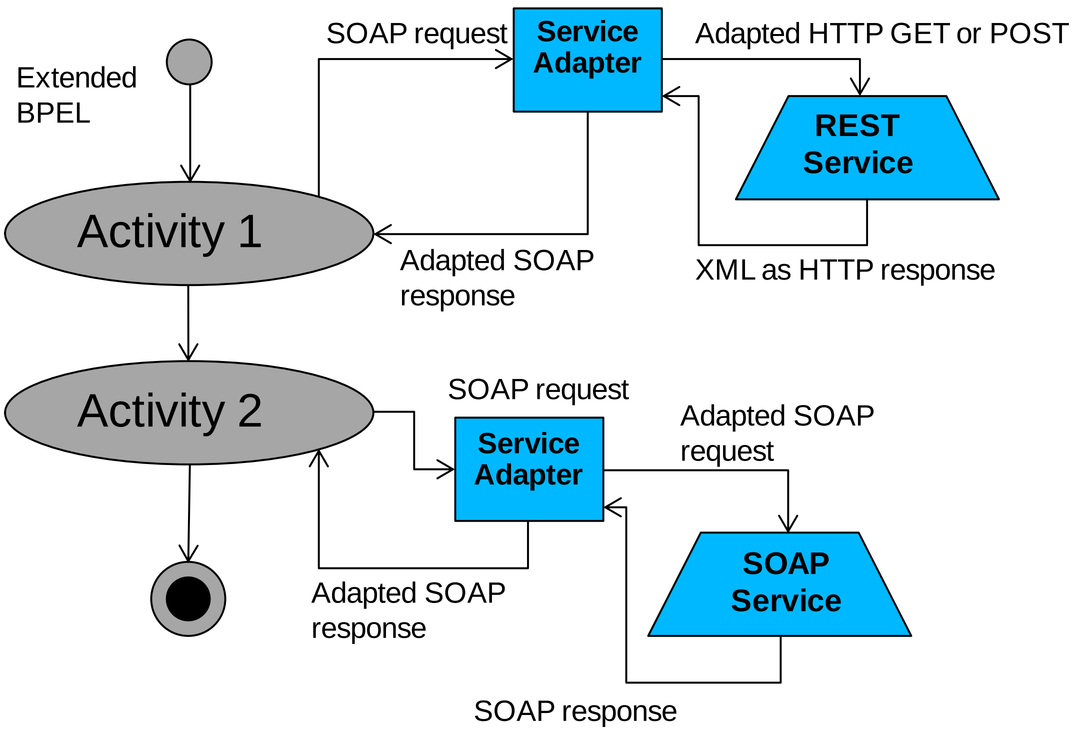


Figure 8 - Hybrid Process Execution with service adapters

Another approach is used when invoked services use the lowering and lifting mechanisms for mapping the input and output of the service from and to the conceptual layer (i.e., from and to RDF triples). In this scenario the SOAP requests bodies always contain RDF triples. The EEv3 is able to manage RDF and, using the lifting/lowering Service, to convert the RDF in a concrete service request and finally to invoke the actual service. Then, after invocation, the EE v3 is able to apply the lifting mechanism and to manage the result of the invocation as RDF.

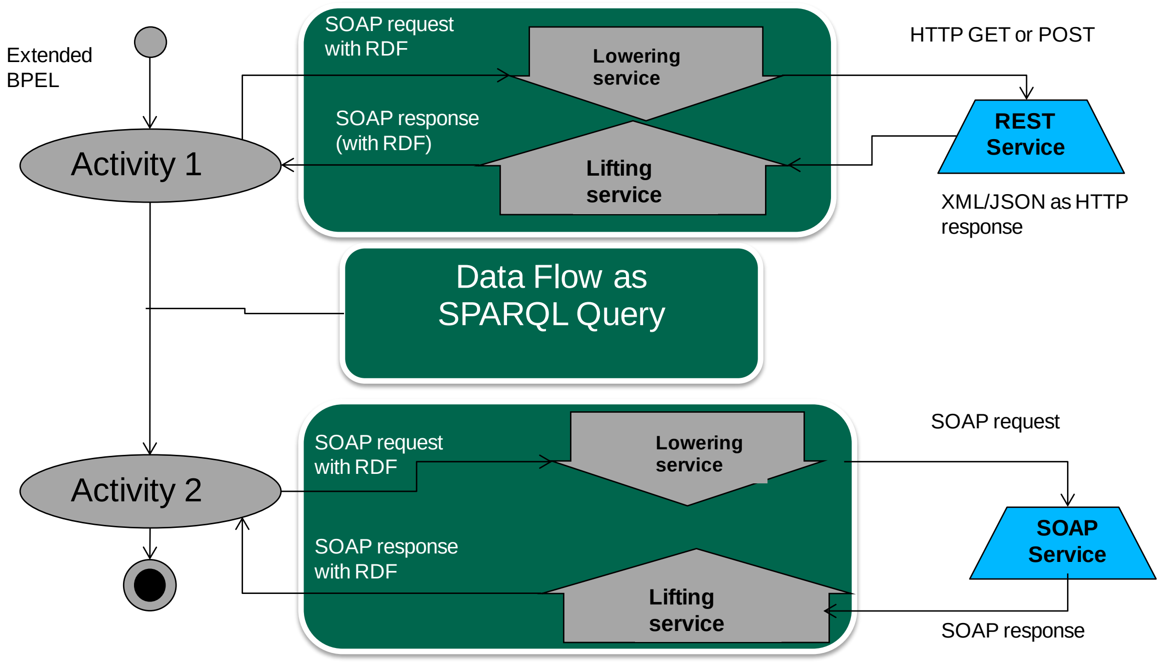
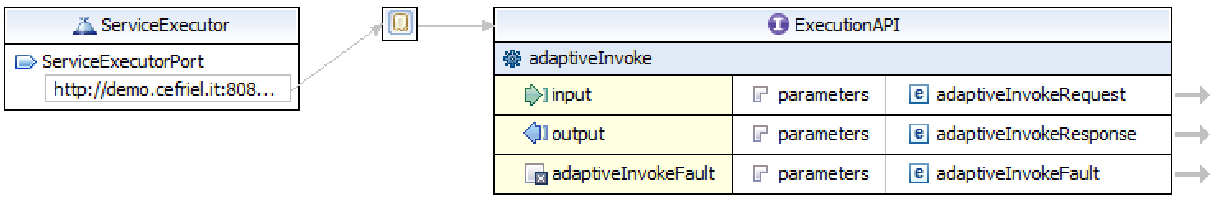


