From BPEL4WS Process Model to Full OWL-S Ontology

Muhammad Ahtisham Aslam, Sören Auer and Martin Böttcher
University Of Leipzig
Leipzig, Germany
ahtisham_a@hotmail.com
{auer,boettcher}@informatik.uni-leipzig.de
Jun Shen {jun.shen@cs.unisa.edu.au}
School of Computer and Information Sciences, University of South Australia
Adelaide, Australia

Abstract. BPEL4WS is one of the most well known Business Process development languages. It can be used to develop executable Business Processes as a combination of Web Services interaction in a specific sequence called process flow. But still BPEL4WS does not present meaning of a business process so that business processes can be automated in a computer understandable way. On the other hand OWL-S (OWL for services) is designed to represent such kind of semantic information. There exists an overlap in the conceptual model of OWL-S and BPEL4WS that can be used to overcome this lack of semantics in BEPL4WS by mapping BPEL4WS to OWL-S. In this paper we are presenting a mapping strategy and a prototype tool implementing this strategy to map BPEL processes to OWL-S ontology (Service, Profile, Process Model and Grounding) to overcome the semantic limitation of BPEL4WS.

1 Introduction

Web Services [1], the Business Process Execution Language for Web Service (BPEL4WS) [2] shortly known as BPEL and now the OWL-based Web Service Ontology (OWL-S) [3] are result of efforts to achieve efficient and reliable communication in the growing e-business world.

Web Services, as an effort in this direction were developed to achieve efficient, dynamic and platform independent interaction between business partners. Business partners can develop business applications as business services (WSDL services) [4] and publish them in a UDDI (Universal Description, Discovery and Integration) server [10]. Interested parties can find and use these services in a loosely coupled way regardless of their location on the network. But even then Web Services do not provide much semantic information so that they can be discovered and interacted in a computer understandable way. Publishing and discovering a required Web Service involves a lot of human involvement. Bringing the semantics to Web Services technology aims at replacing these human involvements with computer understanding by making the use of ontologies.

Different workflow languages specially Business Process Execution Language for Web Services (BPEL4WS) uses Web Services in a more meaningful way by combining Web Services functionality in a specific sequence to perform some specific business task. BPEL4WS defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners [14]. BPEL has good process modeling capabilities but lack of semantic information in a BPEL processes left the aim of process automation on the road. Even though BPEL has good process modeling capabilities even then semantic limitations of BPEL are a big hurdle in business process automation.

OWL-S as a parallel effort in this regard is an attraction for Semantic Web and process modeling community. OWL-S aims to make Web Services computer-interpretable, described with sufficient information to enable automation of variety of tasks including Web Service discovery, invocation, and composition [17]. Since both OWL-S and BPEL have process modeling capabilities therefore mapping of a BPEL process to an OWL-S ontology will result in use of process modeling capabilities and semantic capabilities of OWL-S. This semantically enriched information about service capabilities (annotated with ontological concepts) will help in achieving the aim of business process automation.

In this paper we present a mapping strategy to map BPEL processes to OWL-S ontology and our prototype mapping tool implementing this mapping strategy. Our BPEL to OWL-S mapping tool is an extension to BPEL4WS2OWL-S tool, by “CICEC Lab” [9], to achieve real and more consistent mapping. The tool by “CICEC Lab” provides a very initial kind of mapping. For example it parses the WSDL files and in mapped OWL file it writes the WSDL operation as an Atomic Process with its inputs and outputs without Profile, Process Model and Grounding of that Atomic Process. We can say that mapping takes no care for OWL-S specifications. Work by CICEC Lab (Jun Shen and Yun Yang) has following major drawbacks:

- Atomic Processes are not supported according to OWL-S specifications.
- Complex message types are not supported.
- No data binding is supported between Atomic Processes.
- Atomic Processes have no “Profile”, “Process Model” and “Grounding”.
- Atomic Processes can not be invoked and executed in resulting OWL-S service.
- Resulting OWL-S service has no “Profile” and “Grounding”.
- Mapping does not support the OWL-S specification.

Therefore unavailability of service profile that can be used to present semantically enriched information about service capabilities keeps the mapping away from its goal. Also unavailability of service grounding results in communication restrictions with corresponding partner services. Unavailability of support for complex message types keep mapping away from creating data flow. We can summarize the work by Jun Shen and Yun Yang by saying that the tool parses the BPEL file and tells about the activities flow and map it to their relevant OWL-S control constructs. WSDL files are parsed and each operation is presented as an Atomic Process (without OWL-S supported specifications).

Our work is an effort to achieve more consistent mapping resulting in full OWL-S suite of ontologies. Our work supports complex message types, WSDL operations are mapped to OWL-S Atomic Processes (with Profile, Process Model and Grounding), data flow between Atomic Processes is supported and the mapped OWL file has the complete OWL-S suit of ontology (Profile, Process Model and Grounding). Also Atomic Processes are grounded with real WSDL services so that they can be invoked and executed on network.

The remaining paper is organized as follows: In section 2 we discuss some important concepts, concerning the relevant technologies i.e. BPEL, OWL-S and some introductory sentences about the OWL-S API being an important part of mapping implementation. In section 3 we discuss the mapping strategy. Section 4 describes creation of OWL-S suite of ontologies. Section 5 describes the mapping tool. In section 6 we discuss the related work and future planes. Section 7 draws the conclusion of our work.

2 Background

2.1 BPEL4WS

From early days of efforts for Web Services aim of the community is to develop a technology with which business partners can communicate with each other in platform independent and computer understandable way without human interaction. The Web Services community in this regard was successful in developing a language “Web Services Description Language” (WSDL) [4] with which business partners can communicate in platform independent way. But still efforts for seamless communication between partners are on the way.

Different workflow languages e.g. WSFL [11], MS XLANG [12] and BPEL4WS [2] were developed to bring the use of Web Services to a higher level, at which they can be used in a more meaningful way to perform some specific task. Among these languages BPEL got more attraction of the community for modeling business processes as a composition of web services.

