Towards Semantic Business Processes: Concepts, Methodology and Implementation

Muhammad Ahtisham Aslam[1], Sören Auer[1,2] and Klaus-Peter Fähnrich[1]

[1]Betriebliche Informationssysteme, University of Leipzig, Germany
{aslam,auer,faehnrich}@informatik.uni-leipzig.de
[2]Computer and Information Science Department, University of Pensylvania, USA
auer@seas.upenn.edu

Abstract

The Business Process Execution Language for Web Services (BPEL4WS (shortly BPEL)) is one of the most popular languages and a de facto standard for modelling business processes as Web services compositions. However, it only allows using hard-coded syntactical interfaces for partners and the process itself, i.e. semantic descriptions of services cannot be used within a process model. The lack of an ontological description of the process elements cause limitations in the ways services are used within a process. A service providing the same functionality as the one referenced in the process model, but via a different syntactical interface, cannot be used instead. As a result, a process model cannot find an alternate service that performs the same functionality but exposes a different interface, and can crash. Also, another drawback of such business processes is that they expose syntactical interfaces and cannot be discovered and composed dynamically by other semantic enabled systems slowing down the process of interaction between business partners. OWL-S on the other hand is a suite of OWL ontologies and can be used to describe the compositions of Web services on the basis of matching semantics as well as to expose semantically enriched interfaces of business processes as OWL-S composite services. Consequently, translating BPEL process descriptions to the OWL-S suite of ontologies can overcome the syntactical limitations of BPEL processes enabling them to 1) edit and model the composition of Web services on the basis of matching semantics 2) provide semantically enriched information of business processes. This semantically enriched information helps for dynamic and automated discovery, invocation and composition of business processes as semantic Web services (SWSs). Describing an approach and its implementation that can be used to enable business processes for semantic based dynamic discovery, invocation and composition by translating BPEL process descriptions to the OWL-S suite of ontologies is the aim of this chapter.

1 SWS Technology Emergence and Current Status

Investigating capabilities and limitations of the semantic Web, semantic Web languages, SWSs and SWSs languages that can be used to overcome syntactical limitations of process modeling languages (e.g. BPEL (Francisco, et al., 2003; Matjaz, et. al. 2004)) is the preliminary step to understand the problem and to navigate through possible solutions. Here, we describe that how different workflow modeling languages (e.g. BPEL) can be used to model business processes as compositions of multiple services and what are limitations of such syntax based Web services compositions. Then we describe the vision of the semantic Web and provide a short overview of semantic Web languages (e.g. RDF (Graham & Jeremy, 2004), RDF-S (Dan, et al., 2004) and OWL (Deborah, & Frank, 2004)). We discuss how OWL ontologies can be used to provide machine understandable meanings of data. We also describe how the SWS community makes use of the semantic Web language (i.e. OWL) to provide machine understandable meanings for Web services. We also shortly summarize technical features of SWS languages (e.g. OWL-S (David et al., 2006), WSDL-S (Rama et al., 2006), WSMO (Sinuhe, 2005)) and compare them with respect to their semantic and workflow modeling capabilities. By analyzing and comparing existing SWS languages we argue that the semantic and process modeling capabilities of OWL-S are much better as compared with other SWS languages and it can be used to address the syntactical limitations of traditional process modeling languages (e.g. BPEL) by translating BPEL process descriptions to the OWL-S suite of ontologies.

1.1 Workflow Modeling

Different workflow languages like Web Services Flow Language (WSFL) (Frank, 2001), MS XLANG (Satish, 2001) and Business Process Execution Language for Web services (BPEL4WS, shortly (BPEL)) (Francisco, et al., 2003; Matjaz, et al. 2004) have been developed to define workflows. WSFL from IBM addresses workflow on two levels: 1) it takes a directed-graph model approach for defining and executing business processes 2) it

defines public interfaces that allows business processes to advertise as Web services (Jun et al. 2006). XLANG is an XML based business process language that can be used to orchestrate Web services. An XLANG service description is a WSDL (David, & Canyang, 2007) service description with an extension element that describes the behaviour of the service as a part of a business process. MS XLANG is the language that is used in MS BizTalk Server (Microsoft's business process modeling tool). However, processes modelled in BizTalk server can easily be exported and imported to BPEL (an industry wide accepted standard for modeling business processes).

