Bridging The Semantic Gap Between Business Processes and Semantic Web Services

Muhammad Ahtisham Aslam [1],Sören Auer [1,2], Jun Shen [3] , Klaus-Peter Fähnrich [1]
[1] Betriebliche Informationsysteme
University of Leipzig, Germany
[2] Computer and Information Science Department
University of Pensylvania, USA
[3] School of Information Systems and Technology
University of Wollongong, Australia
{aslam,auer,faehnrich}@informatik.uni-leipzig.de, auer@seas.upenn.edu, jshen@uow.edu.au

Abstract

Bridging the semantic gap between business process models and semantic Web services becomes increasingly important in order to help automating business process integration in large organizations. Traditional workflow languages (such as BPEL4WS) support the modeling of business processes as syntax based compositions of Web services. When such processes are exported as Web services they as well expose syntactical interfaces. These syntactical interfaces allow only static composition and hence limit interactions between business partners. The obstacles of syntax based integration and composition can be addressed by enhancing business processes with semantics. This enables us to 1) edit and model the compositions of Web services on the basis of matching semantics 2) provide semantically enriched descriptions of business processes. In particular, it will support the dynamic and automated discovery, invocation and composition of business processes as semantic Web services. In this paper we present a mapping strategy that helps to overcome the syntactical limitations of BPEL processes by presenting them as OWL-S semantic Web services. The proposed strategy supports the mapping of BPEL process descriptions to complete OWL-S suite of ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies). A prototypical implementation of the proposed approach has also been presented.

Keywords: Web Service, Semantic Web Service, Business Process Modeling.

1 Introduction

The trend of developing business applications as services resulted in quick adoption of Web services. With the wide acceptance of the Web service technology different workflow languages (e.g. BPEL4WS [6], MS XLANG [13], IBM WSFL [7]) were developed. These workflow languages can be used to model business processes as compositions of multiple Web services to perform complex tasks that a single Web service alone cannot perform. Major drawbacks of these languages are 1) they compose Web services on the basis of their syntactical information 2) when these processes are exposed as Web services they have same syntactical limitations as traditional WSDL [3] services. Consequently, modeling Web services compositions and discovering, invoking and composing them on the basis of syntactical information is not very efficient and (due to many single points of failure) unreliable. Different research groups in the semantic Web and semantic Web service (SWS) community are working on developing standard languages (e.g. OWL-S [8], WSDL-S [1] and WSMO [2]) to equip Web service with semantics. The SWS community has also presented different approaches to dynamically discover, invoke and compose these services on the basis of matching semantics. Due to dynamic and automated behavior of SWSs they are getting more and more attraction of large business organizations. The aim is to expose business processes as SWSs to achieve the goal of business process automation. At this stage an approach is needed that can be used to shift existing business processes (e.g. BPEL processes) to SWSs (e.g. OWL-S services) in an efficient and cost affective way rather than to build semantic based business services from scratch. Such an approach will help to 1) model the composition of services on the basis of matching semantics 2) provide semantically enriched interfaces of business processes (i.e. BPEL processes) as OWL-S composite services. This semantically enriched information can be used for dynamic discovery, invocation and composition of business processes as SWSs.

Enhancing existing business processes with semantics to enable them for semantic based composition modeling, dynamic discovery, invocation and composition raises following challenges:

- How to provide semantics of individual services within a composition.
- Modeling the composition by defining control flow between child services.
- Defining data flow between child services on the basis of matching semantics.
- Exposing semantically enriched interfaces of resulting composite services.

By successfully addressing these challenges we can enable business processes to:

- Expose semantically enriched interfaces as *Profile* ontologies for semantic based dynamic and automated discovery, invocation and composition.
- Edit and model the composition on the basis of matching semantic information rather than syntactical information.

To achieve these goals by addressing above-mentioned challenges we have developed a strategy (i.e. mapping specifications) that can be used to map existing BPEL processes to OWL-S SWSs. Even though, many efforts have been done before to bridge the semantic gap between process modeling languages (e.g. BPEL, FBPML etc.) and OWL-S SWSs but our approach is unique with respect to its support for mapping of BPEL processes to complete OWL-S suite of ontologies.

The remaining paper is organized as follows: Section 2 describes a motivational scenario for our work. Mapping specifications have been discussed in Section 3. Implementation of the mapping tool has been discussed in Section 4. Section 5 evaluates our approach. The related work has been discussed in Section 6. Section 7 concludes our work.

2 Motivational Scenario

In order to understand the problems raised due to syntactical limitations of BPEL, we consider an example scenario of Web services composition. To keep the complexity of the scenario within limitations we consider a simple *Translator and Dictionary* process example (BPEL file of the process model and OWL files of mapped OWL-S *composite* service and *atomic* processes are available with the tool download[1]). The *Translator and Dictionary* process is modeled in MS BizTalk Server as syntax-based composition of the *Translator* service and the *Dictionary* service. The *Translator* service is a Web service that can be used to translate a string from one language to another language. The *Dictionary* service is a Web service that can be used to get the meaning of an *English* word in *English* (i.e. only the *English* language is supported). Now we define two problem scenarios (tasks) that cannot be performed by anyone of these two services.

1. How we can get the meaning of a *German* word in *English*? Because the *Dictionary* service supports only the meaning of an *English* word in *English*, not the meaning of a *German* word in *English*.
2. How we can get the meaning of a *German* word in *German*? Because the *Translator* service only translates string from one language to other language (not give the meaning of a word) and the *Dictionary* service only gives the meaning of an *English* word in *English*.

In both of above scenarios none of a single Web service (i.e. neither the *Translator* Service nor the *Dictionary* service) is able to perform the required task. As a solution, we model a BPEL process as syntax based composition of the *Translator* service and the *Dictionary* service to perform the required task. For the first problem scenario we can define a workflow (i.e. a BPEL process) as a composition of the *Translator* service and the *Dictionary* service as follows (as shown in Figure 1):

1. The process accepts an input string (i.e. the *German* word) from the user.
2. Transfers this string as an input to the *Translator* service to translate the string from *German* to *English*.
3. Output of the *Translator* service (i.e. the *English* translation of the input string) is given as an input to the *Dictionary* service.
4. As a last step of the process, the *Dictionary* service returns the meaning of the input string.