Figure 1 gives an overview of BPEL activities. Figure 1 shows that BPEL presents two kind of activities to be used for process modeling i.e. *Basic* or *Primitive Activities* and *Structured Activities*. Primitive activities e.g. “Invoke”, “Receive” “Reply” are used to model interaction between business partners, where as workflow in a BPEL process model is modeled by using the Control Constructs e.g. “IfThenElse”, “While Loop” etc. These activities can be nested in some structured activities according to requirements e.g. “Sequence” activity can be used to perform sub activities in a sequence. “Flow” activity can be used to perform sub-activities concurrently and to synchronize sub-activities. Key components of a BPEL process model are partners that associate a web service defined in an accompanying WSDL document with a particular role and variables. Variables contain the messages passed between partners and correspond to message in accompanying WSDL documents [5]. Butt effective dynamic service binding cannot be performed by solely matching WSDL messaging interfaces.

BPEL4WS Activities	
Basic OR Primitive Activities	**Structured Activities**
• Receive	• Sequence
• Send	• Flow
• Invoke	• Switch
	• While
	etc.

Fig.1. BPEL4WS activities table.

Although BPEL has many advantages, it also has some limitations. Because expressiveness of WSDL service behavior is restricted to interaction specifications and BPEL uses WSDL portType as service information, therefore BPEL inherits the limitations of WSDL. Furthermore, BPEL cannot express the inheritance and relationships among the web services. It cannot provide well-defined semantics for automated composition and execution. Moreover, these languages are based on XML in essence, they are limited in semantic descriptions without enough ontology support [6].

2.2 OWL-S

Towards ultimate goal of seamless interaction among networked programs and devices, industry has developed orchestration and process modeling languages such as WSFL [11], MS XLANG [12] and recently BPEL4WS [2]. Unfortunately, lack of support for semantically enriched information in these modeling languages leaves us a long way from seamless interoperation. Researchers in the Semantic Web community have taken up this challenge proposing top-down approaches to achieve aspects of Web Service interoperation [5]. Different Semantic Web and Semantic Web Services technologies for example RDF, DAML, OWL, and now the OWL-S ontology are result of efforts in this direction.

Ontology is a set of concepts, their properties, and relationship between them. Ontologies provide the building blocks for expressing semantics in a well-defined manner [7]. Where as OWL-S provides an ontology developed for web services and consists three types of knowledge: *Profile*, *Process Model* and *Grounding* [3].

“Service Profile” provides semantically enriched information about the capabilities of a service and what a service is doing. “Service Profile” specifies *inputs* required by a service and *outputs* generated by a service, pre-conditions that need to be true for using the service and effects that service will produce in surrounding world after its execution.

Rather than a program that can be executed, a “Process Model” is specification of ways a client may interact with a service. A “Process Model” can have one or more “Simple”, “Atomic” and “Composite Processes”. An “Atomic Process” is a description of a service that can be executed in single step and expects a message as an input and may returns a message in response as an output. A “Composite Process” maintains the state of the process. A “Composite Process” may consist of sub Composite or Atomic Processes. “Simple Processes” are non-invokeable processes and have no grounding, but like Atomic Processes they can be executed in single step.

“Grounding” specifies how to access a service. Technical details, for example, communication protocols, message formats, port numbers used to contact the service, are specified in “Grounding”. Normally the “Grounding” suffices to express how the components of a message are bundled, i.e. how inputs are put together to make a message to a service, and how replies are disassembled into the intended outputs [3].

2.3 OWL-S API

OWL-S API provides with java APIs for programmatic access to read, execute and write OWL-S service descriptions. The API provides an Execution Engine that can invoke Atomic Processes that have WSDL [4] or "Universal Plug and Play Language" (UPnP) [13] groundings, and Composite Processes that uses OWL-S Control Constructs e.g. Sequence, Split etc [8]. OWL-S's exchange syntax is RDF/XML and many processors work with an RDF based model, in part, to facilitate the smooth integration of OWL-S service descriptions with other Semantic Web knowledge bases. However working with the RDF triples directly can be quite cumbersome and confusing and the OWL-S API was designed to help programmers to access and manipulate OWL-S service descriptions programmatically [8]. We have also implemented the use of OWL- S API in our tool to write the OWL-S ontology, for the BPEL process model, according to mapping specifications discussed in the next section.

3 Mapping Specification

In this section we shall discuss the mapping from BPEL process to OWL-S ontology. We shall also discuss the mapping criteria used for mapping in areas where specifications does not support direct mapping, for example the "assignment" activity in BPEL has no equivalent Control Construct in OWL-S etc.

3.1 Overview

Figure 2 gives an overview of mapping specifications. Figure 2 shows that BPEL primitive activities are mapped to OWL-S "Perform" statement to perform the relevant Atomic Processes. Also if a primitive activity is an I/O activity (communicating with the outer world) then this activity is used to create the "Profile" of the resulting OWL-S service. Also the figure 2 shows that BPEL structured activities are mapped to relevant OWL-S control constructs. On the basis of this mapping overview the next section will discuss the mapping in more detail.

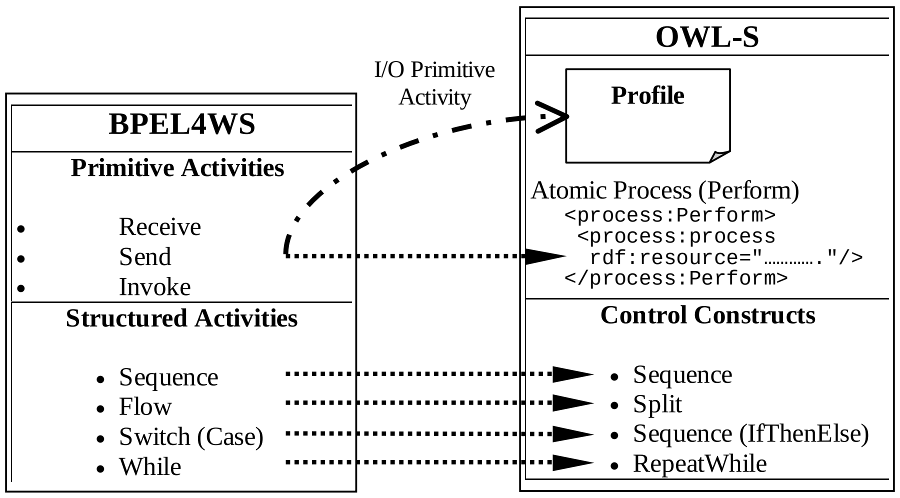


Fig.2. Overview of mapping specifications.