1.1.1 BPEL4WS

BPEL4WS is a mature business process modeling language and is industry wide accepted standard for modeling business processes as Web services compositions. A BPEL process consumes Web services operations to perform a specific business task by defining control flow and data flow between these Web services operations and can itself be exported as a Web service. BPEL supports the implementation of any kind of business process in a very natural manner and has gradually become the basis of a standard for Web service description and composition (Jun, et al., 2006). Several characteristics of BPEL make it language of choice for modeling business processes. For example, BPEL is a language that combines workflow capabilities of IBM WSFL and structural constructs of MS XLANG. Most of process modeling tools (e.g. MS BizTalk Server, IBM WebSphere, SAP NetWeaver etc.) provides support for importing and exporting BPEL processes from one framework to other. In presence of all these capabilities the BPEL has many shortcomings resulting in limitations for seamless interoperability of business processes. These limitations can be addressed successfully by getting across semantic gap between process modeling languages and upcoming semantic Web and SWSs languages. Figure 1 gives an overview of evolution and relation between these syntax and semantic based languages.

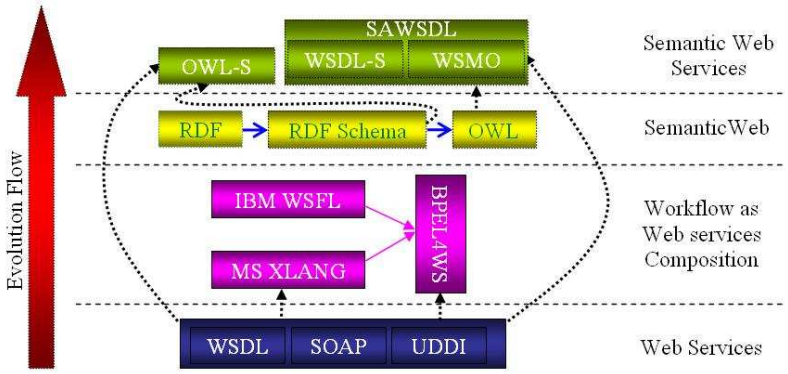

Figure 1. Evolution and relation between Web services, workflow, semantic Web and SWSs languages.

1.2 Semantic Web

Semantic Web is an extension to the current Web (WWW) to present more meaningful data that is easily and efficiently process able and understandable for machines. It aims at providing common formats for exchanging data and languages for describing relations between data objects. For this purpose different languages (e.g. RDF, RDF-S and OWL) have been presented. Resource Description Framework (RDF) was developed to provide a standard way to model, describe, and exchange information about resources. Providing information as RDF triples was not enough for the vision of the semantic Web to become true. Consequently, further development resulted in Resource Description Framework Schema (RDF-S). RDF-S is semantic extension to RDF, as it enhances the information description capabilities of RDF by describing the groups of related resources and relationship between these resources. Lacks of information expression capabilities of RDF-S (e.g. defining properties of properties, necessary and sufficient conditions for class membership, or equivalence or disjoint of class) resulted in more expressing semantic Web language (i.e. Web Ontology Language (OWL)). OWL is intended to be used when the information contained in documents need to be processed by applications, as opposed to the situations where the content only needs to be presented to humans (Deborah, & Frank, 2004, p. 1). Figure 2 (taken from Evren Sirin's talk "using Web ontologies for Web Services Composition") gives a very interesting example of OWL ontology. This sample ontology defines relation of a student with his geographical location, university, course etc.

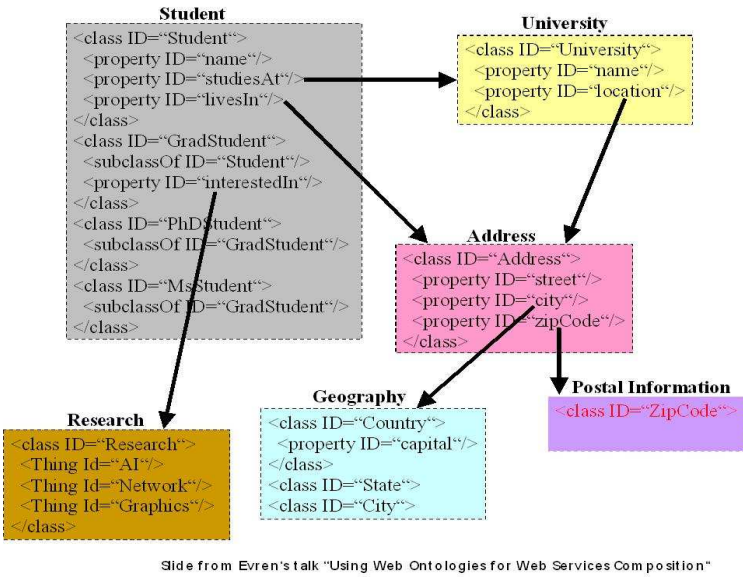

Figure 2. Relational semantics defined with OWL ontology.