Similarly the task pointed in second scenario (i.e. getting the meaning of the *German* word in *German*) can be accomplished by enhancing the process model of Web services composition by following steps (as shown in Figure 2):

1. The output of the *Dictionary* service (i.e. the meaning of the word) is given as input to the *Translator* service to translate it back from *English* to *German*.
2. As a last step of the process the *Translator* service translates the string (i.e. meaning of the word) back from *English* to *German*.

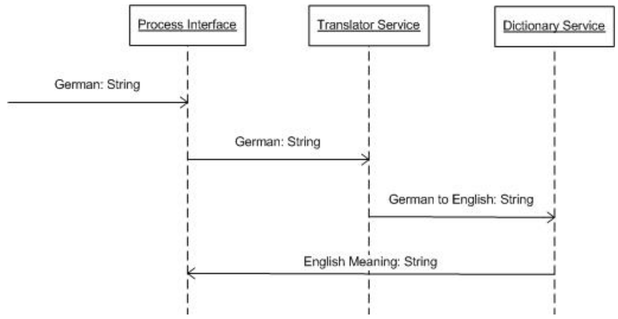


Figure 1 Sequence of services in the process according to first scenario

[1] http://bpel4ws2owls.sourceforge.net/

Figures 1 and 2 give examples of two processes as Web services compositions to perform tasks identified in problem Scenarios 1 and 2. If we analyze the process more at semantic level then the following issues are identified:

- When the process is exported as a Web service, it exposes syntactical interface.
- Web services with in the process provide no information for semantic based editing and modeling of composition.

One thing to note at this point is that we have provided two example scenarios for modeling business processes as Web services compositions. For the first scenario we modeled a BPEL process in MS BizTalk Server (BPEL file of the process is available with tool download). In remaining paper we use this BPEL process to provide some code samples of mapping specifications. Till the end of Section 3 the whole BPEL process is mapped to OWL-S service. Then we use this mapped OWL-S service to answer the problem questions (i.e. to expose semantically enriched interface of the process as SWS and to compose Web services for the problem Scenario 2 on the basis of matching semantics rather than on the basis of their syntactical information).

3 Mapping Specifications

In this section we describe mapping specifications that can be used to translate BPEL process descriptions to OWL-S service descriptions. Since, OWL-S is suite of OWL ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies) therefore, we describe mapping of BPEL process to OWL-S at three levels (i.e. mapping of BPEL process model to OWL-S *Profile, Process Model* and *Grounding* ontologies). Table 1 summarizes the specifications for mapping BPEL process to OWL-S. Mapping specifications describe from abstract level to components and activities level translation of BPEL process to OWL-S service. We have also highlighted some

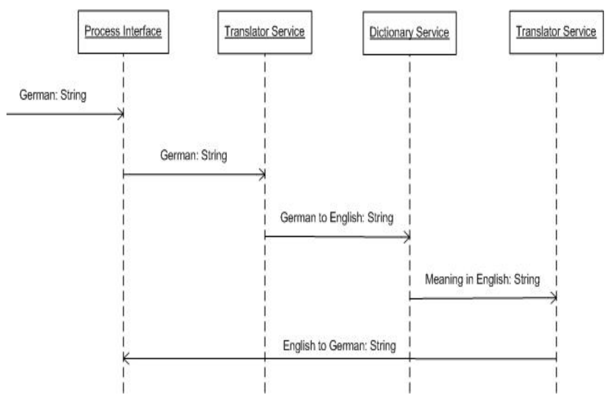


Figure 2 Sequence of services in the process according to second scenario

Table 1 Summary of BPEL4WS to OWL-S mapping specifications

Ontology	BPEL4WS	OWL-S
Profile		
	Receive (message variable)	Input parameters
	Invoke (input message variable)	Output parameters
	Invoke (input/output message variable)	Input/Output parameters
	Reply (message variable)	Output parameters
Process Model		
	Executable process	Composite process
	Primitive activity (operation)	Atomic process
	Primitive activity (Invoke)	*Perform* CC
	Sequence	Sequence
	Flow	Split-Join
	Switch-case	Sequence(If-Then-Else)
	While	Repeat-While
	Condition statement	SWRL expression
	Assignment	Data flow specifications
	Terminate	Note
	Throw	Note
	Wait	Note
Grounding		
	Primitive activity (operation)	hasAtomicProcessGrounding
	Complex Message	xsltTransformationString

Note: No equivalent control construct (CC) in OWL-S is available for direct mapping.

areas where direct mapping is not possible or needs some additional work to be done from the end user.

The Algorithm 1 provides a very abstract level description of the recursive algorithm used for extracting OWL-S suite from BPEL process model. It is used to traverse through BPEL process activities as long as these activities come to an end. An important thing to note is that when an activity is not an input/output (I/O) *primitive* activity then it is mapped to *perform* CC (as described in Lines 13 and 33 of Algorithm 1) to perform the relevant *atomic* process. In next section we describe in detail about the extraction of *Process Model* ontology from BPEL process model.

3.1 Mapping to the OWL-S Process Model Ontology

In this section we describe how a BPEL process model is mapped to OWL-S *Process Model* ontology with defined control and data flow. The *Process Model* mapping specifications describe about how BPEL *primitive* and *structured* activities, condition statements, input/output data passing between different activities, variables etc. are mapped to OWL-S relevant control constructs (CCs), SWRL expressions and parameters respectively. We also provide some code examples of mapping BPEL activities to OWL-S CCs. Now we describe step by step mapping of BPEL process components to OWL-S CCs.

