Expressing Business Process Models as OWL-S Ontologies

Muhammad Ahtisham Aslam[1], Sören Auer[1,2], Jun Shen[3], Michael Herrmann[4]

[1] Betriebliche Informationsysteme, Universität Leipzig, Germany
{aslam,auer}@informatik.uni-leipzig.de
[2] Computer and Information Science Department, University of Pennsylvania, USA
auer@seas.upenn.edu
[3] School of IT and CS, University of Wollongong, Australia
jshen@uow.edu.au
[4] DaimlerChrysler AG, Sindelfingen Germany
michael.hm.herrmann@daimlerchrysler.com

Abstract. BPEL4WS is a well-established business process standard that can be used to orchestrate service-based workflows. However, the rapid growth and automation demands of e-business and grid applications require BPEL4WS to provide enhanced semantic annotations to achieve the goal of business processes automation. Here, OWL-S (OWL for Web Services) is designed to represent such kind of semantic information. Furthermore, there exists a similarity in the conceptual model of OWL-S and BPEL4WS that can be employed to overcome the lack of semantics in BPEL4WS by mapping the BPEL4WS process model to the OWL-S suite of ontologies. The mapped OWL-S service can be used to increase flexibility and to automate BPEL based grid scenarios even further. This is achieved by dynamic discovery, composition and invocation of OWL-S services, for example within e-business and grid environments. Hence, the aim of this paper is to establish a mapping from the BPEL process model to the complete OWL-S suite of ontologies. We present a mapping strategy and a tool supporting this strategy. This allows the semantic annotation of workflows defined as BPEL4WS processes to enable the automation of a variety of e-business tasks.

1 Introduction

Combining Web Services and Grids is a promising way to leverage existing work in both business and scientific environments. BPEL4WS is a standard language to define workflows as a combination of Web Services interactions. BPEL4WS combines the power of graph oriented (WSFL) language and procedural workflow language (XLANG). Strong support for Web Services in BPEL makes it very attractive workflow language for the use in Grid environments. A BPEL workflow is a Web Services orchestration and can act itself as a Web Service. A BPEL process seen as a Web Service can be combined with other Web Services to create new ones. But such an interaction and combination of a BPEL process with other Web Services has the same syntactical limitations as the syntax of the base Web Services. With such syntactical limitations, processes defined by using a workflow language (e.g. BPEL4WS) cannot be automated to meet the demands of rapidly growing e-business world.

We consider a B2B interaction in which two business partners have defined their business processes as BPEL processes. Integrating these processes to perform some complex business task requires prior agreements between both business partners. But such kind of manual discovery of compeers and making prior agreements between them is not an efficient and flexible approach for the automation and integration of BPEL processes. Even exporting such a workflow, as a Web Service is not enough for the purpose of automation of BPEL based Grid service scenarios. Mapping a business process (BPEL process) to an OWL-S service description will support business process automation by enabling semantic base discovery, invocation and composition by semantic enabled systems.

In this paper we present a mapping strategy to map BPEL processes to the OWL-S ontology and its prototypical implementation as a tool[1] (BPEL4WS2OWL-S Mapping Tool) that can be used to map BEPL4WS processes to the complete OWL-S suite of ontologies. Our work is an extension and improvement to work by CICEC Lab [8] that has the following drawbacks:

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

- Atomic processes are not supported according to OWL-S specifications.
- Atomic processes cannot be invoked and executed in resulting OWL-S service.
- Complex message types are not supported.
- No data binding is supported between atomic processes.
- Data flow between atomic processes is defined in separate OWL file, which is not according to OWL-S specifications.
- Mapping does not support the OWL-S specification.
- *Profile* and *Grounding* ontologies are not supported.