1.3 Emerging SWSs Languages

Different efforts are going on to develop SWSs languages (e.g. WSDL-S, WSMO and OWL-S). All of these SWSs languages working groups are using OWL to provide domain specific semantics of a Web service. Here we provide short descriptions of these SWSs languages to compare their process modeling and semantic capabilities and limitations.

1.3.1 WSDL-S

WSDL-S is a SWS development language that is being developed jointly by the University of Georgia and IBM. WSDL-S extends WSDL *operation* and *message* tags by annotating them with domain ontologies to provide semantics. In addition with extending WSDL, WSDL-S also adds new tags (i.e. *LSDISExt:precondition* and *LSDISExt:effect*) to WSDL specifications to describe pre-conditions and effects of a Web service operation. Figure 3 summarizes WSDL-S approach. Also, WSDL-S concepts are being fed to upcoming SWS language (i.e. Semantic Annotation for WSDL (SAWSDL) (Joel, & Holger, 2006)) as a joint effort of WSDL-S and WSMO working groups. Since, WSDL-S concepts are being implemented as major of the SAWSDL approach therefore, we do not discuss it separately.

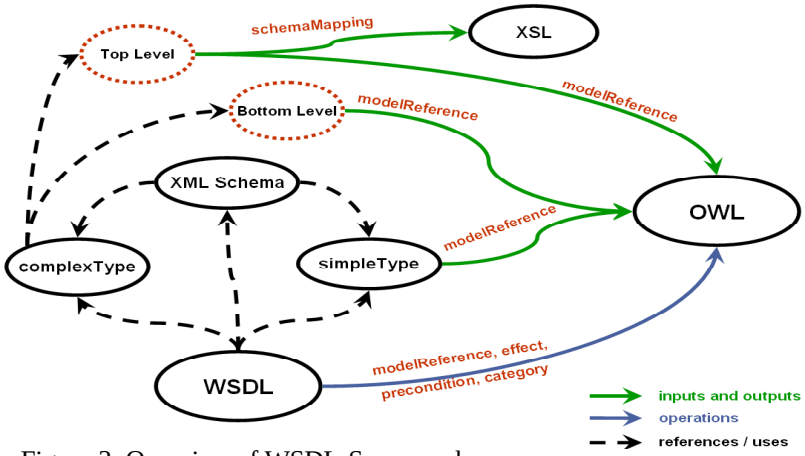

Figure 3. Overview of WSDL-S approach.

1.3.2 WSMO

Web Service Modeling Ontology (WSMO) is part of ongoing research to achieve dynamic, scalable and cost-effective infrastructure for transaction and collaboration of business services. Web Service Modeling Language (WSML) (Joel, et al., 2006) is formal language used to describe WSMO services. The Web Service Execution Environment (WSMX) (Christoph, et al. 2005) is execution environment for dynamic discovery, invocation and composition of WSMO services.

1.3.3 OWL-S

OWL-S is another language being developed to provide Web services semantics to facilitate dynamic and automated discovery, invocation and composition of Web services. OWL-S is suite of OWL ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies). *Profile* ontology provides semantically enriched information about Web service capabilities that helps in semantic based publishing and discovery of Web services. *Process Model* ontology describes how to use a service and can be used for semantic based composition modeling of complex services. *Grounding* ontology describes how to access a service. OWL-S uses OWL ontologies to provide universally unique meaning of a service by annotating its *inputs, outputs* with domain ontologies and by describing its *pre-conditions* and *effects.* Also, *Process Model* ontology has very expressive capabilities to model composition of multiple Web services like workflow languages but based on their semantic descriptions. Two major reasons for choosing OWL-S to semantically describe BPEL process models are: 1) *Profile* ontology can be used to provide semantically enriched meaning of a process as OWL-S SWS 2) *Process Model* ontology of OWL-S suite can be used to edit and model compositions of multiple SWSs (like a workflow language). Table 1 describes a comparison of these SWSs languages.