3.2 Atomic and Composite Processes

Beyond the BPEL specifications the OWL-S provides three kinds of processes, "Simple Processes", "Atomic Process" and "Composite Process". Where as BPEL has two kind of processes i.e. "Abstract Processes" and "Executable Processes". "Abstract Processes" provide means of synchronization with other processes at various level of granularity for the purpose of planning and reasoning [6]. "Simple Processes" in OWL-S also play the same role as BPEL "Abstract Processes" by providing a level of abstraction. Since to keep the complexity of work within limitations, in the current version synchronization between processes is not supported therefore we restrict ourself on the mapping of “Executable Processes” to “Atomic and Composite Processes”.

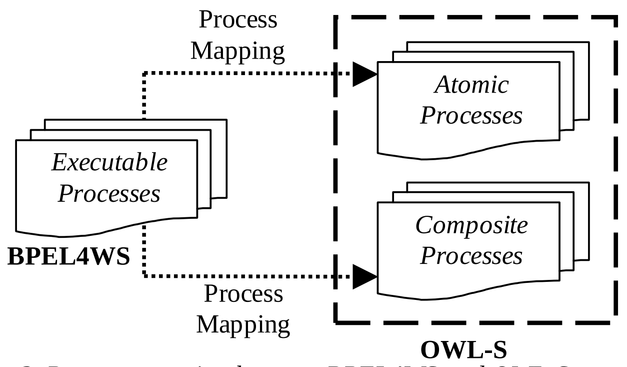


Fig.3. Process mapping between BPEL4WS and OWL-S processes.

3.2.1 Atomic Processes

“Atomic Process” corresponds to an action a service can perform in a single interaction and which can be executed in a single step (by sending and receiving appropriate messages). Also Atomic Processes have no sub-process.

BPEL process gives flow information of different activities in a business process. Where as messages exchanged between partners, port types and partner links, expressing business partners and relation between partners are expressed in BPEL corresponding WSDL file. A business process interacts with partner services through interfaces supported by corresponding web services. “Operations” supported by partner services (WSDL services) can be used to perform some specific task by sending them an input message and probably receiving some output message. Like an operation supported by a web service, an “Atomic Process” in OWL-S is a process that can perform some action in a single step. Therefore partner web services (WSDL Services) are parsed and corresponding Atomic Processes (with Profile, Process Model and Grounding) are created for each supported operation. *Grounding* of each Atomic Process specifies the real Web Service (WSDL service), so that it can be invoked in OWL-S service to perform some specific task.

For more clarification consider the “TranslationAndDictionary” process example (available with our tool). The example contains a BPEL file and relevant WSDL file and two WSDL services (“DictionaryService.wsdl” and “TranslatorService.wsdl”, providing functions “getMeaning” and “getTranslation” respectively). The figure below shows the partner link for the interacting web service in the BPEL’s corresponding WSDL file.

```
<plnk:partnerLinkType name="Dictionary_Ser_PortType">
  <plnk:role name="portRole">
    <plnk:portType name="q1:DictionaryPortType" />
  </plnk:role>
</plnk:partnerLinkType>
```

Fig.4. “Partner Link” in BPEL corresponding WSDL file showing interaction with “Dictionary Service”.

Figure below (fig. 5) shows supported operation “getMeaning” in the “DictionaryService.wsdl” file.

```
<wsdl:portType name="DictionaryPortType">
  <wsdl:operation name="getMeaning">
    <wsdl:input message="tns:DictionaryRequest" />
    <wsdl:output message="tns:DictionaryResponse" />
  </wsdl:operation>
</wsdl:portType>
```

Fig.5. WSDL operation ”getMeaning” to be mapped to OWL-S Atomic Process “getMeaningProcess”.

So according to specifications, the tool creates Atomic Process “getMeaningProcess” for operation “getMeaning”. Similarly all partner services (WSDL services) are explored and Atomic Processes “OWL files” are created for each supported operation, so that these “OWL files” can be used as “Atomic Processes” to perform some specific task (being they have real grounding to be invoked). Also, tool will not be able to create the Atomic Process, if the WSDL service would not be accessible on network.

3.2.2 Primitive Activities and Atomic Processes

As a logical equivalent of the OWL-S “Perform” statement used to perform an “Atomic Process”, “BPEL” has “Primitive Activities” e.g. “Receive”, “Invoke” and “Reply” activities used to perform some specific operation by sending and receiving appropriate messages. “Receive” activity is used to receive some message from some resource e.g. from some web service. “Reply” activity is used to send a message in response to some “Receive” activity. Where as the “Invoke” activity represents combine behavior of both, “Receive” and “Reply” activities i.e. it invoke a service by sending it an input message and then receive a message as an output of web service operation. Figure 6 shows “Invoke” activity statement in a BPEL process.

```
<invoke partnerLink="Dictionary_Ser_Port" portType="q3:DictionaryPortType " operation=
  getMeaning" inputVariable= ”Message_1_To_Dic_Service”
  outputVariable= ”Message_1_From_Dic_Service”/>
```

Fig.6. “Invoke” activity sending and receiving message from “Dictionary Service”.

In above statement “Invoke” activity is sending an input message “Message_1_To_Dic_Service” to perform operation “getMeaning” and receiving a message "Message_1_From_Dic_Service" as a response of “getMeaning” operation. Like OWL-S “Atomic Processes”, BPEL primitive activities can be used to perform some specific operation in a single step and they have no sub activity to be performed. We can map these BPEL primitive activities to OWL-S “Perform” statement to perform the relevant “Atomic Process”. The above primitive activity (Invoke) statement can be mapped to OWL-S “Perform” statement to perform the OWL-S Atomic Process “getMeaningProcess” (as shown in figure 7).

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

Fig.7. OWL-S “perform” statement to perform Atomic Process “getMeaningProcess”.