Since, *Profile* ontology is not supported by [8], therefore, mapped OWL-S *Process Model* ontology cannot describe its capabilities so that it can be discovered on the basis of matching semantics. Also, unavailability of *Grounding* ontology results in communication restrictions with other semantic enabled services. [8] Supports only mapping from BPEL process model to OWL-S *Process Model* ontology and this mapping also have many drawbacks and limitations as discussed above. Therefore, work by CICEC Lab (Jun Shen and Yun Yang) needs to be enhanced and improved to support more consistent mapping to *Process Model* ontology. Also, such a mapping will have real world effects if mapping from BPEL process model to complete OWL-S suite of ontologies (*Profile, Process Model* and *Grounding*) will be supported.

Our work is an effort to achieve more consistent mapping, resulting in full OWL-S suite of ontologies (*Profile, Process Model* and *Grounding* ontologies). Our work supports complex message types to create more consistent data flow. 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 suite of ontologies (*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 specifications (for mapping BPEL4WS process model to OWL-S suite of ontologies) and mapping of BPEL process model to OWL-S *Process Model* ontology. Section 4 describes how *Profile* and *Grounding* ontologies are created during mapping process. Section 5 descries the architecture and user interface of our mapping tool. In section 6 we discuss the limitations of our work. Section 7 discusses the related work and sections 8 draw the conclusion of our work and discuss the future plans.

2 Background

2.1 BPEL4WS

Different workflow languages (e.g. WSFL [9], MS XLANG [10] and BPEL4WS [2]) were developed to bring the use of Web Services to a higher level by composing them together to perform complex business tasks that a single Web Service cannot perform. Among these languages BPEL got more attraction of the community for modeling business processes as a composition of Web Services. A BPEL process is modeled by using BPEL *primitive and structured activities*. Figure 1 gives an overview of BPEL activities. Figure 1 shows that BPEL has two kinds of activities *basic* or *primitive activities* and *structured activities*. *Primitive activities* (e.g. *invoke, receive* and *reply*) are used to model interaction between business partners, where as workflow in a BPEL process model is modeled by using the *structured activities* (e.g. *sequence, flow* etc.) BPEL *primitive activities* can be nested in *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, which 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 [11]. But effective dynamic service binding cannot be performed by solely matching WSDL messaging interfaces.

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

Fig.1. BPEL4WS activities table.

Since, 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 [12].

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 [9], MS XLANG [10] 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 [11]. OWL-S, OWL ontology for Web Services, is developed to provide Web Services semantics and consists of three types of knowledge: *Profile, Process Model* and *Grounding*.

Profile provides semantically enriched information about capabilities of a service and what a service is doing. *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 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 return a message in response as an output. A *composite process* maintains the state of the process. A *composite process* may consist of sub c*omposite* or *atomic processes. Simple processes* are non-invoke able 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 a Java API 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.) [14]. 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 [14]. We have also implemented the use of OWL- S API in our tool to write the OWL-S services, for the BPEL process model, according to mapping specifications discussed in the next section.

3 Mapping Specifications

In this section we discuss the mapping from BPEL process to the OWL-S ontology. We also discuss the criteria used for mapping in areas where specifications of the BPEL and the OWL-S do not support direct mapping. For example *Assignment* activity in BPEL has no equivalent control construct in OWL-S so that it can be directly translated to OWL-S control construct.

3.1 Overview

Figure 2 gives an overview of mapping specifications. Figure 2 shows that BPEL *primitive activities* are mapped to OWL-S *Perform* control constructs. If a *primitive activity* is an I/O activity (used to create the interface of the BPEL process) then this activity is used to create the *Profile* of the resulting OWL-S service. Also, 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 describes the mapping of BPEL process model to OWL-S *Process Model* ontology in more detail.