Figure 9 - Hybrid Process Execution with lifting and lowering support

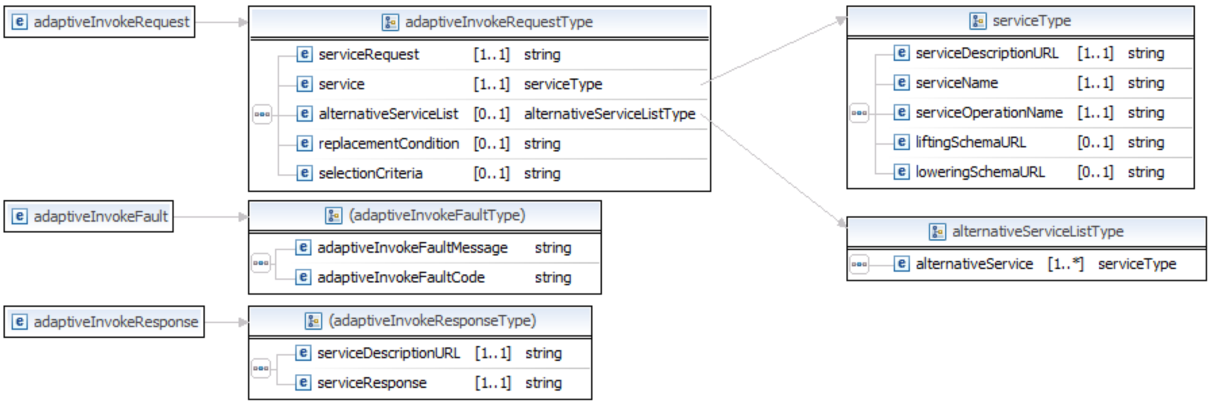
This mechanism is implemented by a service executor service. The WSDL interface of the service executor is described in Figure 10 and in Figure 11.


ServiceExecutor
ServiceExecutorPort
http://demo.cefriel.it:808...

ExecutionAPI
adaptiveInvoke

input	parameters	adaptiveInvokeRequest
output	parameters	adaptiveInvokeResponse
adaptiveInvokeFault	parameters	adaptiveInvokeFault

Figure 10 - Service Executor: adaptiveInvoke operation


adaptiveInvokeRequest → adaptiveInvokeRequestType

serviceRequest	[1..1]	string
service	[1..1]	serviceType
alternativeServiceList	[0..1]	alternativeServiceListType
replacementCondition	[0..1]	string
selectionCriteria	[0..1]	string

serviceType

serviceDescriptionURL	[1..1]	string
serviceName	[1..1]	string
serviceOperationName	[1..1]	string
liftingSchemaURL	[0..1]	string
loweringSchemaURL	[0..1]	string

alternativeServiceListType

alternativeService	[1..*]	serviceType

adaptiveInvokeFault → (adaptiveInvokeFaultType)

adaptiveInvokeFaultMessage	string
adaptiveInvokeFaultCode	string

adaptiveInvokeResponse → (adaptiveInvokeResponseType)

serviceDescriptionURL	[1..1]	string
serviceResponse	[1..1]	string

Figure 11 - Service Executor: adaptiveInvoke operation details

As described in the previous section, EE v3 is able to generate request to the SPARQL Engine by sending some RDF as input and the appropriate SPARQL query (obtained at deploy time from the LPML annotations). Figure 13 shows the execution of the data flow through the SPARQL query engine.

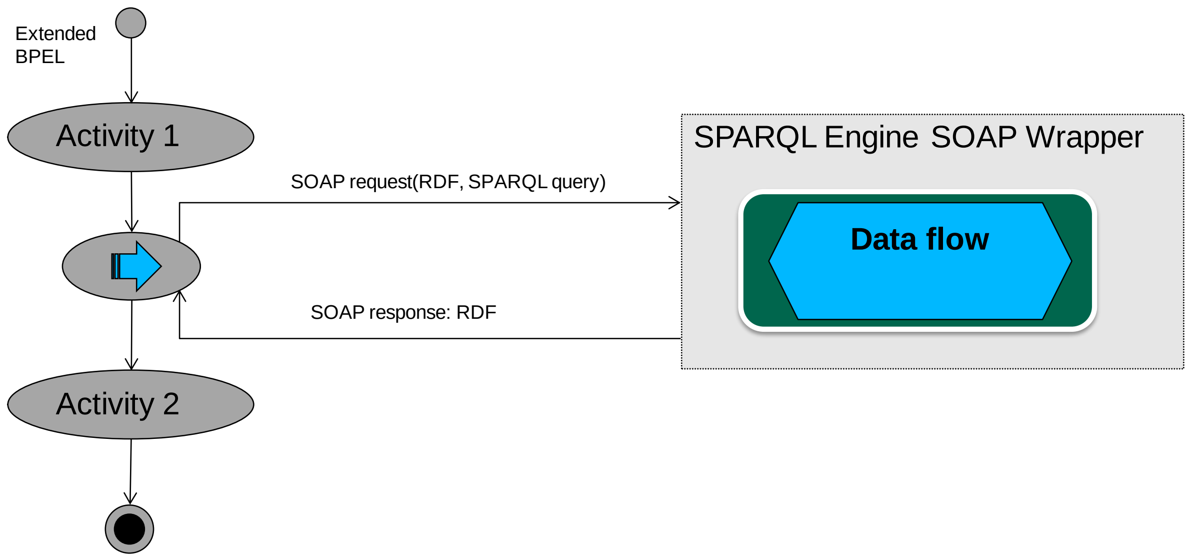


Figure 12 - SPARQL query for data flow

Furthermore, as described in the previous section, the EE v3 provides also a way to evaluate the conditions expressed as SPARQL queries. Figure 13 shows the evaluation of conditions through the SPARQL query engine.