Where as the “getMeaningProcess” is Atomic Process that can be performed in single step and is supported by the “getMeaning.owl” file created in section 3.2.1.

3.2.3 Structured Activities and Composite Processes

“Structured Activities” in BPEL describe the order in which set of child primitive or structured activities will be performed e.g. “Sequence” structured activity describes that the child primitive or structured activities within a “Sequence” activity will be performed in a sequence. Similar to BPEL structured activity “Sequence”, the OWL-S has “Sequence” control construct that is used to perform the child Atomic or Composite Processes in a sequence. Due to their logical matching behavior, BPEL structured activities are mapped to OWL-S control constructs with in an OWL-S Composite Process.

Against an “Atomic Process”, a “Composite Process” is not a behavior a service will do, but a behavior (or set of behaviors) the client can perform by sending and receiving a series of messages [3]. A “Composite Process” may consist of sub Atomic or Composite Processes. Like BPEL structured activities the OWL-S Composite Process defines how sub Atomic Processes or Composite Processes within a Composite Process are performed by using the OWL-S control constructs.

Let us consider the code below (taken from “TranslationAndDictionary” example available wit tool), describing that structured activity “Sequence” has two sub primitive activities that can be performed in a sequence in a BPEL process.

```
  <sequence>
        ..…
            <invoke partnerLink="To_Translation_Service_Port_1" portType=
"q2:TranslatorPortType” operation=”getTranslation” inputVariable=
"Message1_To_Translation_Service" outputVariable="Message1_From_Translation_Service"/>
        ………
        …….. 
            <invoke partnerLink="Dictionary_Ser_Port" portType="q3:DictionaryPortType"
   operation="getMeaning" inputVariable=”Message_1_To_Dic_Service”
   outputVariable="Message_1_From_Dic_Service" />
        …………….
</sequence>
```

Fig.8. “Sequence” activity having child primitive activities (Invoke).

Figure 9 shows the mapping of BPEL structured activity “Sequence” to the OWL-S control construct “Sequence” which perform two Atomic Processes “getTranslationProcess” and “getMeaningProcess” in a sequence. Where two “perform” statements are result of mapping of two “Invoke” activities with in BPEL “Sequence” activity (as discussed in section 3.2.2).

```
    <process:composedOf>
     <process:Sequence>
       <process:components>
         <process:ControlConstructList>
           <list:first>
             <process:Perform>
               <process:process
rdf:resource="http://examples.org/DummyURI.owl#getTranslationProcess"/>
            </process:Perform>
               ……………………………………………
             <process:ControlConstructList>
               <list:first>
                 <process:Perform>
                   <process:process
rdf:resource="http://examples.org/DummyURI.owl#getMeaningProcess"/>
                 </process:Perform>
               </list:first>
                   ………….
                   ………….
             </process:ControlConstructList>
           </list:rest>
         </process:ControlConstructList>
       </process:components>
     </process:Sequence>
    </process:composedOf>
```

Fig.9. OWL-S “Sequence” Control Construct performing two Atomic Processes in a sequence.

“Flow” activity in BPEL is used to create concurrency and synchronization between sub-activities and has an equivalent OWL-S control construct “Split”. In OWL-S, “Split” control construct is used for concurrent execution of process components and “SplitJoin” control construct is used to define processes that have partial synchronization. But in current version we have implemented the mapping of “Flow” activity to “Split” control construct and synchronization between process components is not yet supported. Consider the following example (Demo example) code (figure 10) showing the BPEL “Flow” activity having “Sequence” and “While” sub-activities.

```
  <flow name="Main_Flow">
       <sequence name="CREATION_SEQUENCE">
       ………..
              <while name="CREATION_WHILE" condition= ……/">
              …………
              </while>
       …….....
       </sequence>
  </flow>
```

Fig.10. BPEL “Flow” activity having sub “Sequence” and “While” activities.

Mapping of BPEL “Flow” activity to OWL-S control construct “Split” is shown in figure 11. Where “Repeat-While” is OWL-S equivalent control construct for BPEL “While” activity (as discussed later in this paper).

```
<process:composedOf>
  <process:Split>
    <process:components>
      <process:ControlConstructBag>
        <list:first>
          <process:Sequence>
            <process:components>
              <process:ControlConstructList>
                <list:first>
                  <process:Repeat-While>
                    <process:whileProcess>
                    ..................................
                    ..................................
          </list:first>
         </process:ControlConstructBag>
       </list:rest>
      </process:ControlConstructBag>
    </process:components>
  </process:Split>
</process:composedOf>
```

Fig.11. BPEL “Flow” activity mapped to OWL-S “Split”.

“Switch” structured activity supports conditional behavior, supporting conditional branches defined by the “Case” element and having optional “otherwise” branch. The BPEL “Switch” activity is mapped to OWL-S sequence “Sequence” of “IfThenElse” control constructs. Where each “Case” is mapped to an “IfThenElse” control construct and “Otherwise” part of “Case” statement is mapped to “Else” part of “IfThenElse” control construct. The figure below (figure 12) shows BPEL “Switch” activity having a “Case” element and “Case” condition statement.

```
<switch name="Solvency_Switch">
     <case condition="bpws:getVariableData('status', 'status', '//type')=3">
            <sequence name="Solvency_Sequence">
            ………..
            </sequence>
     </case>
</switch>
```

Fig.12. BPEL “Switch” activity statement.

Figure 13 shows the mapping of BPEL “Switch” activity to OWL-S sequence of “IfThenElse” control constructs.

```
<process:Sequence>
  <process:components>
    <process:ControlConstructList>
      <list:first>
        <process:If-Then-Else>
         <process:then>
           <process:Sequence>
            ..............................

           </process:Sequence>
         </process:then>
        </process:If-Then-Else>
      </list:first>
    </process:ControlConstructList>
  </process:components>
</process:Sequence>
```

Fig.13. Mapping of “Switch” activity to OWL-S sequence of “IfThenElse” Control Constructs.