	OWL-S	WSMO	WSDL-S
Language	OWL	WSML	WSDL with Extensions
Multiple Interfaces	Supported	Supported	Not supported
Service Semantics	Supported	Supported	Not Supported
Operational Semantics	Not Supported	Not Supported	Supported
Composite Processes	Supported	Not Supported	BPEL with Extensions
Simple Process Invocation	Supported WSDL Grounding	Not Supported WSDL Grounding	Not Supported WSDL
Development Tool	Available	Available	Available

Table 1. Comparison of SWSs languages

1.4 Problem Scenario

In order to understand the problems raised due to semantic limitations of BPEL we consider an example scenario of Web services composition (i.e. a BPEL process). The example scenario helps to realize needs for establishing correspondence between syntax based and semantic based compositions of Web services.
To keep the complexity of 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 service and *atomic* processes are available with the tool download). The *Translator and Dictionary* process is modeled in MS BizTalk Server as syntax-based composition of two services (i.e. 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 supported language by using its operation *getTranslation*. 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 by the *Dictionary* service) by using its operation *getMeaning*. Now we define two problem scenarios (tasks) that cannot be performed by anyone of these two services (i.e. neither by the *Translator* Service nor by the *Dictionary* Service). To perform these tasks we need to model a BPEL process as composition of these two Web services. The two scenarios are:

- 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*.
- 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 gives the meaning of only *English* words in *English*.

In both of above scenarios none of a single Web service is able to perform required task. As a solution we model a BPEL process as composition of these services. In first problem scenario we can define a workflow (Figure 4) as composition of the *Translator* service and the *Dictionary* service and it consists of the following steps.

- Process accepts input string (i.e. *German* word) from the user (a user may be a human user or another Web service).
- Transfers this string as an input to the *Translator* service to translate the string from *German* to *English*.
- Output of the *Translator* service (i.e. the *English* translation of the input string) is given as an input to the *Dictionary* service.
- As a last step of the process, the *Dictionary* service returns the meaning of the input string.

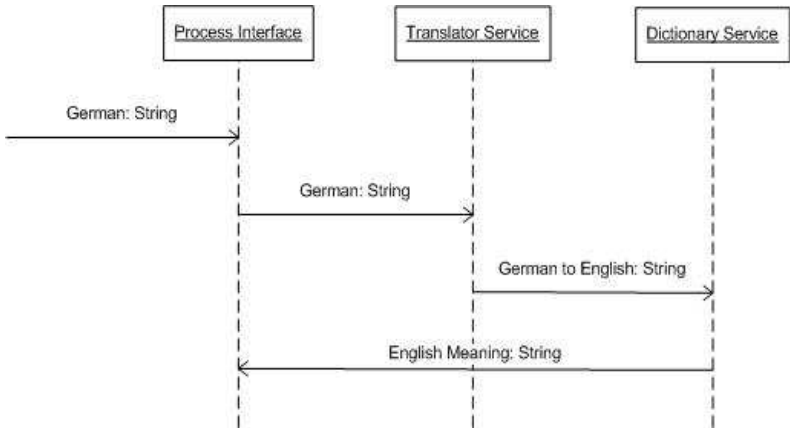

Figure 4. Sequence of services in process according to first scenario.

Similarly task pointed in second scenario (i.e. getting meaning of the *German* word in *German*) can be accomplished by enhancing process model of Web services composition by following steps (as shown in Figure 5):