3.1.1 BPEL Process to OWL-S Composite Process

BPEL process model is composition of multiple Web services with defined control and data flow between composed services. A BPEL process model is mapped to an OWL-S *composite* process that is a semantic based composition of multiple *atomic* and *composite* processes. The control flow and data flow between different Web services operations within a BPEL process model is mapped to control flow and data flow between process components of an OWL-S composite service.

3.1.2 Web Service Operation to OWL-S Atomic Process

We discussed before that a BPEL process is composition of Web services operations that can be performed in a single step. Since, small tasks with in a BPEL process are performed by executing Web services operations therefore, a successful and useful mapping of BPEL process model to OWL-S is intimately dependent on translation of each Web service operation involved within a BPEL process to an OWL-S *atomic* process. As much as we know, till now there has no effort been done which supports the mapping of a BPEL process to OWL-S and translates Web services operations within a BPEL process to OWL-S *atomic* processes. Each Web service operation that is mapped to OWL-S *atomic* process is stored in a separate OWL file. Since, our sample *Translator and Dictionary* process consists of two Web services operations (i.e. *getMeaning* and *getTranslation* operations) therefore, these two Web services operations are mapped to two *atomic* processes (i.e. *getMeaningProcess* and *getTranslationProcess*) and stored in separate OWL files (i.e. *getMeaning.owl* and *getTranslation.owl*). We can also execute these *atomic* processes by using some execution engines (e.g. OWL-S API) or by importing and executing them in SWS development tool (e.g. Protégé [5] (OWL-S Editor [4])).

```
Input: Tree view list of BPEL process and WSDL services
Output: OWL-S suite of ontologies
1  begin
2      Extract BPEL activity from tree
3      Map structured activity to OWL-S CC
4      Get child activities
5      while child activities exist do
6          if activity is not structured activity then
7              if activity is assignment activity then
8                  while activity is assignment activity do
9                      Traverse activity list
10                 end
11                 if activity is non-I/O primitive activity
                   (i.e. invoke activity) then
12                     Map it to perform CC to perform atomic
13                     process
14                     Create data flow
15                     Add reference of atomic process
                       Grounding
16                 end
17             end
18             if activity is not assignment activity then
19                 if activity is I/O receive activity then
20                     Create composite process input
21                     Create Profile input parameters
22                 else
23                     if activity is I/O reply activity then
24                         Create composite process output
25                         Create Profile output parameters
26                     else
27                         if activity is I/O invoke activity
                           then
28                             Create composite process output
29                             Create Profile output parameters
30                         else
31                             if activity is non-I/O invoke
                               activity then
32                                 Map it to perform CC to
33                                 perform atomic process
34                                 Create data flow
35                                 Add reference of atomic
                                   process Grounding
36                             end
37                         end
38                     end
39                 end
40             end
41         end
42         if child activity is structured activity then
43             Go to Line 3 (i.e. Map structured activity to
               OWL-S CC)
44         end
45     end
46 end
```

Algorithm 1 Abstract level definition of mapping algorithm

3.1.3 Primitive Activity to Perform Control Construct

In above section we have discussed that a Web service operation performed by a *primitive* activity is mapped to an OWL-S *atomic* process. The *primitive* activity that performs Web service operation is mapped to OWL-S *Perform* CC to perform that *atomic* process within mapped OWL-S service. For example, consider the *primitive* activity (<*invoke*>) that is used to perform a Web service operation *getTranslation*. The *primitive* activity is mapped to *Perform* CC to perform the process *getTranslationProcess* (as shown in sample code below), where *getTranslationProcess* is *atomic* process created in previous section (i.e. Section 3.1.2) and stored in *getTranslation.owl* file.

```
<process:process rdf:resource="http://examples.org/DummyURI.owl#
        getTranslationProcess"/>
```

3.1.4 Structured Activity to OWL-S Control Construct

BPEL *structured* activities are used to define control flow between different child activities. OWL-S provides a number of CCs (e.g. *Sequence*, *Split* etc.) for defining control flow between sub processes. Table 1 summarizes mapping of BPEL *structured* activities to OWL-S CCs on the basis of their matching behavior. As sample of mapping these activities we describe the translation of BPEL *structured* activity (i.e. *switch* activity) to relevant OWL-S CC (i.e. sequence of *If-Then-Else* CCs), because mapping of *switch* activity is a little bit tricky.

A *switch* activity is used to describe the conditional behavior and consists of a list of one or more conditional branches defined by using *case* elements. A *case* element has a *condition* attribute to define its condition and can have an optional *otherwise* branch that is executed if the *case* condition becomes false. The *switch* activity is mapped to *Sequence* CC of OWL-S specifications and each *case* element listed under *switch* activity is mapped to

If-Then-Else CC. The *condition* part of each *case* element is translated to SWRL expression and *otherwise* part of *case* element is mapped to *else* part of *If-Then-Else* CC. We can summarize the mapping of *switch* activity with a list of *case* elements as a sequence (*Sequence*) of *If-Then-Else* CCs mapped with optional *else* part. Let us consider following *switch* activity example:

```
<switch>
 <case condition="bpel:getVariableData('Input\_Message',
                  'part', 'inputLang')='English'">
  <invoke partnerLink="Dictionary\_Ser\_Port"...../>
 </case>
</switch>
```

Sample code of mapped *switch* activity in OWL-S is as under:

Example. 1 BPEL Switch activity mapped to OWL-S *Sequence* of *If-Then-Else* CC and *condition* statement translated to SWRL expression (In all remaining examples '&bpel4ws2owls[2], '&dummyURI[3]' are dummy URIs that are used by mapping tool).

```
1  <process:Sequence>
2   ..........
3   <process:If-Then-Else>
4    <process:ifCondition>
5     <expression:SWRL-Condition>
6      <expression:expressionBody rdf:parseType="Literal">
7       <swrl:AtomList xmlns:swrl="http://www.w3.org/2003/11/swrl#">
8        ..........
9        <swrl:BuiltinAtom>
10        <swrl:builtin rdf:resource="http://www.w3.org/2003/11/
11                          swrlb#equal">
12        </swrl:builtin>
13         ..........
14        <rdf:first rdf:resource="&bpel4ws2owls#inputLang">
15        </rdf:first>
16         ..........
17        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#
18                   String">'English'</rdf:first>
19        ..........
20       </swrl:BuiltinAtom>
21      </swrl:AtomList>
22     </expression:expressionBody>
23    </expression:SWRL-Condition>
24   </process:ifCondition>
25   <process:then>
26    <process:process rdf:resource="&dummyURI#getMeaningProcess"/>
27   </process:then>
28  </process:If-Then-Else>
29 </process:Sequence>
```