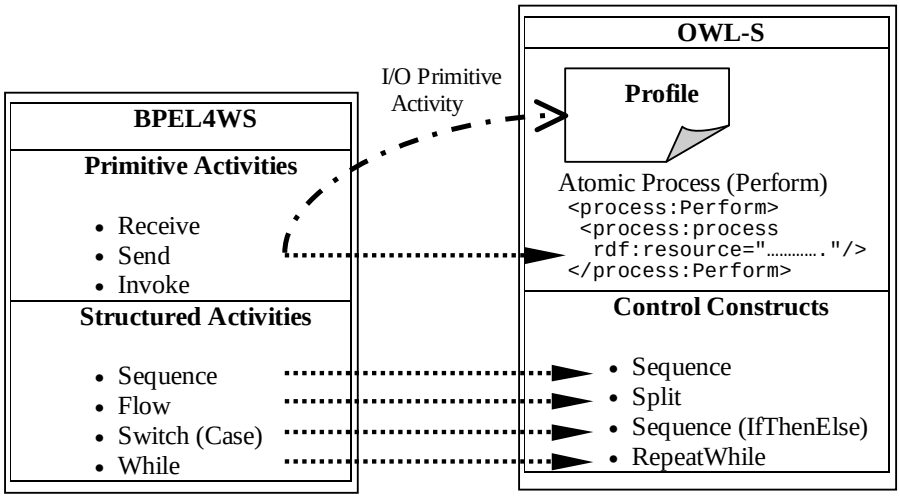

Fig.2. Overview of mapping specifications.

3.2 Mapping to OWL-S Process Model

OWL-S has three kinds of processes, *simple processes, atomic process* and *composite process*. Where as the BPEL have two kinds of processes, *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 [12]. *Simple processes* in OWL-S also play the same role as BPEL *abstract processes* by providing a level of abstraction. To keep the complexity of work within limitations, in the current version, synchronization between processes is not supported therefore, we restrict our self on the mapping of *executable processes* to *atomic* and *composite 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, an atomic process has 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's 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 (supported by that Web

Service operations) 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. Each *atomic process* is grounded with real Web Service (WSDL service), so that it can be invoked in resulting OWL-S composite service to perform some specific task.

For more clarification consider the "Translation And Dictionary" 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", having operations "getMeaning" and "getTranslation" respectively). The figure 3 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.3. "Partner Link" in BPEL's corresponding WSDL file showing interaction with "Dictionary Service".

Figure below (fig. 4) 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.4. WSDL operation "getMeaning" will be mapped to OWL-S atomic process "getMeaningProcess".

So according to specifications, the tool creates *atomic process* "getMeaningProcess" for Web Service operation "getMeaning". Similarly all partner services (WSDL services) are parsed and *atomic processes* "OWL files" are created for each supported operation. Also, tool will not be able to create the *atomic process* for a Web Service operation, if the WSDL service would not be accessible on network.

3.2.2 Primitive Activities and Atomic Processes. OWL-S *Perform* control construct can be used to perform an *atomic process.* BPEL *primitive activities* (e.g. *receive, reply* and *invoke*) can be used to perform a Web Service operation by sending and receiving appropriate messages. Due to their logical matching behavior we map BPEL *primitive activities* to OWL-S *Perform* control constructs to perform relevant atomic processes. *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 behaviour 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 5 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.5. Invoke activity sending and receiving message from"Dictionary Service".

Figure 5 shows an *invoke* activity statement which sends an input message "Message_1_To_Dic_Service" to perform "getMeaning" operation and receives a message "Message_1_From_Dic_Service" as a response of "getMeaning" operation. Like OWL-S *atomic processes*, the 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 map these BPEL *primitive activities* to OWL-S *Perform* control constructs (as shown in figure 6).