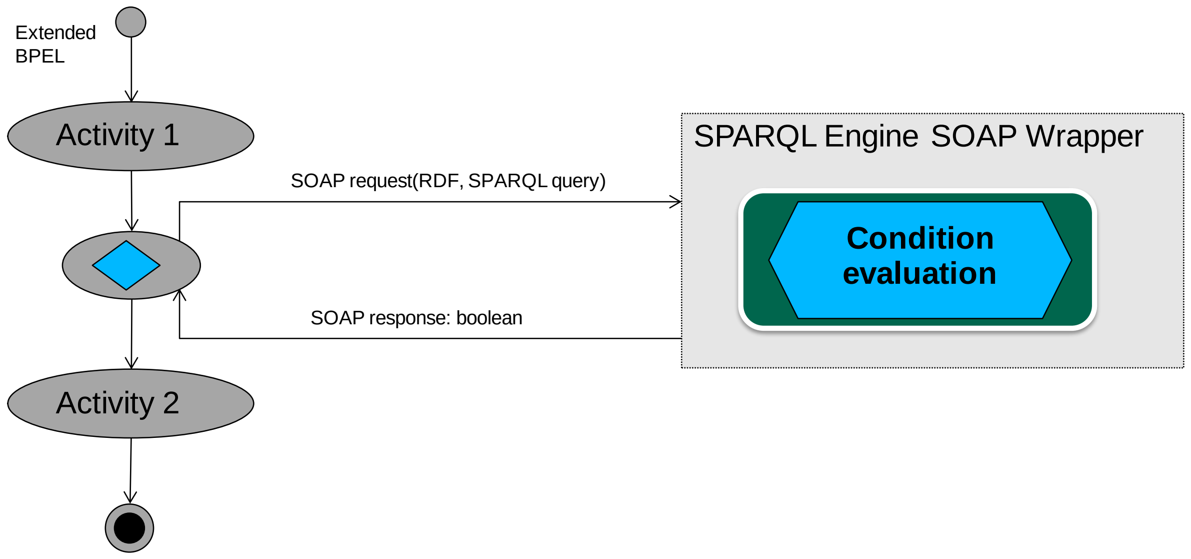


Figure 13 - SPARQL query for condition evaluation

Of course, as defined in the standard BPEL 2.0 specification[4], XPATH expressions are natively supported in EE v3. The modeler can choose which language to use according the kind of services involved in the composition.

2.5.3 Monitor

The execution of the processes makes possible the monitoring of the activities and services involved and the other lightweight processes. EE v3 is integrated with the Analysis Platform by receiving and sending meaningful information (monitoring events). The EVOPublisher

[4] http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf

component, already described in [9], is an extension of the component developed in the SUPER IP project and it is responsible of sending monitoring events to the Analysis Platform.

3. Use case driven experiments and validation

In this chapter we describe some experiment used for validating the EE v3 derived from the use cases proposed by the industrial partners of SOA4All.

3.1 Use case scenarios

Each use case scenario given by the SOA4All industrial partners has very specific requirements as for process design and execution. This section gives a short description of part of the use case requirements and scenarios related to the process execution focusing on the driven the developments of the new version of the EE. In detail:

- The WP7 description focuses on the need for combining SOAP services and human tasks inside a lightweight process [21];
- The WP8 description focuses on REST service support through lifting and lowering schema provided by the service provider as part of the service description and on data flow described by the modeler as SPARQL queries;
- The WP9 description focuses on the process structure (parallel execution).

The three set of requirements, described in more detail in the following sections, added to the ones already described in previous deliverables [8] and [9], propose a lot of challenges as for service execution. The identified solutions, described in Section 2 and 3, are implemented in the EE v3. The implementation of the scenarios is realized following the lightweight process life cycle presented in Section 2.

3.1.1 WP7 requirements and scenarios

WP7 is one of the three SOA4All use cases and has the public sector as its target domain. It envisions an integrated service delivery platform realized based on the technologies and tools developed in SOA4All. It is a lightweight business processes composed of public Web services (hosted by 3rd party service providers), and human activities (to be executed by end users) [10].

The WP7 use case is not completely new since it is described already in [9] with no significant modification. We briefly report here the description of the scenario for the sake of completeness.

WP7 use case includes one scenario that builds an open service platform for the public sector that can be used by the public servants of an administration to model and execute administrative procedures. In [20] they illustrate a typical scenario by a concrete example following the approach of narrative storyboards for describing use cases: the administrative procedure to register a new business at the responsible public administration of the City of X, Germany.

This new business registration process is complex enough to be modeled and executed by civil servants who have no previous expertise on process modeling using SOA technologies. However, the SOA4All approach based on Lightweight Process Modeling Language (see [12]) and accompanying tooling support (see [19], [10]) lowers this barrier imposed to model business processes.

In this scenario, the new business registration process is modeled following an iterative semi-assisted approach, where manual and automatic changes are introduced in the model along each iteration. Moreover, the model intensively reuses modeling fragments (templates, SWSs) retrieved from existing knowledge repositories. Afterwards, the model is optimized, deployed and activated for execution upon demand.

The new registration business process is differentiated from other use case scenarios, not only for its complexity (it uses mostly all the modeling primitives available in LPML), but also because it uses human tasks (see [11], [9]). Human tasks are special process tasks that are performed not by external SWS, but by humans. Typically, a human task receives some input information (documents, forms, links, etc.) that are analyzed by an external reviewer who, upon task completion, returns a report (task acceptance, explanation, fulfilled form, etc).