In Example 1, we have discussed a very simple conditional scenario in which *switch* activity involves single *case* element that is mapped to *If-Then-Else* CC. If a *switch* activity has multiple *case* elements, which may have optional otherwise branches, then Algorithm 2 is used to traverse through the list of *case* elements and to map each *case* element to *If-Then-Else* CC with in a *Sequence* CC.

[2] http://www.BPEL2OWLS.org/ChangeTestURI.owl
[3] http://examples.org/DummyURI.owl

Conditions are an important part of BPEL activities (e.g. *switch*, *while* etc.) and OWL-S CCs (e.g. *If-Then-Else*, *Repeat-While* etc.). Without mapping *condition* statements, only mapping of BPEL activities to OWL-S CCs that depend on conditions, is not useful. Consequently *condition* statements of BPEL activities are translated to SWRL expressions that are supported by OWL-S specifications. Mapping *condition* statements to SWRL expressions support all conditional operators (e.g. $=, !=, <, >, <=, >=$ etc.). Example 1, Lines 3-24 show a mapped SWRL expression of *If-Then-Else* CC.

3.1.5 Message Assignment to Data Flow

We can discuss the mapping of data flow at two levels: 1) defining inputs and outputs of a *composite* process 2) defining data flow to pass messages between process components (i.e. sub processes) within *composite* process. To understand the data flow definition at first level we consider our sample *Translator and Dictionary* process in which *receive* activity receives a message from the outer world to start a process. Such a message that initiates a process is defined as input message of *composite* process within *Process Model* ontology of mapped OWL-S service. In remaining process this message is referred as a message that belongs to the process *TheParentPerform* to pass this message as input of sub processes. Similarly such situations are also possible in which the output of a sub process becomes the output of the *composite* process. In such cases output of sub process is also defined as output of the process *TheParentPerform*.

As an example consider a *receive* activity *Translator and Dictionary* process which receives a message to start the process. The definition of message (*Input_Message*) received by *receive* activity is given in relevant WSDL file (as shown in sample code below).

```
Input: Switch activity
Output: Sequence of If-Then-Else CCs in resulting
        OWL-S service
begin
  if activity equal to switch then
    Map switch activity to Sequence CC
    Traverse through switch activity
    while activity equal to case do
      Map case element to If-Then-Else CC
      Map condition statement to SWRL expression
      Go to Line 4 of Algorithm 1 (i.e. to map child
      activities under case element)
    end
    if activity equal to otherwise then
      Create else part of If-Then-Else CC
      Go to Line 4 of Algorithm 1 (i.e. to map child
      activities under otherwise part)
    end
  end
end
```

Algorithm 2 Algorithm to traverse through *Switch* activity and its *case* elements and to map them to relevant OWL-S CC

```
<types>
 <xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   ..........
   <xs:complexType>
    <xs:sequence>
     <xs:element name="inputStr" type="xs:string" />
     <xs:element name="inputLang" type="xs:string" />
     <xs:element name="outputLang" type="xs:string" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xsd:schema>
</types>
```

An important thing that need to be noted is that above message definition is based only on syntax and provides no semantic information that can be used by computer agents for the purpose of dynamic and automated discovery, invocation and composition. When such messages are mapped to OWL-S, they are annotated with domain ontologies to provide semantics for computer agents to reason about them (we discuss these issues in detail in Section 3.2). However, above message is used to define data flow as input of *composite* process (as shown below).

```
<process:CompositeProcess rdf:about="&bpel4ws2owls#TestProcess">
 <process:composedOf>
  ..........
   <process:hasInput rdf:resource="&bpel4ws2owls#inputLang"/>
   <service:describes rdf:resource="&bpel4ws2owls#TestService"/>
   <process:hasInput rdf:resource="&bpel4ws2owls#inputStr"/>
   <process:hasInput rdf:resource="&bpel4ws2owls#outputLang"/>
  ..........
 </process:composedOf>
</process:CompositeProcess>
```

This input message of the *composite* process can be passed as input of sub processes (*atomic* or *composite* processes) within *Process Model* ontology. For example, in our mapped OWL-S service, *getTranslation1* is an *atomic* process within mapped *composite* process. The sample code below shows that input parameter of *composite* process (i.e. *inputLang*) can be passed as input (*inputLanguage*) of the sub *atomic* process (*getTranslation1*).

```
<process:Perform rdf:about="&dummyURI#getTranslation1">
 <process:hasDataFrom>
  <process:InputBinding>
   <process:valueSource>
    <process:ValueOf>
     <process:theVar rdf:resource="&bpel4ws2owls#inputLang"/>
      <process:fromProcess rdf:resource="http://www.daml.org/
              services/owl-s/1.1/Process.owl#TheParentPerform"/>
    </process:ValueOf>
   </process:valueSource>
   <process:toParam rdf:resource="&dummyURI#inputLanguage"/>
  </process:InputBinding>
 </process:hasDataFrom>
    ..........
    ..........
</process:Perform>
```

We have also discussed that within a BPEL process model output of one Web service operation can be used as input of the next Web service operation. During the mapping of a BPEL process to OWL-S service, passing messages between sub processes within a *composite* process is addressed by using the *Binding* class.

3.1.6 Variables to Local Parameters

Like traditional programming languages, we can also declare variables in a BPEL process to store and share data between different activities within a process. Such variables within a BPEL process are mapped to local variables (*LocalVariable*) in OWL-S that can be used to manipulate and share data between sub *atomic* and *composite* processes.