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

Fig.6. OWL-S Perform control construct 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 a BPEL process model describe the order in which set of the child *primitive* or *structured activities* is performed. For example *structured activity* (*sequence*) describes that the child *primitive* or *structured activities* within a *sequence* activity are performed in a sequence. Similar to BPEL *structured activity* (*sequence*), the OWL-S has *Sequence* control construct, which 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.*

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 uses its control constructs to define control flow between sub *atomic* and *composite processes.*

Let us consider the example below (taken from "Translation And Dictionary" example available with our tool), describing that *structured* activity (*sequence*) has two sub *primitive activities* that can be performed in a sequence with in a BPEL process model.

```
<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.7. Sequence activity having child primitive activities (invoke).

Figure 8 shows the OWL-S control construct *Sequence* as a result of mapping of BPEL *structured activity* (*sequence).* Figure 8 shows that *Sequence* control construct has two *atomic processes* (i.e. "getTranslationProcess" and "getMeaningProcess") that can be performed in a sequence. Where two *Perform* control construct statements are result of mapping of two *invoke* activities with in BPEL *sequence* activity (as discussed in section 3.2.2).

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 *Split-Join* 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.

```
<process:CompositeProcess
      rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestProcess">
 <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>
</process:CompositeProcess>
```

Fig.8. OWL-S sequence control construct statement.

Switch, *structured activity* supports conditional behavior. A conditional branch is defined by the *case* element, which can have optional *otherwise* branch. The BPEL *switch* activity is mapped to OWL-S sequence (*Sequence*) of *If-Then-Else* control constructs. Where each *case* element is mapped to an *If-Then-Else* control construct and *otherwise* part of the *case* statement is mapped to *Else* part of *If-Then-Else* control construct. Figure below (figure 9) shows the 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.9. BPEL switch activity statement.

Figure 10 shows the mapping of the BPEL *switch* activity to the OWL-S sequence of *If-Then-Else* 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.10. Mapping of "switch" activity to OWL-S sequence (Sequence) of If-Then-Else control constructs.

Also, the *while* activity in BPEL is mapped to the *Repeat-While* control construct in OWL-S, to repeatedly perform a specific process.

Conditions: mapping for condition statements is not fully supported in this version. 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. in "Demo.owl" file, in case of our Demo example project). This condition statement can be used to manually create the *SWRL* expressions for conditions in OWL-S.

3.3 Data Flow

According to the BPEL and the OWL-S specifications there are no logically equivalent activities in BPEL and OWL-S for direct mapping of the *Assignment* activity, which can be used to define data flow between process components in the OWL-S *composite process*. Therefore, here, we discuss the criteria we have implemented for defining data flow between *atomic processes* within a *composite process.*