When a process reaches a human task, it registers it within the Human Task Server (HTS)[5] and blocks the execution until the task is evaluated and validated by an appropriate reviewer. Human tasks are registered for a particular role. Human task reviewers access the Human Task Server through the Human Task Client, which lists pending human task according to the role of the reviewer. The reviewer analyzes each task individually, considering its input information and producing an approval/disapproval report.

3.1.2 WP8 requirements and scenario

The scenario described in [25] is based on a service called Offers4All which could be a service offering of a retail division of a telco such as BT Retail or a non-telco company. For the purposes of the description we will assume that Offers4All is provided by the OfferNet company that in the Telco 2.0 sense can be seen as an upstream customer of BT.

In the scenario, a composition is created allowing one user to contact nearby users by the best means (see Figure 14).

In this scenario the requirements are:

- REST service support: create process and bind REST services to it;
- Loops: take multiple outputs from one service and call other services for each of these;
- Wait: only proceed when a condition is met (could use polling service).

The REST services used in the WP8 scenario are provided with a semantically annotated service description that includes lifting and lowering schemas. Lifting and lowering schemas provided are scripts that can be executed in order to translate a service request or response in an RDF format according to a specific ontology.

[5] Human Task Server is a Intalio Tempo instance. Both Human Task Server and Human Task Client are developed in the context of WP7

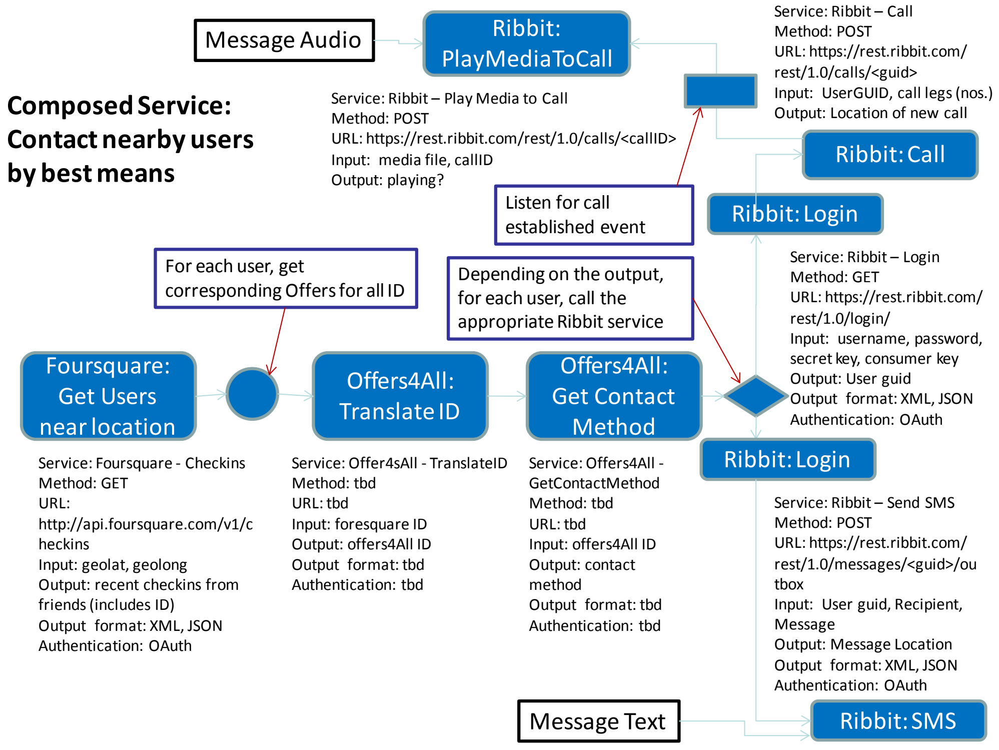

Message Audio
Ribbit: PlayMediaToCall
Service: Ribbit – Call
Method: POST
URL: https://rest.ribbit.com/rest/1.0/calls/<guid>
Input: UserGUID, call legs (nos.)
Output: Location of new call
Composed Service: Contact nearby users by best means
Service: Ribbit – Play Media to Call
Method: POST
URL: https://rest.ribbit.com/rest/1.0/calls/<callID>
Input: media file, callID
Output: playing?
Ribbit: Call
Listen for call established event
Ribbit: Login
For each user, get corresponding Offers for all ID
Depending on the output, for each user, call the appropriate Ribbit service
Service: Ribbit – Login
Method: GET
URL: https://rest.ribbit.com/rest/1.0/login/
Input: username, password, secret key, consumer key
Output: User guid
Output format: XML, JSON
Authentication: OAuth
Foursquare: Get Users near location
Offers4All: Translate ID
Offers4All: Get Contact Method
Ribbit: Login
Service: Foursquare - Checkins
Method: GET
URL: http://api.foursquare.com/v1/checkins
Input: geolat, geolong
Output: recent checkins from friends (includes ID)
Output format: XML, JSON
Authentication: OAuth
Service: Offer4sAll - TranslateID
Method: tbd
URL: tbd
Input: foresquare ID
Output: offers4All ID
Output format: tbd
Authentication: tbd
Service: Offers4All - GetContactMethod
Method: tbd
URL: tbd
Input: offers4All ID
Output: contact method
Output format: tbd
Authentication: tbd
Service: Ribbit – Send SMS
Method: POST
URL: https://rest.ribbit.com/rest/1.0/messages/<guid>/outbox
Input: User guid, Recipient, Message
Output: Message Location
Output format: XML, JSON
Authentication: OAuth
Message Text
Ribbit: SMS

Figure 14 - WP8 Process description

3.1.3 WP9 requirements and scenario

The work package 9 scenario described in detail in [23] and [24] shows how the SOA4All results may be applied in the eCommerce domain. A web shop catalogue is updated from three different sources that provides a service, with different interfaces but similar functionalities, for retrieving a list of products. Thus, the WP9 scenario needs parallel execution flows and synchronization via a gateway for invoking the three services and merge the results in a single list of products. Basically this means to model and execute a process similar to the one depicted in Figure 15.