In this section we have discussed the translation of BPEL process model to OWL-S *Process Model* ontology. Translation of some of BPEL activities to OWL-S CCs has been described with their syntactical examples to describe mapping aspects with respect to their language specifications. The mapped *Process Model* ontology can be used to further edit and model more complex service in a semantic environment (as discussed in Section 5 to evaluate our approach).

3.2 Mapping to the OWL-S Profile Ontology

Profile ontology is used to describe semantically enriched information about capabilities of a BPEL process when it is mapped to OWL-S SWS. Semantically enriched information about capabilities of mapped process model is described as 1) *inputs* required by the service 2) *outputs* generated by the service 3) *pre-conditions* required to use a service 4) *effects* that service produces in surrounding world after its execution. Annotating these input/output parameters, pre-conditions and effects with domain ontologies defined in separate owl files provide their semantics. Since, BPEL process model provides no semantic information about a process therefore, *Profile* ontology parameters of mapped OWL-S service are automatically annotated by mapping tool with dummy ontological concepts (i.e. URIs). Since, semantic information about service capabilities can vary from user to user therefore, it is not possible to judge the user require-ments automatically, generate domain ontologies according

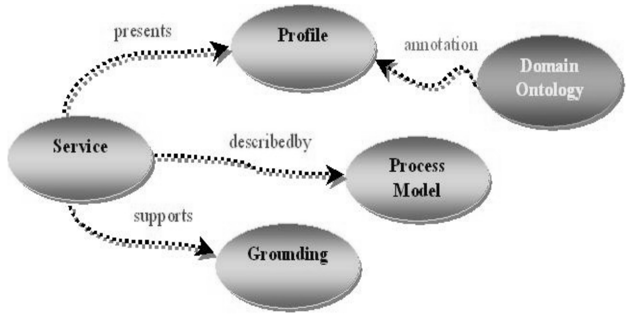


Figure 3 Annotating *Profile* ontology with domain ontology concepts

to those requirements and annotate *Profile* ontology parameters with these ontological concepts. Maximum process of generating *Profile* ontology from BPEL process is performed automatically by mapping tool but end user can provide semantically enriched information about capabilities of mapped OWL-S service by annotating input/output parameters of *Profile* ontology with their required domain ontologies (as shown in Figure 3).

3.2.1 Extracting the Profile Ontology

Activities in a BPEL process can have dual behavior and mapping of these activities depends on the behavior that a primitive activity plays in a process. For example, in a BPEL process, *primitive* activities can have dual behavior i.e. they can be used to 1) perform a Web service operation 2) interact with the outer world (i.e. to create interface of a BPEL process model). Mapping of *primitive* activities that are used to perform Web services operations with in a BPEL process has been discussed in Sections 3.1.2 and 3.1.3. Here, we are concerned with *primitive* activities that can be used to create interface of a BPEL process model. A BPEL process can have one or more *primitive* activities (i.e. *receive*, *invoke* and *reply* activities) that can be used to interact with the outer world. Such activities are declared as input/output (I/O) activities during mapping process.

Example. 2 An example of mapped *Profile* ontology.

```
<profile:Profile rdf:about="&bpel4ws2owls#TestProfile">
 <profile:textDescription>This Profile is created by BPEL2OWLS Tool
 </profile:textDescription>
 <profile:hasInput>
  <process:Input rdf:about="&bpel4ws2owls#inputStr">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/
    XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#string
   </process:parameterType>
  </process:Input>
 </profile:hasInput>
 <profile:hasInput>
  <process:Input rdf:about="&bpel4ws2owls#inputLang">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/
    XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#string
   </process:parameterType>
  </process:Input>
 </profile:hasInput>
 <profile:hasInput>
  <process:Input rdf:about="&bpel4ws2owls#outputLang">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/
    XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#string
   </process:parameterType>
  </process:Input>
 </profile:hasInput>
 <profile:hasOutput>
  <process:Output rdf:about="&bpel4ws2owls#TestOutput0">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/
    XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#string
   </process:parameterType>
  </process:Output>
 </profile:hasOutput>
 <rdfs:label>BPEL2OWLS Profile</rdfs:label>
 <service:presentedBy rdf:resource="&bpel4ws2owls#TestService"/>
</profile:Profile>
```

Message parts of these I/O activities messages are used to create input and output parameters of *Profile* ontology. For example, if a process has a *receive* activity which receives a message from user to start a process then this activity is declared as an I/O activity and message parts of the message received by this activity are used to create input parameters of resulting *Profile* ontology.

As an example, consider a (*<receive>*) activity and its message that has three parts (i.e. *inputStr, inputLang* and *outputLang*, defined in BPEL's corresponding WSDL file). These message parts are used to create input parameters of resulting *Profile* ontology (as shown in Example 2).

A *reply* activity can be used to send a message to the outer world in response to a *receive* activity. If a *receive* activity has corresponding *reply* activity then message parts of the message of such *reply* activity are used to create output parameters of mapped *Profile* ontology. It is also possible that a *receive* activity do not has corresponding *reply* activity (as you can see in some example BPEL processes available with tool download) and BPEL process uses *invoke* activity to send output message to the outer world. In this case message parts of the message of *invoke* activity are used to create output parameters of *Profile* ontology of mapped OWL-S service.

Another prestigious issue that we think is important to highlight is that mapping specifications support to extract one *Profile* ontology from a BPEL process model. It means that if a BPEL process has multiple activities that act as interfaces of BPEL process then message parts of messages of only two *primitive* activities are used to create input and output parameters of *Profile* ontology of mapped OWL-S service. Even though OWL-S specifications support to create multiple *Profile* ontologies for one *Process Model* ontology but automatic mapping from BPEL process model to OWL-S suite extracts one *Profile* ontology for one *Process Model* ontology. End users can also define multiple *Profile* ontologies for one *Process Model* ontology to provide different meaning of same service.