The BPEL *Assignment* activity can be used to assign an output message or message part of a *primitive activity*, as an input message or message part of other *primitive activity*. According to mapping specifications, *invoke* activities before and after *Assignment* activity are mapped to the OWL-S *Perform* control constructs (as discussed in section 3.2.2) and message and message parts in the *from* and *to* part of *Assignment activity* are used to create the data flow between these *atomic processes*. Figure 11 gives a simplified view of criteria we used to create data flow. Figure 11 shows that *primitive activities* (performing operations "Operation1" and "Operation 2" respectively) before and after *Assignment* activity are mapped to *atomic processes* ("Atomic Process 1" and "Atomic Process 2" respectively) and message and message parts of these activities used in *Assignment* activity are mapped to data flow statements in OWL-S by using <*Process:hasDataFrom*> and <*Process:InputBinding*> constructs.

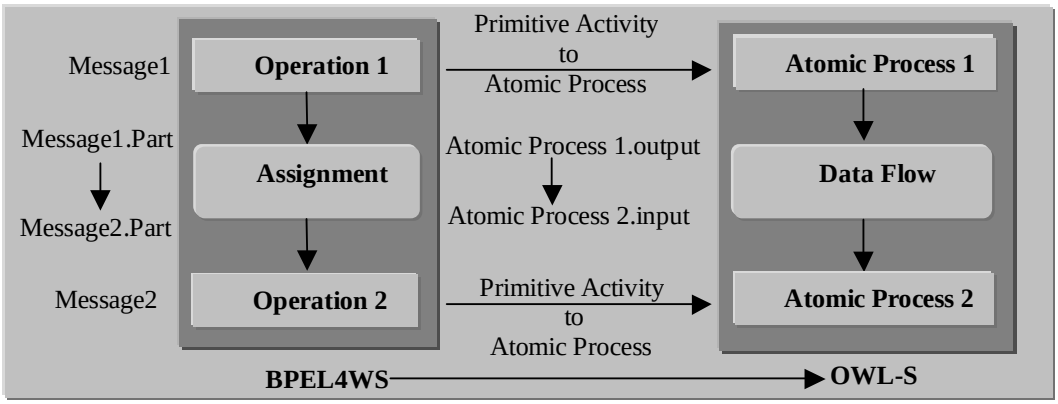

Fig.11. Mapping of Assignment activity to create data flow between process components.

4 Profile and Grounding Ontologies

In above section we have discussed in detail, how a BPEL process model is mapped to OWL-S *Process Model* ontology. This section describes how *Profile* and *Grounding* ontologies are created during the whole mapping process.

4.1 Profile

A BPEL process can have multiple interfaces as *receive*, *reply* or *invoke* activities. These activities can have input/output messages, which are defined in the BPEL's 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 *primitive activity* options, message variable of the first *receive primitive activity* that receives a message from the outer world is selected to define the input of the OWL-S *composite process*. Message parts of this message variable are defined as input parameters of the resulting *Profile* ontology and these input parameters are annotated with dummy ontological concepts to define semantics of resulting OWL-S service. 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*. If a *receive* activity don't has corresponding *reply* activity then first *primitive activity* (i.e. first *invoke* activity) sending some message to the outer world (working as an interface of BPEL process) is taken as an output activity to define the output of the OWL-S *composite process*. Message variable of this activity is used to define the output parameters in the *Profile* ontology of resulting OWL-S service and theses output parameters are also annotated with dummy ontological concepts to provide semantics of mapped BPEL process as OWL-S service. End user of the mapping tool needs to replace these URIs of dummy ontological concepts with their domain ontologies.

A *primitive activity* is declared as an Input/Output (I/O) activity if its port type and operation is supported by the BPEL's corresponding WSDL file. Even though, OWL-S specifications support multiple instances of *Profile* ontology for one instance of *Process Model* ontology but tool create one instance of *Profile* ontology for one instance of *Process Model* ontology. Therefore, in case, if a BPEL process has more than one I/O *primitive activities* (which create the interface of BPEL process) then only one instance of *Profile* ontology is created according to above discussed specifications.