Also, “While” activity in BPEL is mapped to “RepeatWhile” control construct in OWL-S, to repeatedly perform a specific process. Let us see the example code below (figure 14) showing the BPEL “While” activity with “While” condition and having sub activity “Sequence”.

```
<while name="CREATION_WHILE" condition="bpws:getVariableData('status',
 status','//type')=-1 or bpws:getVariableData('status', 'status', '//type')=-2"
xmlns:demo="urn:demo:MarketplaceService xmlns:svc="http://bpeldemo.ibm.com/services/">

       <sequence name="While_Sequence">
       ………………………
       </sequence>
</while>
```

Fig.14. BPEL “While” activity with “While Condition” and having “Sequence” activity as sub-activity.

Figure 15 shows the mapping of BPEL “While” activity to OWL-S “Repeat-While” control construct.

```
  <process:Repeat-While>
   <process:whileProcess>
    <process:Sequence>
     <process:components>
      …………………………………….
     </process:components>
    </process:Sequence>
   </process:whileProcess>
  </process:Repeat-While>
```

Fig.15. BPEL “While” activity mapped to OWL-S “RepeatWhile” Control Construct.

Conditions: Since, expressions in OWL-S “IfThenElse” “Condition” are not supported. Also there exist no appropriate way to map the statement part “bpws:getVariableData(… ” to OWL-S, that’s why in this version automatic mapping of “Condition” is not fully supported. But information about the “Condition” statement can be found in “OWL-S Process Ontology” e.g. “Demo.owl” (in case of our Demo example project). This condition statement can be used to manually create the “Condition” i.e. SWRL expressions for conditions in OWL-S.

3.3 Data Flow

Data flow is an important part of the OWL-S specifications. OWL-S specifications deal with the data flow at two levels, at one level of data flow we deal with passing data from one Atomic Process to another. At second level we define the inputs and outputs of Composite Process by specifying the input of a sub-process as an input of the Composite Process and possibly specifying the output of Composite Process as an output derived from some sub-process.

According to BPEL and OWL-S specifications there are no logically equivalent activities in BPEL and OWL-S for mapping and defining the data flow. Therefore, here we shall discuss the criteria we have implemented for defining data flow between Atomic and Composite Processes.

```
<invoke partnerLink="To_Translation_Service_Port_1" portType= “q2:TranslatorPortType
 ” operation="getTranslation" inputVariable="Message1_To_Translation_Service"
 outputVariable="Message1_From_Translation_Service" />
  <assign>
   <copy>
     <from variable="Message1_To_Translation_Service" part=”getTranslationResult”/>
     <to variable="Message_1_To_Dic_Service" part="inputString" />
   </copy>
  </assign>
<invoke partnerLink="Dictionary_Ser_Port" portType="q3:DictionaryPortType"
 operation="getMeaning" inputVariable="Message_1_To_Dic_Service"
 outputVariable="Message_1_From_Dic_Service" />
```

Fig.16. BPEL “Assignment” activity assigning the value between two message parts.

Logical behavior of BPEL "Assignment" activity shows that it can be used between two primitive activities to assign an output message or message parts (in case of complex message types) of first activity, as an input message or message part for next activity. Figure 16 shows an example of BPEL "Assignment" activity statement (taken from "TranslationAndDictionary"). According to mapping specifications, "Invoke" activities before and after "Assignment" activity are mapped to OWL-S "Perform" statement to perform Atomic Process "getTranslationProcess" and "getMeaningProcess" respectively (as discussed in section 3.2.2). "Assignment" activity between these two "Invoke" activities is assigning the value of message part "getTranslationResult" of the message variable "Message1_From_Translation_Service" to message part "inputString" of message variable "Message_1_To_Dic_Service". Therefore while treating the "Assignment" activity, message part in the "<from>" part of "Assignment" activity is searched in output parameters of relevant Atomic Process i.e. "getTranslationProcess" Atomic Process (where "getTranslationProcess" is supported by "getTranslation.owl" file created in section 3.2.1). Similarly message part in the "<to>" part of assignment activity is searched in the list of input parameters for "getMeaningProcess" Atomic Process. If both of these message parts are found as input and output of relevant Atomic Processes then data flow is created between these Atomic Processes, describing the binding between specified variables (message parts). Figure 17 gives a simplified view of criteria used to create the data flow between two atomic processes.

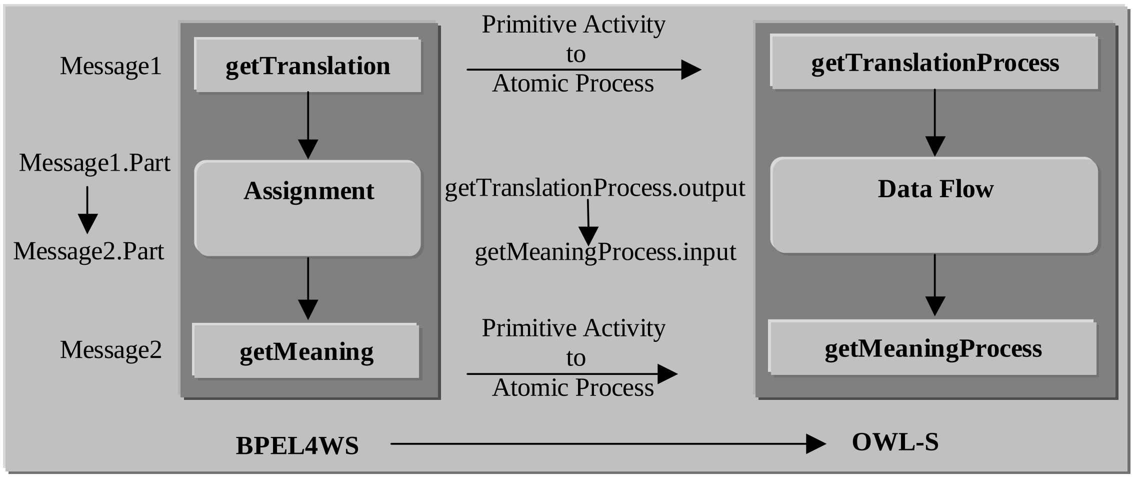


Fig.17. Showing use of BPEL "Assign" activity to create data flow in OWL-S.