3.3 Mapping to the OWL-S Grounding Ontology

Grounding ontology of OWL-S suite describes that how to access a service. Access details described in *Grounding* ontology include information about protocol, transport and message formats. These details enable *Grounding* to provide concrete level specifications needed to access a service. Concrete level definition of inputs and outputs of *atomic* processes in some transmittable format is provided in *Grounding* ontology. For this purpose, during the mapping process, original WSDL services are referred in *Grounding* to access real implementation of *atomic* services. When a Web service operation within a BPEL process is mapped to OWL-S *atomic* process (during the mapping process) then input and output messages of Web service operation are defined as set of inputs and outputs in the *Grounding* ontology of that *atomic* process. That's why in Section 2 we have seen that input/output messages of I/O activities are not directly used to create *Profile* ontology but message parts of these messages are used as set of input and output parameters in *Profile* ontology. These input and output parameters when annotated with domain ontologies, provide Web service semantics.

Now about types of messages and message parts: there are two possibilities 1) the message is a complex message of some OWL class type 2) the message is of other usual data type (e.g. string, int etc.). In first case, in which message is of some OWL class type, we need to give the definition of OWL class. This definition can be given within the same document or can be defined in separate OWL file and can be referred in the type parameter.

An OWL-S service *Grounding* is an instance of the *Grounding* class which has sub class *WsdlGrounding*. Each *WsdlGrounding* class contains a list of *WsdlAtomicProcessGrounding* instances which refer to groundings of *atomic* processes. *WsdlAtomicProcessGrounding* has properties (e.g. *wsdlInputMessage*, *wsdlInput, wsdlOutputMessage, wsdlOutput* etc.). A *wsdlInputMessage* and *wsdlOutputMessage* objects contain mapping pairs for message part of WSDL input/output messages and is presented by using an instance of *WsdlInputMessageMap*. If a message part is of some complex type (e.g. some OWL class) then XSL Transformation (XSLT) property gives an XSLT script that generates message parts from an instance of the *atomic* process. As an example consider grounding (as shown in sample code below) of mapped OWL-S service.

```
<grounding:WsdlGrounding rdf:about="&bpel4ws2owls#TestGrounding">
 <service:supportedBy rdf:resource="&bpel4ws2owls#TestService"/>
 <grounding:hasAtomicProcessGrounding rdf:resource="&dummyURI
           #getTranslationAtomicProcessGrounding"/>
 <grounding:hasAtomicProcessGrounding rdf:resource="&dummyURI
           #getMeaningAtomicProcessGrounding"/>
</grounding:WsdlGrounding>
```

The above sample code gives an example of grounding of mapped composite service (i.e. *TestService*), where *getTranslationAtomicProcessGrounding* and *getMeaningAtomicProcessGrounding* are groundinges of two *atomic* processes that are sub processes within mapped *composite* process. The sample ontology shown below gives an example of *Grounding* ontology of the getTranslation *atomic* process.

```
<grounding:WsdlGrounding rdf:about="#getTranslationGrounding">
 <grounding:hasAtomicProcessGrounding>
  <grounding:WsdlAtomicProcessGrounding
           rdf:ID="getTranslationAtomicProcessGrounding"/>
 </grounding:hasAtomicProcessGrounding>
 <service:supportedBy rdf:resource="#getTranslationService"/>
</grounding:WsdlGrounding>

<grounding:WsdlAtomicProcessGrounding rdf:about=
          "#getTranslationAtomicProcessGrounding"}>
 <grounding:wsdlInputMessage rdf:datatype="http://www.w3.org/2001/
           XMLSchema#anyURI">&wsdlFileAddress#TranslatorRequest
 </grounding:wsdlInputMessage>
  <grounding:wsdlInput>
   <grounding:WsdlInputMessageMap>
    <grounding:wsdlMessagePart "http://www.w3.org/2001/
              XMLSchema#anyURI">&wsdlFileAddress#inputLanguage
    </grounding:wsdlMessagePart>
    <grounding:owlsParameter rdf:resource="&wsdlFileAddress
              #inputLanguage"/>
   </grounding:WsdlInputMessageMap>
  </grounding:wsdlInput
  ..........
  .......... (other message parts)
</grounding:WsdlAtomicProcessGrounding>
```

4 Implementation of the Mapping Tool

We have developed a tool (i.e. BPEL4WS 2 OWL-S Mapping Tool) that can be used to translate existing BPEL processes to OWL-S services. The BPEL4WS 2 OWL-S Mapping Tool is an open source project and has hundreds of download since the time it has been uploaded to open source project directory (i.e. sourceforge.net).

4.1 Architecture

The overall architecture of the BPEL4WS 2 OWL-S Mapping Tool consists of three components (i.e. WSDL Parser, BPEL Parser and OWL-S Mapper) as shown in Figure 4. As, it is clear from name that the WSDL Parser parses each WSDL file with in mapping project and creates their object view. An important feature of the WSDL Parser

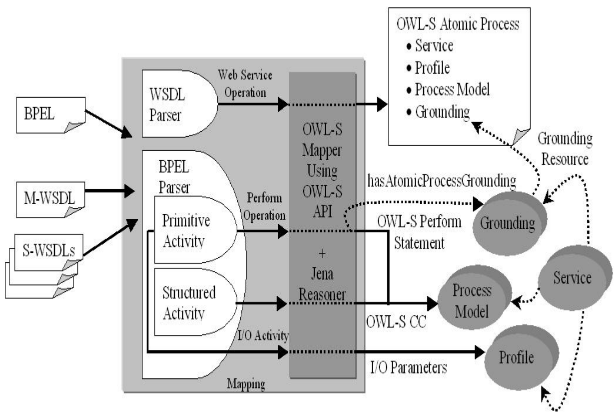


Figure 4 Architecture of the mapping tool

is that it extracts information about operations supported by Web services and sends this information to the OWL-S Mapper which maps each Web service operation to OWL-S atomic process. The OWL-S Mapper writes the generated OWL-S atomic process in a separate OWL file and saves it in atomic processes directory of the mapping project.