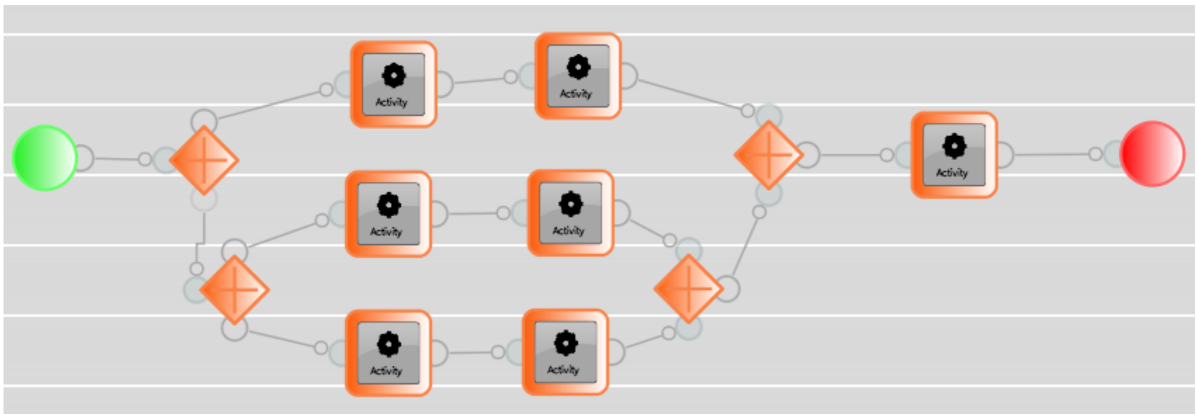


Figure 15 - LPML Process with parallel execution paths

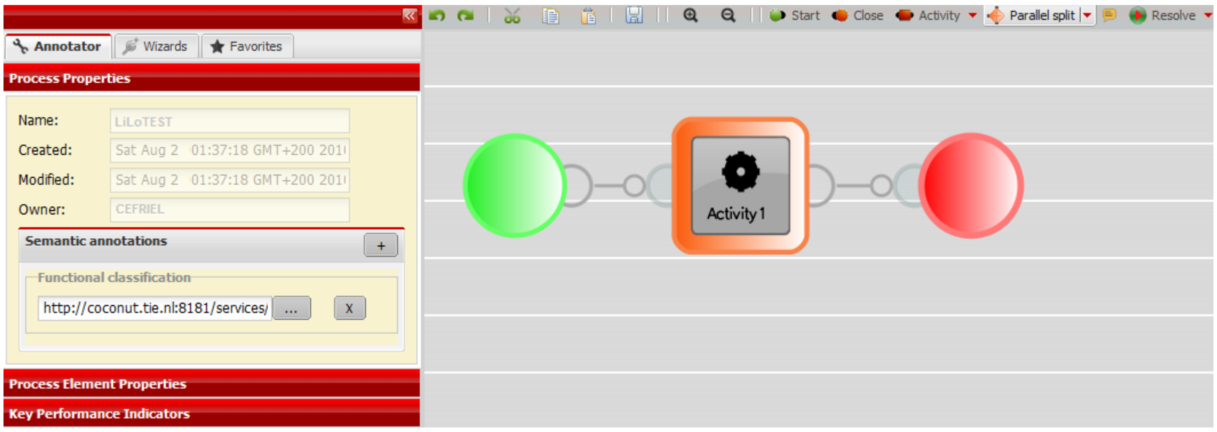
The translation of the diagram should basically be to execute the 3*2 activities in parallel and afterwards to invoke the last activity. Activities in the WP9 scenario are realized using SOAP services.

3.2 Example of Use of the Execution Engine

The lightweight process life cycle presented in section 2 is used in this section for explaining the use cases implementation. The following sections describe the artifacts generated and used by the SOA4All tools in all the phases involved from the composition to the execution of a process focusing on the deployment and execution phase. This sequence of tasks has to be executed for implementing each of the three use case scenarios described above. When present, differences in the implementation between the use case scenarios are highlighted in each section.

3.2.1.1 Compose

The composition phase is described in detail in [15] and [19]. This section describes briefly the case of the creation of a very simple process composed of only one activity. First of all, the user creates the process depicted in Figure 16 using the Process Editor. This process invokes the getProduct operation of ProductWebService[6].



Name: LiLoTEST
Created: Sat Aug 2 01:37:18 GMT+200 201
Modified: Sat Aug 2 01:37:18 GMT+200 201
Owner: CEFRIEL
http://coconut.tie.nl:8181/services/

Figure 16 - LiLoTEST example process

Optionally the Optimizer and the DTC can be used in order to optimize the process and to bind the activity to a different service, respectively.

This step must be performed for each process and thus in all the three use case scenarios implementations.

3.2.1.2 Deploy

Once completed the process model, from the PE, the user invokes the LPM Deployer Service[7] in order to deploy a process in the EE. The PE generates, and LPM Deployer Service receives, a SOAP request similar to the one depicted in Listing 1. In Listing 1 the URL `http://soa4all.eu/execution/lpm/LiLoTEST.lpm` points to a LPM model serialized in XML generated by the PE.

When this request is received a batch procedure starts and it prepares and deploys several different artifacts or it generates an error message for the PE in case of problems in the model. This phase is described in detail in [9], below the description of the artifacts generated during the deploy phase for the LiLoTEST process is reported.

[6] The Product Service is one of the service involved in the WP9 scenario. It is available at the following URL: http://coconut.tie.nl:8181/services/productWS?wsdl

[7] http://demo.cefriel.it:8084/axis2d/services/LPMDeployer?wsdl

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:lpm="http://soa4all.eu/execution/lpm">
  <soapenv:Header/>
  <soapenv:Body>
    <lpm:deployServiceLPM>
      <lpm:LPMName>LiLoTEST</lpm:LPMName>
      <lpm:LPMURI>http://soa4all.eu/execution/lpm/LiLoTEST.lpm</lpm:LPMURI>
    </lpm:deployServiceLPM>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 1 – SOAP Request LPM Deployer Service

The first part of the deployment is the LPM2BPEL translation, during this transformation Activity1 is translated into an Extended BPEL. In Figure 17 is depicted a BPEL fragment generated with the execution of the first activity.