Now "Data Flow" for Composite Processes: A BPEL process can have multiple interfaces, available as "Receive", "Reply" or "Invoke" activities, supported by port type, operation and input or output messages defined in the BPEL corresponding WSDL file. Such activities can be used to receive and send a message as an input and output of a BPEL process. Therefore among these multiple input and output options, input of the first "Receive" primitive activity receiving a message from the outer world is defined as input for the OWL-S Composite Process. If a "Receive" activity has corresponding "Reply" activity then message variable of this "Reply" activity is used to set the output of the OWL-S Composite Process. In other case first primitive activity e.g. any "Invoke" activity sending some message to the outer world is taken as an output activity to define the output of the OWL-S Composite Process. Also a primitive activity is declared as an Input/Output activity if the BPEL corresponding WSDL file supports its port type and operation. For example the "Receive" activity statement shown below (taken from "TranslationAndDictionary" example) is used to create input of the OWL-S Composite Process.

```
  <receive            partnerLink="Input_Port"           portType="q1:Input_PortType"
operation="Operation_1" variable="Input_Message" createInstance="yes" />
```

Since this receive activity don't has corresponding "Reply" activity (in case of our example), and BPEL process is sending its output to the outer world by using the "Invoke" activity, having message "Message_2_From_Translation_Ser" as output message. Therefore this activity is used to create output of the OWL-S Composite Process.

```
  <invoke            partnerLink="Output_Port"           portType="q1:Output_PortType"
operation="Operation_1" inputVariable="Message_2_From_Translation_Ser" />
```

Off course in case of a complex message, message parts are considered to create the input and output of the OWL-S Composite Process and are used to create the profile of the resulting OWL-S Service (discussed in next section).

4 OWL-S Suite Of Ontologies

This section explains the creation of the OWL-S suite of ontologies. Figure 18 gives a simplified view of mapping process. Figure 18 shows that BPEL4WS2OWL-S tool maps the input BPEL and relevant WSDL files to complete OWL-S suite of ontologies i.e. Profile, Process Model and Grounding.

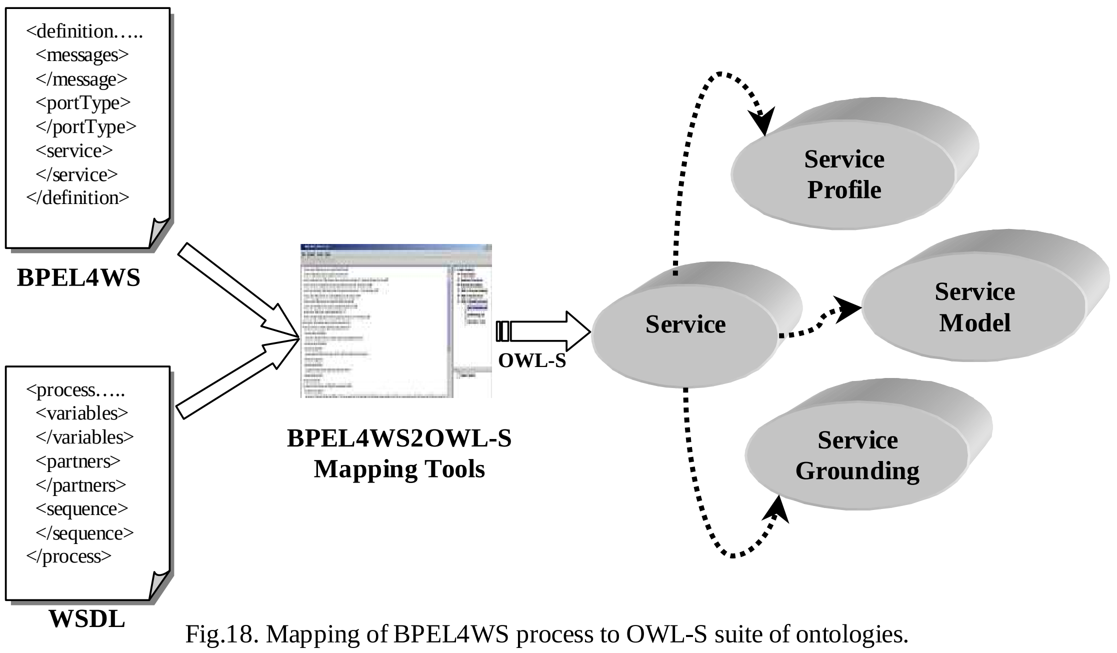


Fig.18. Mapping of BPEL4WS process to OWL-S suite of ontologies.

4.1 Profile

On the basis of the input and output created for the OWL-S Composite Process (discussed in data flow section), the service profile is created for inputs and outputs of the OWL-S service e.g. the code below shows the profile for the "TranslationAndDictionary" example.

```
  <profile:Profile rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestProfile">
    <profile:hasInput
     rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#inputStr"/>
    <profile:hasInput
     rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#inputLang"/>
    <profile:hasInput
     rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#outputLang"/>
    <profile:hasOutput
     rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#return"/>
    <profile:textDescription>This Profile is created by BPEL2OWLS Tool
    </profile:textDescription>
    <rdfs:label>BPEL2OWLS Profile</rdfs:label>
    <service:presentedBy
     rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestService"/>
  </profile:Profile>
```

Fig.19. Service Profile for "TranslationAndDictionary" process example.

4.2 Process Model

In section 3 we have discussed in detail about specification for mapping from BPEL process model to OWL-S process model. OWL-S process model is combination of these BPEL mapped activities in OWL-S Composite and Atomic Processes with relevant data flow.

4.3 Grounding

"Grounding" of a service specifies details about how to access a service. In case of our mapped OWL-S service, "Grounding" of OWL-S service specifies the location of the grounding of each Atomic Process (as shown in fig. 20). Also concrete messages are specified explicitly in grounding. Off course mapping is not able to define the xsltTransformation [16] for complex messages. Web Services Description Language (WSDL) service, being XML format for describing network services is referred in grounding of each Atomic Process to have access to the original implementation of WSDL service.