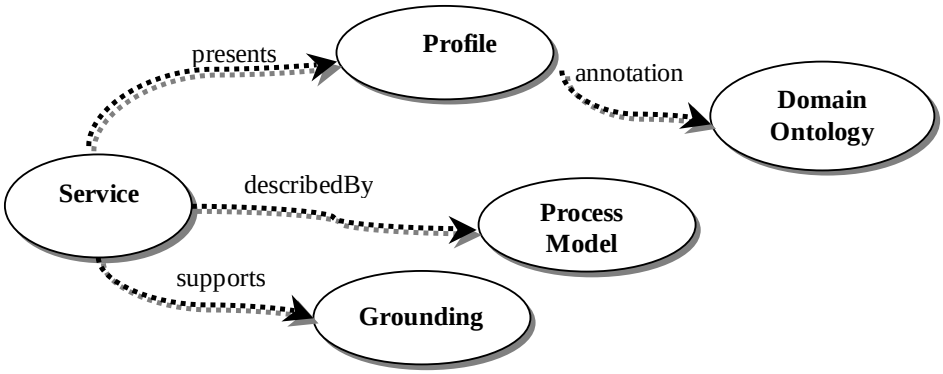

Fig.12.Annotating input/output parameters in Profile ontology with domain ontology.

4.2 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 composite service specifies the address of the grounding of each *atomic process* (as shown in fig. 13). Also, concrete messages are specified explicitly in *Grounding*. Of course, it is not possible to automatically define the XSL Transformation [15] 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">
  ……
 <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.13. Service Grounding for "Translation And Dictionary" example.

In figure 13 “getMeaningAtomicProcessGrounding” and “getTranslationAtomicProcessGrounding” are groundings for “getMeaningProcess” and “getTranslationProcess” defined in “getMeaning.owl” and “getTranslation.owl” created in section 3.2.1.

5 BPEL4WS2OWL-S Mapping Tool

In this section we describe the internal architecture of the tool and its user interface. Figure 14 gives a simplified view of the architecture of the tool and shows that tool consists of three major components (i.e. WSDL Parser, BPEL Parser and OWL-S Mapper which uses the OWL-S API).

Function of the WSDL Parser is to parse each of the Web Services (WSDL files) taking part in BPEL process composition and to transfer information about Web Service operations to the OWL-S Mapper. The OWL-S Mapper creates an OWL-S *atomic process* (with *Profile, Process Model* and *Grounding* ontologies) for each Web Service operation. The mapped OWL-S *atomic processes* are stored in a separate OWL file and saved in project atomic processes directory. Then BPEL Parser is activated which parses the BPEL process model file. BPEL *primitive activities* with in a BPEL process model that are used to perform a Web Service operation, are transferred to OWL-S Mapper with information about Web Service operation performed by that *primitive activity*. The OWL-S Mapper maps these *primitive activities* to OWL-S *Perform* control construct statements that are used to perform the relevant *atomic processes*. Where as, OWL-S Mapper has already created and saved the *atomic process* that an OWL-S *Perform* control construct has to perform. Also, BPEL Parser transfers each BPEL *structured activity* to OWL-S Mapper, which maps the BPEL *structured activity* to relevant OWL-S control construct within the OWL-S *composite process*. Notice that the BPEL Parser takes care for *primitive activities*, which are working as an interface of the BPEL process model. The OWL-S Mapper uses message parts of these activities to create the input/output parameters of *Profile* ontology of resulting OWL-S service (as discussed in section 4.1).

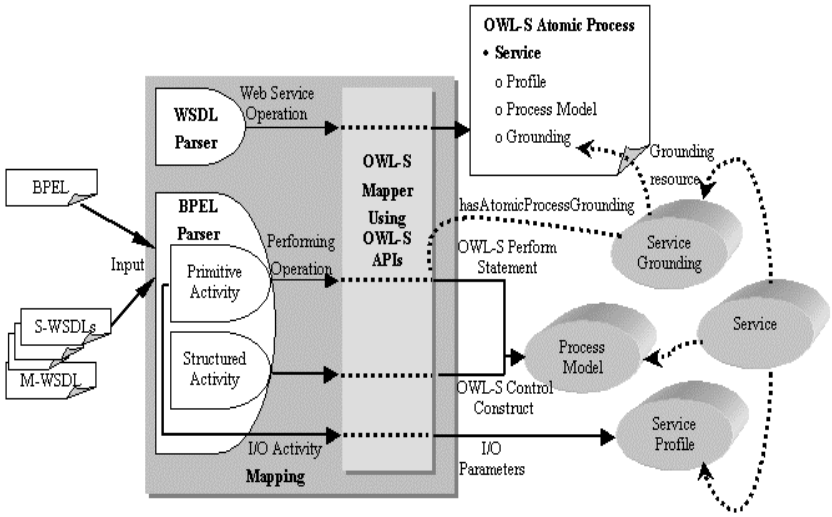

Fig. 14. Architecture of BPEL4WS2OWL-S Mapping Tool.

The “BPEL4WS2OWL-S mapping tool” provides an easy to use interface (fig.15) employing menus and buttons to perform the mapping process. The mapping process includes creating a new project, adding input BPEL and WSDL files, validating the input files, building the project (to create object view of input files) and finally mapping the project. The resulting OWL-S ontology files can be viewed in the project explorer (upper right window) and the contents of these files can be seen in upper left window of the tool. The lower left window acts as an output window to see the output of different mapping actions. The lower right window is an object explorer, which gives an object view of the input files.