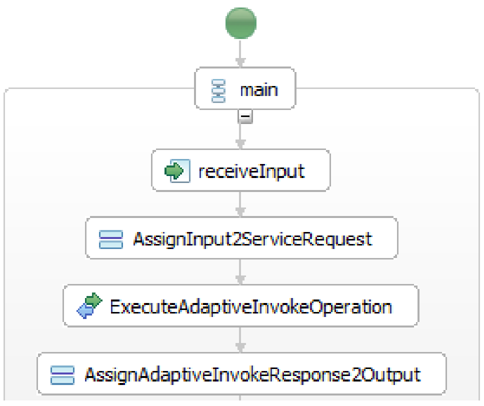

main
receiveInput
AssignInput2ServiceRequest
ExecuteAdaptiveInvokeOperation
AssignAdaptiveInvokeResponse2Output

Figure 17 - LiLoTEST extended BPEL

In case of WP7 scenario the AssignInput2ServiceRequest and the AssignAdaptiveInvokeResponse2Output represent standard XPATH expressions that copy part of an XML message to and from the input and output messages of a service.

In case of WP8 and WP9 scenarios, where REST and SOAP services with lifting and lowering schema annotations in the description are used, the AssignInput2ServiceRequest and the AssignAdaptiveInvokeResponse2Output are XPATH expressions that copy the RDF input and output messages from and to BPEL variables.

At the end of the deployment phase, the process is exposed as a service in the EE v3. All the generated artifacts, depicted in Figure 18, are deployed in the in the appropriate location in the Apache Ode installation that is part of the SOA4All EE.

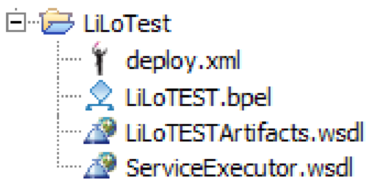


Figure 18 – Generated artifacts for the "LILOTest" process

After this operation the Process is available at its endpoint. At the following URL, a list of all the processes deployed in the EE deployed at the CEFRIEL premises is available:

- http://demo.cefriel.it:8085/ode/services/listServices address.

3.2.1.3 Describe

Supported by the SOWER Editor[8] the users can add semantic annotations to the WSDL process interface description. In case of WP7, at least, the user has to annotate the input and output operation parameters of the WSDL service with the lowering and lifting schema.

```
<xsd:element name="ProcessResponse"
sawsdl:loweringSchemaMapping="http://coconut.tie.nl:8080/storage/repositories/eCommerce_Ground
ing_V2/files/productws_lowering_mapping_schema.xsl"
             xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
  <xsd:complexType>
    <xsd:sequence>
<xsd:element minOccurs="1" name="serviceDescriptionURL" type="xsd:string"/>
<xsd:element minOccurs="1" name="serviceResponse" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ProcessRequest"
sawsdl:liftingSchemaMapping="http://coconut.tie.nl:8080/storage/repositories/eCommerce_Groundi
ng_V2/files/productws_lifting_mapping_Schema.xsl"
             xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="input" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Listing 2 – Semantic Annotations of “LiLoTEST” process description

In case of WP8 and WP9 the lifting and lowering schemas are empty since the input of the first service and the output of the last service invoked in the process are available as RDF.

3.2.1.4 Publish

At this stage the service description can be published on iServe at the following URL:

- http://iserve.kmi.open.ac.uk/browser.html.

3.2.1.5 Execute

In order to execute the process the user has to invoke a service published in iServe. The user is not necessarily aware that he is invoking a process since it is not different from any other service. The following endpoint can be retrieved from iServe and used in order to execute the LiLoTEST process:

- http://demo.cefriel.it:8085/ode/services/LiLoTEST?wsdl

An example of SOAP Request for the invocation is reported in Listing 3. When the EE receives a request in the format depicted in Listing 3, a process instance is started and during its execution some service has to be invoked. During the execution of the LILOTest process, a real service involved in the WP9 scenario is invoked. Listing 4 shows the SOAP request generated by the EE.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
```