The BPEL Parser traverse through a BPEL file and creates object view of process activities. It parses *primitive* activities and sends information about these activities to OWL-S Mapper. Before sending information to the OWL-S Mapper, the BPEL parser declares either a *primitive* activity is an I/O activity or not (Section 4.2 describes in detail that how an activity is declared and mapped as an I/O activity). If a *primitive* activity is declared as an I/O activity then the OWL-S Mapper uses message parts of the message of this activity to create input/output parameters of *composite* process that ultimately are used to create the *Profile* ontology parameters. If a *primitive* activity is not an I/O activity then the OWL-S Mapper maps this activity to *Perform* CC to perform relevant *atomic* process. Also, the BPEL Parser parses *structured* activities in defined control flow of input BPEL process and sends information about these activities to the OWL-S Mapper. The OWL-S Mapper translates them to relevant CCs to define control flow of mapped OWL-S composite service. If a BPEL parser comes to some conditional *structured* activity then it simply sends condition string to the OWL-S Mapper that creates corresponding SWRL expression (as explained in Section 3.1.4).

The OWL-S Mapper is actually responsible for writing resulting OWL-S service according to defined mapping specifications. It uses OWL-S API [12] developed by mindswap lab to write resulting OWL-S composite service. OWL-S API is set of APIs that can be used to *read*, *write* and *execute* OWL-S services. Since, OWL-S API uses

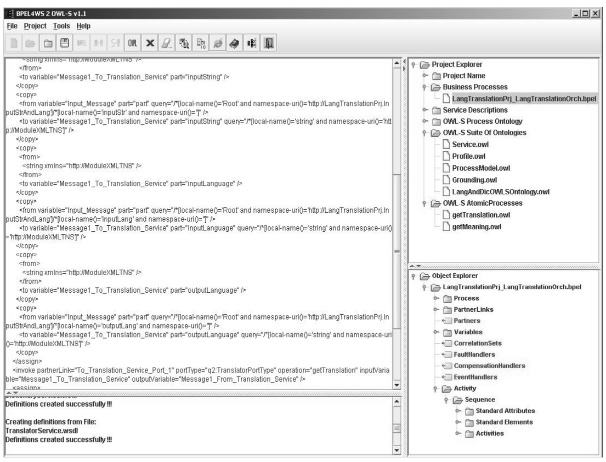


Figure 5 Overview of the mapping tool

a third party reasoner (e.g. jena reasoner) to reason the mapped OWL-S ontology therefore, our tool also uses jena reasoner (as default reasoner) for such reasoning purposes.

4.2 User Interface

The BPEL4WS 2 OWL-S Mapping Tool provides a very easy to use interface which consists of four major parts (i.e. Project Explorer, Object Explorer, Content Window and Output Window) and a *Toolbar* and *Menu bar* (as shown in Figure 5).

The Project Explorer can be used to see the input and output files of a mapping project. The Object Explorer provides object view of input BPEL and WSDL files. The Content window can be used to see contents of any of the input/output files. User can simply select a file in the Project Explorer to see its contents in the Content Window. Output of different actions performed (e.g. Validate, Build and Map) can be seen in the Output Window.

5 Evaluation

In Section 2, we defined two problem scenarios (as show in Figures 1 and 2). We modeled a BPEL process in MS BizTalk Server as syntax based composition of the *Translator* service and the *Dictionary* service to perform the task defined in first scenario. Then we analyzed the BPEL and OWL-S processes and their components and defined step by step mapping of BPEL process to OWL-S SWS. Till the end of Section 3 the whole BPEL process was mapped to OWL-S SWS (the sample BPEL process and mapped OWL-S composite service and atomic processes are available with the tool download) with each Web service operation within a BPEL process mapped to OWL-S *atomic* process.

As a first step to edit the mapped OWL-S service to perform the task discussed in the second scenario (as shown in Figure 2), we replace dummy URIs of input and output parameters of mapped *atomic* and *composite* processes with domain ontologies (as discussed in Section 3.2). The annotation of input/output parameters can be performed by opening the mapped OWL files (i.e. *atomic* and *composite* processes) in the OWL-S Editor (even though some compatibility issues between OWL-S Editor and our tool still need to be addressed) or in any other editor (e.g. Notepad). Annotating input/output parameters helps to edit and extend the *composite* process by defining data flow between sub processes on the basis of matching semantics. The mapped OWL-S service takes *inputString, inputLang* and *outputLang* as inputs of the service. The first *atomic* process (i.e. *getTranslationProcess*) translates the input string from input language (i.e. German) to output language (i.e. English) and the second *atomic* process (i.e. *getMeaningProcess*) provides the meaning of the input word in English language. From here we start editing the mapped service and add one more *atomic* process (i.e. *getTranslationProcess*) within the *Sequence* CC of *composite* process. This *atomic* process is used to perform the additional task defined in second scenario (i.e. to translate the meaning of the German word back from English to German). For this purpose we define data flow for this newly added *atomic* process. The *getTranslationProcess* process takes as input *inputLang* (i.e. English), *outputLang* (i.e. German) and *inputStr* (i.e. output of the *atomic* process *getMeaningProcess*). The data flow can be defined on the basis of matching semantics by using the data binding between *atomic* processes.

In Section 1, we identified two goals that we want to achieve from this work 1) to expose semantically enriched interfaces of business processes 2) semantic based composition modeling. We address both of these problems by mapping BPEL process to OWL-S. *Profile* ontology of mapped OWL-S service provides semantically enriched interface of BPEL process that can be used for dynamic discovery, invocation and composition of BPEL process as OWL-S service. Mapped OWL-S service is edited and extended on the basis of matching semantic information rather than syntactical information to solve the problem defined in second scenario (Figure 2).