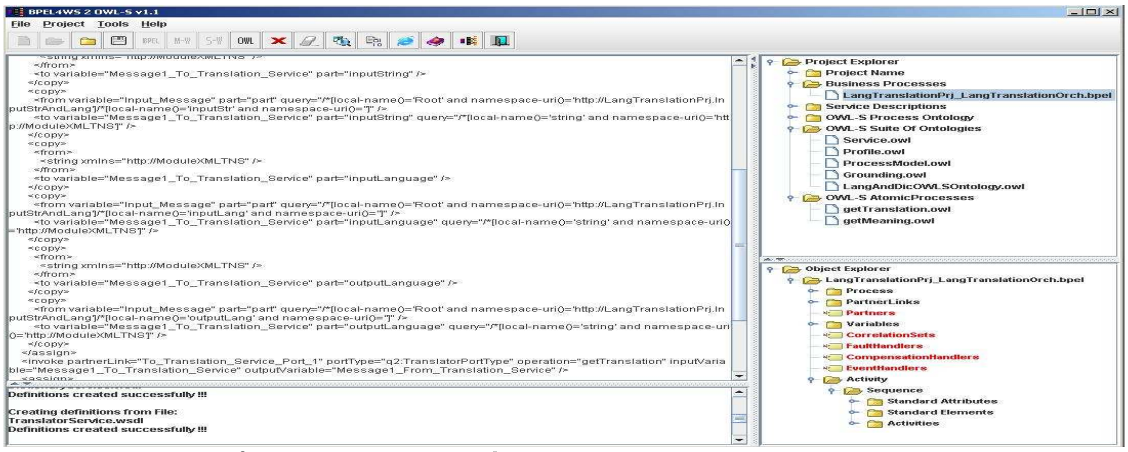
```
<grounding:WsdlGrounding
  rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestGrounding">
 <service:supportedBy
   rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestService"/>
  <grounding:hasAtomicProcessGrounding
   rdf:resource="http://examples.org/DummyURI/getMeaning.owl#
   getMeaningAtomicProcessGrounding"/>
  <grounding:hasAtomicProcessGrounding
   rdf:resource="http://examples.org/DummyURI/getTranslation.owl#
   getTranslationAtomicProcessGrounding"/>
</grounding:WsdlGrounding>
```

Fig.20. Service Grounding for "TranslationAndDictionary" example.

In figure 20 "getMeaningAtomicProcessGrounding" and "getTranslationAtomicProcessGrounding" are groundings for "getMeaningProcess" and "getTranslationProcess" defined in "getMeaning.owl" and "getTranslation.owl" created in section 3.2.1.

5 Prototype of Mapping Tool

The tool provides very easy to use environment. As an input for mapping, tool requires an input BPEL file and relevant WSDL files. Where as WSDL files are further divided in to *Master* and *Slave* WSDL files. Master WSDL file refers to BPEL corresponding WSDL file containing information about partner links and messages exchanged between interacting partners (web services) etc. and slave WSDL files correspond to WSDL web services interacting with business process.



```
    </from>
    <to variable="Message1_To_Translation_Service" part="inputString" />
   </copy>
   <copy>
    <from variable="Input_Message" part="part" query="/*[local-name()='Root' and namespace-uri()='http://LangTranslationPrj.In
putStrAndLang']/*[local-name()='inputStr' and namespace-uri()='']" />
    <to variable="Message1_To_Translation_Service" part="inputString" query="/*[local-name()='string' and namespace-uri()='htt
p://ModuleXMLTNS']" />
   </copy>
   <copy>
    <from>
     <string xmlns="http://ModuleXMLTNS" />
    </from>
    <to variable="Message1_To_Translation_Service" part="inputLanguage" />
   </copy>
   <copy>
    <from variable="Input_Message" part="part" query="/*[local-name()='Root' and namespace-uri()='http://LangTranslationPrj.In
putStrAndLang']/*[local-name()='inputLang' and namespace-uri()='']" />
    <to variable="Message1_To_Translation_Service" part="inputLanguage" query="/*[local-name()='string' and namespace-uri()
='http://ModuleXMLTNS']" />
   </copy>
   <copy>
    <from>
     <string xmlns="http://ModuleXMLTNS" />
    </from>
    <to variable="Message1_To_Translation_Service" part="outputLanguage" />
   </copy>
   <copy>
    <from variable="Input_Message" part="part" query="/*[local-name()='Root' and namespace-uri()='http://LangTranslationPrj.In
putStrAndLang']/*[local-name()='outputLang' and namespace-uri()='']" />
    <to variable="Message1_To_Translation_Service" part="outputLanguage" query="/*[local-name()='string' and namespace-uri
()='http://ModuleXMLTNS']" />
   </copy>
  </assign>
  <invoke partnerLink="To_Translation_Service_Port_1" portType="q2:TranslatorPortType" operation="getTranslation" inputVaria
ble="Message1_To_Translation_Service" outputVariable="Message1_From_Translation_Service" />
```

```
Definitions created successfully !!!

Creating definitions from File:
TranslatorService.wsdl
Definitions created successfully !!!
```

```
Project Explorer
  Project Name
  Business Processes
    LangTranslationPrj_LangTranslationOrch.bpel
  Service Descriptions
  OWL-S Process Ontology
  OWL-S Suite Of Ontologies
    Service.owl
    Profile.owl
    ProcessModel.owl
    Grounding.owl
    LangAndDicOWLSOntology.owl
  OWL-S AtomicProcesses
    getTranslation.owl
    getMeaning.owl

Object Explorer
  LangTranslationPrj_LangTranslationOrch.bpel
    Process
    PartnerLinks
    Partners
    Variables
    CorrelationSets
    FaultHandlers
    CompensationHandlers
    EventHandlers
    Activity
      Sequence
        Standard Attributes
        Standard Elements
        Activities
```

Fig.21. An overview of prototype mapping tool.

As a first step, tool requires to create a new project and adding the input BPEL and master and slave WSDL files in the project. Then we validate the project for input BPEL and WSLD files. This validation results in validation message for input files or error message, telling that one of the input files is not validated. If project validation is successful then we build the project. Building of project parses BPEL file and each WSDL file to create object view of corresponding files, to use in mapping. After successfully building the project, it is mapped to OWL-S, resulting in creation of “Atomic Processes” for each supported operation and OWL-S suite of ontologies. “Service Profile” specifies the semantically enriched information (enriched with dummy URIs that need to be changed with original ontological concepts according requirement) about inputs and outputs of the OWL-S service. “Process Model” controls the interaction with services and “Grounding” of OWL-S service describes how to access the mapped OWL-S service.