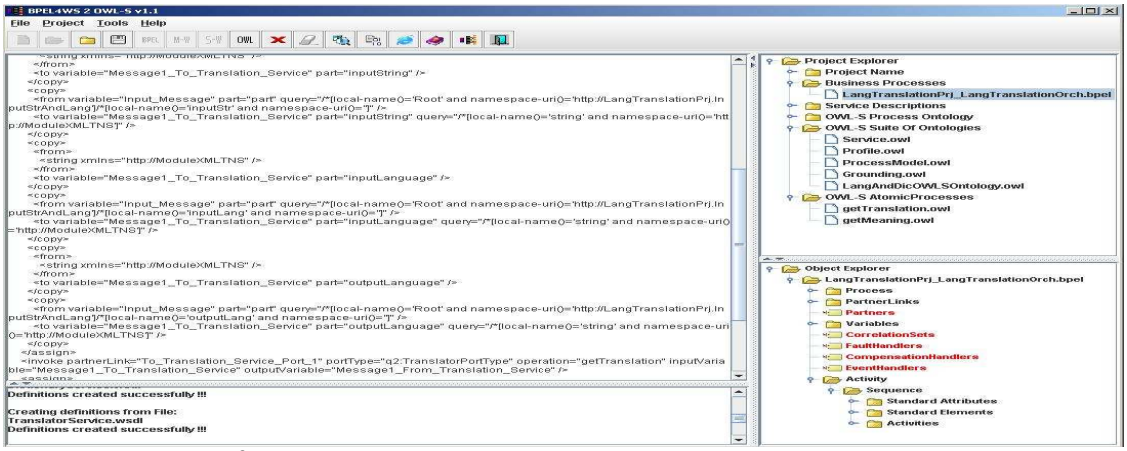

Fig.15. An overview of BPEL4WS2OWL-S mapping tool.

6 Limitations

Here, we describe the limitation of our work in two phrases: one is the OWL-S specifications limitation and other is the mapping implementation. OWL-S is not as mature as BPEL, for example, equivalent of BPEL activities like *Assignment*, *Fault Handler*, *Terminate* etc. are not available in OWL-S for direct mapping from BPEL to OWL-S. Information about pre and post-conditions is not described in BPEL so that it can be used in automatic mapping. Input and output parameters in the resulting *Profile* ontology need to be annotated with real world domain ontologies by the end user. Second phrase is about limitations of mapping implementation. For example, synchronization between process components is not supported. Conditions are partially supported. These are the areas where mapping implementation can be improved with further research and development to create more consistent mapping.

Therefore, mapping support in our tool is limited in above discussed areas and needs to further improvements in our tool to produce more consistent mapping. Also, tool needs to be continuously updated with coming versions of related technologies.

7 Related Work

In this paper, we have presented an approach to map syntax based Web Services composition (BPEL process model) to Semantic Web Services composition (OWL-S composite service). Our work supports the mapping of the BPEL process model to the complete OWL-S suite of ontologies. The resulting OWL-S service can be used for dynamic discovery, invocation and composition by semantic enabled systems.

As discussed before that [8] presents an initial kind of mapping to map BPEL process to OWL-S *Process* ontology (with above discussed limitations) to facilitate p2p based service flow system. WSDL2DAML-S [17] (updated to WSDL2OWL-S) presents an approach to add semantics to Web Services by translating Web Service operation to OWL-S ontology. WSDL2OWL-S partially supports the translation to *Process Model* and *Profile* ontologies because Web Services technology (WSDL) do not provide all information that is needed for complete WSDL to OWL-S translation. There have also been other efforts to add semantics to different technologies. WSDL-S [16] is an effort by LSDIS Labs to add semantics to WSDL. WSDL-S's approach is to enhance WSDL tags to add semantics (by annotating them with domain ontologies) to Web Services rather than to define a separate ontology to describe Web Services semantically. WSDL-S also adds new tags to WSDL specifications to support pre and post conditions. [11] Presents an approach to enhance dynamic discovery and composition of semantically enriched Web Services with in a BPEL process model. [11] Propose, rather than to bind a service in a BPEL process at design time, user should define semantic requirements of a required service with in a BPEL process. Our work is a more consistent effort to add semantics to BPEL process model by mapping it to OWL-S composite service in which each Web Service operation is an OWL-S *atomic process.*

8 Conclusion and Future Work