6 Related Work

Several efforts have already been done to address semantic limitations of process modeling languages. For example, the work discussed in [10, 11] describes mapping of BPEL process model to OWL-S *Process Model* ontology. Another effort [9] has been done by a joint group of researchers from University of Edinburgh and School of Informatics to address semantic limitations of Fundamental Business Process Modeling Language (FBPML) by mapping it to OWL-S *Process Model*. The work discussed in [10, 11] and [9] supports only the mapping of process model to OWL-S *Process Model* ontology. It does not support the mapping of *Profile* and *Grounding* ontologies. We can summarize that different have been done to address semantic limitations of process modeling languages by mapping them to semantic Web service languages (e.g. OWL-S) but none of these efforts provide expressive and consistent solution. Our work is unique with these aspects that it supports the mapping of BPEL process model to complete OWL-S suite of ontologies.

7 Conclusion

In this paper we have presented an approach to bridge the semantic gap between business process models and SWSs by mapping BPEL processes to OWL-S suite of ontologies. We have implemented our approach as a mapping tool that can be used to map BPEL processes to OWL-S services. Since, OWL-S is not as mature as BPEL therefore, we have also highlighted different areas where direct mapping is not supported. For example, in order to implement direct mapping of BPEL activities (e.g. terminate, fault handling etc.) we need more consistent specifications of OWL-S to address these issues. We have also highlighted areas where some human user input is required (e.g. changing parameter type by annotating input/output parameters with domain ontologies etc.).

To this end, bridging the semantic gap between business process modeling languages and semantic Web services becomes more important to keep existing business processes alive with upcoming semantic Web service technology. Our approach will help in enabling existing business process (i.e. BPEL processes) with semantics by shifting them to OWL-S services. These OWL-S services can be used for co-operation between business partners in a dynamic and computer understandable way.

Acknowledgement

This work is partially supported by the Higher Education Commission (HEC) of Pakistan under the scheme "Partial Support Scholarship for PhD Studies Abroad”.

References

[1] R. Akkiraju, J. Farell, J. Miller, M. Nagarajan, A. Sheth, and K. Verma. Web service semantics - wsdl-s. [online] Available http://www.w3.org/Submission/WSDL -S/, Nov. 2005.

[2] S. Arroyo, E. Cimpian, J. Domingue, C. Feier, D. Fensel, B. König-Ries, H. Lausen, A. Polleres, and M. Stollberg. Web service modeling ontology primer. [online] Available http://www.w3.org/Submission/WSMO-primer/ , Jun 2005.

[3] D. Booth and C. K. Liu. Web services description language (WSDL) version 2.0 part 0: Primer. [online] Available http://www.w3.org/TR/2006/CR-wsdl20-primer- 20060327, Mar. 2006.

[4] D. Elenius, G. Denker, D. Martin, F. Gilham, J. Khouri, S. Sadaati, and R. Senanayake. The OWL-S editor - A development tool for semantic web services. In 2nd

European Semantic Web Conference, Vol. 3532, 2005, pp. 78-92.

[5] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of Protégé: an environment for knowledge -based systems development. Int. Journal of Human- Computer Studies, 58(1): 2003, pp. 89-1233.

[6] M. Juric, B. Mathew. P. Sarang: Business Process Execution Language for Web Services. A Practical Guide to Orchestrating Web Services Using BPEL4WS. PACKT Publishing, Oct. 2004.

[7] F. Leymann. Web services fow language (wsfl 1.0). [online] Available http://www-306.ibm.com/software/solutions/webservices/ pdf/WSFL.pdf, May 2001.

[8] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. Mcllraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. Owl-s: Semantic markup for web services. [online] Available http://www.ai.sri.com/daml/services /owl-s/1.2/ overview/, Mar. 2006.

[9] G. Nadarajan and Y.-H. Chen-Burger. Translating fundamental business process modelling language to the web services ontology through lightweight mapping. Technical Report EDIINFRR0942, The University of Edinburgh, 2007.

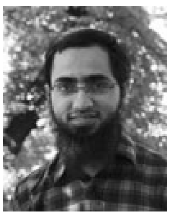
[10] J. Shen, G. Grossmann, Y. Yang, M. Stumptner, M. Schrefl, and T. Reiter. Analysis of business process integration in web service context. In FGCS: The Int. Journal of Grid Computing: Theory, Models and Applications, number ISSN:0167-739X. Elsevier Publishers, May 2006.

[11] J. Shen, Y. Yang, C. Wan, and C. Zhu. From BPEL4WS to OWL-S: Integrating E-business process descriptions. In IEEE Computer Society, 2005, pp. 181-190.

[12] E. Sirin. Owl-s api. [online] Available http://www.mindswap.org/2004/owl-s/api/, Aug. 2001.

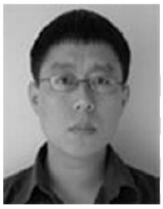
[13] S. Thatte. XLANG web services for business process design. [online] Available http://xml.coverpages.org/XLANG-C-200106.html, 2001.

Biographies



Muhammad Ahtisham Aslam is PhD student and research assistant at Business Information Systems (Betriebliche Informationssysteme) research group at University of Leipzig, Germany. He has written several scientific articles. His research interests include Business Process Modeling, Semantic Web and Semantic Web Services.

Sören Auer is postdoctoral researcher at the depatment of Business Infomation Systems at Universität Leipzig, Germany and the database group at University of Pennsylvania, USA. His research interests include Knowledge Representation and Management, in particular agile, light-weight methodologies and social semantic collaboration.



Jun Shen is lecturer in IT at School of Information Systems and Technology, University of Wollongong, Australia. He has been a research fellow in Swinburne University of Technology, Melbourne and University of South Australia. His research interests include Semantic Web,Web Services, Knowledge Grid and Mobile Services.

Klaus-Peter Fähnrich is professor of computer science and chair of Business Information Systems research group at Institute of Computer Science, University of Leipzig, Germany. His main areas of research and teaching are Business Information Systems, Software Engineering and Management, E-Business and Services Science.