[8] A SOA4All tool that is part of the SOA4All Studio.

```
                  xmlns:soa4="http://soa4all.eu/execution/LiloTEST">
  <soapenv:Header/>
  <soapenv:Body>
    <soa4:LiLoTESTRequest>
      <soa4:input>
        <![CDATA[<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oms="http://ecommerce.soa4all.eu/products/v1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:about="http://anonymous.generated/iOnto#_omsProduct3211ScullerJeans90">
    <oms:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">ScullerJeans</oms:name>
    <rdf:type rdf:resource="http://ecommerce.soa4all.eu/products/v1/Product"/>
    <oms:price rdf:datatype="http://www.w3.org/2001/XMLSchema#string">90</oms:price>
    <oms:id rdf:datatype="http://www.w3.org/2001/XMLSchema#int">32</oms:id>
  </rdf:Description>
  <rdf:Description rdf:about="http://anonymous.generated/iOnto">
    <owl:imports rdf:resource="http://ecommerce.soa4all.eu/products/v1/"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
  </rdf:Description>
</rdf:RDF>]]>
      </soa4:input>
    </soa4:LiLoTESTRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 3 - LiLoTEST Process SOAP Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:exec="http://soa4all.eu/execution/executor">
  <soapenv:Header/>
  <soapenv:Body>
    <exec:adaptiveInvokeRequest>
      <exec:serviceRequest>
<![CDATA[<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oms="http://ecommerce.soa4all.eu/products/v1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:about="http://anonymous.generated/iOnto#_omsProduct3211ScullerJeans90">
    <oms:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">ScullerJeans</oms:name>
    <rdf:type rdf:resource="http://ecommerce.soa4all.eu/products/v1/Product"/>
    <oms:price rdf:datatype="http://www.w3.org/2001/XMLSchema#string">90</oms:price>
    <oms:id rdf:datatype="http://www.w3.org/2001/XMLSchema#int">32</oms:id>
  </rdf:Description>
  <rdf:Description rdf:about="http://anonymous.generated/iOnto">
    <owl:imports rdf:resource="http://ecommerce.soa4all.eu/products/v1/"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
  </rdf:Description>
</rdf:RDF>]]>
      </exec:serviceRequest>
      <exec:service>
        <exec:serviceDescriptionURL>
          http://coconut.tie.nl:8181/services/productWS?wsdl
        </exec:serviceDescriptionURL>
        <exec:serviceName>ProductWebServiceService</exec:serviceName>
        <exec:serviceOperationName>getProduct</exec:serviceOperationName>
        <exec:liftingSchemaURL>

http://coconut.tie.nl:8080/storage/repositories/eCommerce_Grounding_V2/files/productws_lifting
_mapping_Schema.xsl
        </exec:liftingSchemaURL>
        <exec:loweringSchemaURL>

http://coconut.tie.nl:8080/storage/repositories/eCommerce_Grounding_V2/files/productws_lowerin
g_mapping_schema.xsl
```

```
        </exec:loweringSchemaURL>
      </exec:service>
    </exec:adaptiveInvokeRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 4 – Service Executor SOAP Request

This SOAP request is interpreted by the Service Executor that lowers the input from RDF to XML and invokes the Product Web Service[9]. The Product web service response is depicted in Listing 5. And finally the LiLoTEST process response is depicted in Listing 6.

As already described a lot of complex mechanisms can be added to the process in order to allow runtime service substitution, condition evaluation, loops and data flow between activities.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns1:adaptiveInvokeResponse xmlns:ns1="http://soa4all.eu/execution/executor">

<ns1:serviceDescriptionURL>http://coconut.tie.nl:8181/services/productWS?wsdl</ns1:serviceDesc
riptionURL>
      <ns1:serviceResponse>
        <![CDATA[<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oms="http://ecommerce.soa4all.eu/products/v1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <owl:Ontology rdf:about="http://anonymous.generated/iOnto">
    <owl:imports rdf:resource="http://ecommerce.soa4all.eu/products/v1/"/>
  </owl:Ontology>
  <oms:Product rdf:about="http://anonymous.generated/iOnto#_omsProduct3216IBMLaptop1300.0">
    <oms:id rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >3216</oms:id>
    <oms:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >IBMLaptop</oms:name>
    <oms:price rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >1300.0</oms:price>
  </oms:Product>
</rdf:RDF>]]>
      </ns1:serviceResponse>
    </ns1:adaptiveInvokeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 5 – Service Executor SOAP Response

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:soa4="http://soa4all.eu/execution/LiloTEST">
  <soapenv:Body>
    <soa4:LiLoTESTResponse>
      <soa4:output>
        <ns1:adaptiveInvokeResponse xmlns:ns1="http://soa4all.eu/execution/executor">

<ns1:serviceDescriptionURL>http://coconut.tie.nl:8181/services/productWS?wsdl</ns1:serviceDesc
riptionURL>
          <ns1:serviceResponse>
<![CDATA[<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oms="http://ecommerce.soa4all.eu/products/v1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
```