As discussed above that end user needs to develop the domain ontologies and to change the *Profile* ontology of resulting OWL-S service by annotating input/output parameters with these domain ontologies. Therefore, a tool is needed that can be used to develop domain ontologies and an editor which can be used to edit the resulting OWL-S ontology (*Profile* ontology) with these domain ontologies. Once completed, the resulting OWL-S service can provide richer business process semantics in the form of OWL-S ontology for flexible integration and automation of workflows in peer-2-peer workflow systems and in Data Grid applications. For example the approach discussed in [11] can be used to dynamically discover, invoke and compose theses Semantic Web Services to complete a process. Also resulting OWL-S ontology can be edited to create more complex Semantic Web Services by composing different atomic and composite services together.

"Protégé" with its plug-in "OWL-S Editor" is an ideal framework to develop domain ontologies and to edit resulting OWL-S service with these domain ontologies. "Protégé" is an ontology development tool and "OWL-S Editor" is an editor, which can be used to visually develop and edit OWL-S services. "OWL-S Editor" is available as a plug-in for "Protégé". We are working to produce more consistent mapping from BPEL to OWL-S to overcome the above-discussed limitations of our work. We are also working to improve our tool as a **"BPEL4WS2OWL-S import plug-in"** for "Protégé" and "OWL-S Editor", so that the mapped OWL-S services can be directly imported in "OWL-S Editor" for editing.

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. UDDI Version 3.0.2: UDDI Specifications Technical Committee Draft, Dated 20041019. [online] Available http://www.uddi.org/specification.html
6. A First Overview of BPEL4WS. January 25, 2.005. [online] Available http://jroller.com/page/coreteam /Weblog?catname=%2FWorkflow
7. OWL-S' Relationship to Selected Other Technologies [online] Available http://www.daml.org/services /owl-s/1.1/related.html.
8. J. Shen, Y. Yang, C. Zhu and C. Wan. "From BPEL4WS to OWL-S: Integrating E-Business Process Descriptions", Proc. of 2nd IEEE International Conference on Services Computing (SCC 2005), pp.181-188, Orlando, USA, July 2005.
9. Frank Leymann. Web Services Flow Language (WSFL 1.0) May 2001. [online] Available http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
10. UDDI Version 3.0.2: UDDI Specifications Technical Committee Draft, Dated 20041019. [online] Available http://www.uddi.org/specification.html
11. 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.
12. Jun Shen, Yun Yang, Jun Yan, A P2P based Service Flow System with Advanced Ontology Profiles, accepted by International Journal of Advanced Engineering Informatics.
13. http://upnp.org/
14. Evren Sirin: OWL-S API. [Project Home Page] Available http://www.mindswap.org/2004/owl-s/api/.
15. XSL Transformations (XSLT) : [online] Available http://www.w3.org/TR/xslt.
16. R. Akkiraju, J. Farell, J.A. Miller, M. Nagarajan, A. Sheth and K. Verma : "Web Service Semantics – WSDL-S" [online] Available http://www.w3.org/2005/04/FSWS/Submissions/17/WSDL-S.htm.
17. M. Paolucci, N. Srinivasan, K. Sycara, and T. Nishimura, "Toward a Semantic Choreography of Web Services: From WSDL to DAML-S" In Proceedings of the First International Conference on Web Services (ICWS'03), Las Vegas, Nevada, USA, June 2003, pp 22-26.