Tool presents user friendly and easy to use interface. Right upper window acts as a “Project Explorer” to explore project files (BPEL, WSDL and OWL files). Right lower window is “Object Explorer” which gives object view of BPEL, WSDL and data flow OWL files. Object view for OWL-S suite of ontologies is not supported in this version. But contents of these files can be seen in upper left window by double clicking the file in “Project Explorer”. While the left lower window is output window, which gives output messages of different actions, performed e.g. validation, building and mapping.

6 Related Work and Future Plan

Since OWL-S is not as much mature as BPEL e.g. equivalent of BPEL activities like *Assignment*, *Fault Handler*, *terminate* etc are not available in OWL-S for direct mapping from BPEL to OWL-S. Issues like “process:produce” Control Construct (used to create data flow) are under discussion on W3C. Information about pre and post-conditions is not available in BPEL (so that it can be used in mapping) and semantic information needed for mapping inputs and outputs to ontological concepts need to be manually changed with real ontological concepts in resulting OWL-S ontology. Therefore in these areas where mapping is partially supported or needs information to be added by the user, manually changing is also a time consuming and complex task and requires a user to be an expert of OWL-S. So at this stage our BPEL4WS2OWL-S tool needs constant updates with the upcoming versions of the related technologies. Secondly a tool is needed that can be used to develop required ontologies and an editor which can help in editing resulting OWL-S ontology with these ontological concepts more easily and ideally in a visual environment.

“Protégé” with its plug-in “OWL-S Editor” is an ideal environment that can be used to move forward. “Protégé” is an ontology development tool and “OWL-S Editor” is an editor that can be used to visually develop and edit OWL-S ontologies. “OWL-S Editor” is available as a plug-in for “Protégé”. So as a next step to our work we are planning to produce more consistent mapping from BPEL to OWL-S. Especially we shall try to implement support for synchronization between process components and to fully support the “Conditions”. We shall try to make our tool fully compatible with “Protégé” and “OWL-S Editor”. We are also working to improve our tool as a BPEL4WS2OWL-S import plug-in for “Protégé” and “OWL-S Editor”, so that mapped OWL-S services can be directly imported in “OWL-S Editor” and can be edited (to add real ontological concepts for inputs and outputs and for adding pre and post conditions etc.) in a visual environment.

7 Conclusion

BPEL4WS (BPEL) is being used by business process modeling community for modeling business processes. But lack of semantics in BPEL keep us away from business process automation. Where as OWL-S provides with computer understandable semantics about the capabilities of a service that can be used to achieve the aim of process automation by mapping BPEL processes to OWL-S ontology. Therefore to overcome the semantic limitations of BPEL we have presented our work to map BPEL processes to OWL-S ontology, to use the benefits of process modeling capabilities and semantic capabilities of OWL-S. Our BPEL4WS2OWL-S mapping tool generates the OWL-S Atomic Process with its Profile, Process Model and Grounding, for each supported operation. Also BPEL process is mapped to OWL-S ontology (Profile, Process Model and Grounding) according to above discussed mapping specifications.

After successful mapping, end user needs to change semantic information for inputs and outputs in the service profile with some real ontological concepts. In resulting OWL-S service pre and post conditions can be added (if required) and data flow and conditions (which are partially supported in this version) can be completed by editing resulting OWL-S service in some editor e.g. OWL-S Editor. Once missing information is added in resulting OWL-S service, it can be used to execute with the OWL-S API (we are talking about executing the OWL-S service with the OWL-S API because till now OWL-S API is only a way to execute an OWL-S services). Also “Grounding” for each OWL-S Atomic Process refers to real WSDL service so that WSDL service can be accessed over network and relevant operation can be performed. Also in case of complex message types user needs to define the XSLT transformation manually.

References

1. WISEINFO: [online] Available http://wiseinfo.info/web-service.htm
2. Business Process Execution Language for Web Services Version 1.1. 5th May 2003. [online] Available ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.
3. OWL-S: Semantic Markup for Web Services. [online] Available http://www.daml.org/services/owl-s/1.1/overview/.
4. Web Services Description Language (WSDL) 1.1. [online] Available http://www.w3.org/TR/wsdl.
5. Daniel J. Mandell and Sheila A. McIlraith: Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. Proceedings of the Second International Semantic Web Conference 2003.
6. Jun Shen and Yun Yang: Experiences with Integrated E-Business Process Specifications (in review for Journal).
7. Gerald C. Gannod, Raynette J. Brodie and John T.E.Timm: An Interactive Approach for Specifying OWL-S Groundings. Proceedings of the IEEE EDOC Enterprise Computing Conference, Sept. 2005.
8. Evren Sirin: OWL-S API. [Project Home Page] Available http://www.mindswap.org/2004/owl-s/api/.
9. Jun Shen and Yun Yang: BPEL2OWL-S1.1 [online] Download Page http://www.it.swin.edu.au/centres/cicec/bpel2owls.htm
10. UDDI Version 3.0.2: UDDI Specifications Technical Committee Draft, Dated 20041019. [online] Available http://www.uddi.org/specification.html
11. Frank Leymann. Web Services Flow Language (WSFL 1.0) May 2001. [online] Available http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
12. Judith M. Myerson: Web Service Architecture. Published by Tect, USA. [online] Available http://www.webservicesarchitect.com/content/articles/webservicesarchitectures.pdf.
13. http://upnp.org/
14. A First Overview of BPEL4WS. January 25, 2.005. [online] Available http://jroller.com/page/coreteam/Weblog?catname=%2FWorkflow
15. Polar Lake, White Paper, Automating Business Process Management With BPEL and XML [online] Available http://www.polarlake.com/en/assets/whitepapers/AutomatingBusinessProcess Management_BPEL_XML.pdf
16. XSL Transformations (XSLT) : [online] Available http://www.w3.org/TR/xslt.
17. OWL-S’ Relationship to Selected Other Technologies [online] Available http://www.daml.org/services/owl-s/1.1/related.html.