[9] http://coconut.tie.nl:8181/services/productWS?wsdl

```
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <owl:Ontology rdf:about="http://anonymous.generated/iOnto">
    <owl:imports rdf:resource="http://ecommerce.soa4all.eu/products/v1/"/>
  </owl:Ontology>
  <oms:Product rdf:about="http://anonymous.generated/iOnto#_omsProduct3216IBMLaptop1300.0">
    <oms:id rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >3216</oms:id>
    <oms:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >IBMLaptop</oms:name>
    <oms:price rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >1300.0</oms:price>
  </oms:Product>
</rdf:RDF>]]>
          </ns1:serviceResponse>
        </ns1:adaptiveInvokeResponse>
      </soa4:output>
    </soa4:LiLoTESTResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 6 - LiLoTEST Process SOAP Response

4. Conclusions

This document describes the Final Prototype for Adaptive Service Composition Execution (EE v3). It highlights the main facets of the EE v3 focusing on the novelties of the last version and it describes the lifecycle of a process from the design to the execution phase. It focuses mainly on the deploy and execution phase since the composition is already described in detail in [15]. The EE v3 is part of the SOA4All Service Construction Suite that includes also the tools used at design time for the creation of the process models.

The EE v3 implements all the functionalities needed for executing processes described in LPML and for satisfying the requirements of all the three SOA4All use cases:

- support for human tasks;
- support for WSDL/SOAP and REST services exploiting both model reference annotations and lifting and lowering schema annotations;
- support for data handling inside a process using both XML data types and RDF;
- support for gateways, conditions and loops as using both XPATH and SPARQL.

5. References

[1] Schnabel, F., Gorroñogoitia, Y., Radzimski, M., Lecue, F., Mehandjiev, N., Ripa, G., Abels, S., Blood, S., Mos, A., Junghans, M., Agarwal, S., Vogel, J., “Empowering Business Users to Model and Execute Business Processes”, to appear in BPMS2’10, Hoboken, 2010.

[2] De Giorgio, T., Ripa, G., and Zuccalà, M., “An Approach to Enable Replacement of SOAP Services and REST Services in Lightweight Processes”, to appear in Composable Web 2010, Proceedings of the ICWE 2010 workshops, Vienna, Springer Press, 2010.

[3] 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.

[4] 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.

[5] 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.

[6] Maleshkova, M., Kopecky and 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,

[7] Kopecky, J., Vitvar, T., Bournez, C. and Farrell, J., “SAWSDL: Semantic Annotations for WSDL and XML Schema”, IEEE Internet Computing, 2007

[8] SOA4All Deliverable - D6.5.1 Specification and First Prototype of Composition Framework

[9] SOA4All Deliverable - D6.5.2 Advanced Prototype For Adaptive Service Composition Execution

[10] SOA4All Deliverable - D6.3.1. First Specification of Lightweight Process Modelling Language

[11] SOA4All Deliverable - D6.3.2 Advanced Specification Of Lightweight, Context-aware Process Modelling Language

[12] SOA4All Deliverable - D6.3.3 Final Design of the Lightweight Context-aware Process Modelling Language

[13] SOA4All Deliverable - D6.4.1 Specification and First Prototype Of Service Composition and Adaptation Environment

[14] SOA4All Deliverable - D6.4.2 Advanced Prototype For Service Composition and Adaptation Environment

[15] SOA4All Deliverable - D6.4.3 Final Prototype For Service Composition and Adaptation Environment

[16] SOA4All Deliverable - D1.3.3A A Distributed Semantic Marketplace

[17] SOA4All Deliverable - D2.2.2 Service Consumption Platform First Prototype

[18] SOA4All Deliverable - D2.6.2 SOA4All Composer First Prototype

[19] SOA4All Deliverable - D2.6.3 SOA4All Composer Second Prototype

[20] SOA4All Deliverable - D7.2 Scenario Definition

[21] SOA4All Deliverable - D7.4 End User Service Implementation (First Prototypes)

[22] SOA4All Deliverable - D7.6 End User Service Annotation and Context Descriptions

[23] SOA4All Deliverable - D9.2.1 eCommerce framework infrastructure Design

[24] SOA4All Deliverable - D9.3.1 C2C e-Commerce Prototype v1

[25] SOA4All Deliverable - D10.1.2 Business Scenarios and Models v2

[26] SPARQL Query Language for RDF - http://www.w3.org/TR/rdf-sparql-query/

[27] Resource Description Framework (RDF) - http://www.w3.org/RDF/

[28] W3C: Semantic Annotations for WSDL and XML Schema. W3C Recommendation (2007), http://www.w3.org/TR/sawsdl/

Annex A. Technical Annex

This section summarizes the technical changes and improvements, mostly related to the integration with other platform services and API provided by other technical WPs. This section also directs the user to the source code and installation instructions of the EE v3 components.

The EE v3 as the previous version uses the Subversion[10] SOA4All repository to manage its source code and Apache Maven[11] as software project management. Software architecture and components are described in [9] and represent the modules of the soa4all-execution-engine part of the SOA4All project.

The SOA4All EE v3 internal components are available as library. It is possible to create the jar file from the project sources following the procedure described in [9].

- **LPM2BPEL**: translate the LPM process description in LPML into the extended BPEL.

  https://svn.sti2.at/soa4all/trunk/soa4all-service-construction/LPML-BPEL/

- **Proxy Generator**: generates the java code needed at runtime for implementing the runtime adaptation capabilities of the EE.

  https://svn.sti2.at/soa4all/trunk/soa4all-execution-engine/proxy-generator/

- **Mapping Script Generator**: generate mapping scripts for enabling the service adaptation between services (SOAP and REST) during the service replacement.

  https://svn.sti2.at/soa4all/trunk/soa4all-execution-engine/mapping-script-generator-rest/

- **Apache ODE Deployer**: generate the artefact required for the publishing of the process as service. It is part of the LPM Deployer service.

  https://svn.sti2.at/soa4all/trunk/soa4all-execution-engine/lpm-deployer/

- **Evo Publisher**: component responsible of sending monitoring events generated at runtime to the Analysis Platform.

  https://svn.sti2.at/soa4all/trunk/soa4all-execution-engine/evo-publisher/

- **Service Adapter**: syntactic invocation of SOAP and REST Services adapt service messages during the service replacement at runtime using mapping script

  https://svn.sti2.at/soa4all/trunk/soa4all-execution-engine/service-adapter/

For the integration of the EE v3 with the SOA4All platform all these component are used in the different phases of the process lifecycle through different web services, the first two are available online and are:

- **Service Executor SOAP Service**
  - Description: invoke SOAP and REST services with input and output parameters in RDF or data type.
  - Components dependencies:

[10] http://subversion.apache.org

[11] http://maven.apache.org

        - Service Adapter
    - Source code:
      https://svn.sti2.at/soa4all/trunk/soa4all-execution-engine/service-executor/
    - Service endpoint (SAWSDL):
      http://demo.cefriel.it:8084/axis2d/services/ServiceExecutor?wsdl
- **LPM Deployer SOAP Service**
    - Description: public service that represent the interface of the EE v3 for the deployment of processes described in LPM.
    - Components dependencies:
        - LPM2BPEL
        - Proxy Generator
        - Mapping Script Generator
        - Apache ODE Deployer
        - Evo Publisher
    - Source code:
      https://svn.sti2.at/soa4all/trunk/soa4all-execution-engine/lpm-deployer/
    - Service endpoint (SAWSDL):
      http://demo.cefriel.it:8084/axis2d/services/LPMDeployer?wsdl
    -

http://demo.cefriel.it:8085/ode/services/listServices

EE v3 includes many features and technical improvements in all the listed components and services for M30:

- WP7 WP8 WP9 use case requirement and scenarios support;
- Support for the latest version of LPML;
- Integration with latest Process Editor, Consumption Platform and Analysis platform;
- Lifting and lowering support for SOAP and REST Services;
- Invocation of services with grounding component;
- SPARQL Query execution through the SPARQL Engine;
- Conditions on gateways and Loops support;
- RDF parameters mapping in Data flow.

EE v3 includes many features and technical improvements in all the listed components and services for M30:

- WP7 WP8 WP9 use case requirement and scenarios support;
- Support for the latest version of LPML;

- Integration with latest Process Editor, Consumption Platform and Analysis platform;
- Lifting and lowering support for SOAP and REST Services;
- Invocation of services with grounding component;
- SPARQL Query execution through the SPARQL Engine;
- Conditions on gateways and Loops support;
- RDF parameters mapping in Data flow.