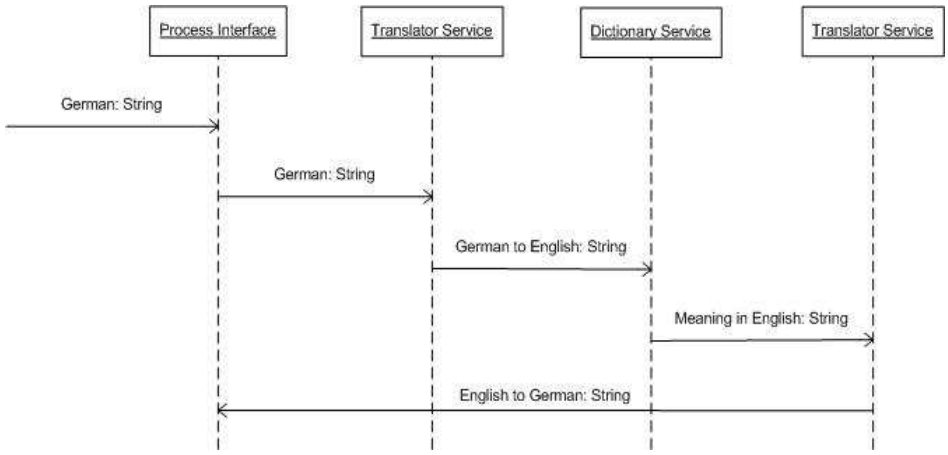

Figure 5. Sequence of services in process according to second scenario.

- Process accepts the input string (i.e. the *German* word) from the user.
- Transfers this string as an input to the *Translator* service to translate the string from *German* to *English*.
- The output of the *Translator* service (i.e. the *English* translation of the input string) is given as an input to the *Dictionary* service.
- 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*.
- As a last step of the process the *Translator* service translates the string (i.e. the meaning of the word) back from *English* to *German*.

If we analyze the process (i.e. composition of Web services) more at semantic level then following issues are identified:

- When the process is exported as a Web service, it has same syntactical limitations as traditional WSDL services resulting in clampdown of process for dynamic discovery, invocation and composition.
- If we want to extend the process (as shown in Figure 4) in a semantic environment to perform the task pointed in second scenario (as shown in Figure 5) then we will realize that:
  - Web services with in composition provide no information for semantic based editing and modeling of process. For example, consider the input message (as shown in Example 1) required by the *Translator* service. This message provides no semantic information about message parts (i.e. *inputString, inputLanguage* and *outputLanguage*).
  - Semantic limitations of Web services with in process restrict to dynamically discover and compose (on the basis of matching semantics) a semantic Web service (e.g. semantic based *Translator* service).

Example 1: A sample WSDL syntax based message.

```
<wsdl:message name="TranslatorRequest">
 <wsdl:part name="inputString" type="s:string" />
 <wsdl:part name="inputLanguage" type="s:string" />
 <wsdl:part name="outputLanguage" type="s:string" />
</wsdl:message>
```

Bridging the semantic gap between syntax based and semantic based composition of Web services can help to address above discussed problems. Example 2 shows annotation of input message part (i.e. *inputLanguage*) with ontology concept (i.e. *SupportedLanguage*) defined in appropriate domain ontology. Providing such semantic information can help to:

- Provide machine understandable meaning of the process as an OWL-S composite service that can help in dynamic discovery, invocation and composition of BPEL process as an OWL-S semantic Web service.
- Shift the process from syntax-based to semantic based composition providing semantically enriched information about each service involved with in composition.
- Edit and model the composition on the basis of matching semantics rather than relying just on syntactical information.
- Defining abstract process (i.e. semantics for a required service) with in composition to dynamically discover and compose a service on the basis of matching semantics defined in abstract process (according to approach discussed in (Evren, et al., 2005)).
- Using an AI planning for automated composition by mapping OWL-S *composite* and *atomic* processes to tasks and operators of the planning language (e.g. HTN planning).

Example 2: Semantically enriched message part.

```
<process:Input rdf:ID="inputLanguage">
 <process:parameterType rdf:datatype="&xsd;#anyURI">
  &this;#SupportedLanguage</process:parameterType>
 <rdfs:label>Input Language</rdfs:label>
</process:Input>
```

In above discussed simple but extensive example we have just considered inputs and outputs of different services for the purpose of composition. In actual scenarios we can use other information related to a Web service (e.g. service provider, response time, geographical location, defining data flow and control flow between services etc.) for more accurate and efficient composition of Web services. *One thing to note* at this point is that we have provided two example scenarios for modeling processes as Web services compositions. For first scenario we modeled a BPEL process in MS BizTalk Server as syntax based composition of two services (i.e. the *Translator* service and the *Dictionary* service). Then we highlighted limitations of such syntax based process modeling. In Sections 2 and 3 we provide detail analysis of BPEL process models and OWL-S SWSs and then on the basis of this analysis we define specifications to translate BPEL process descriptions to OWL-S suite of ontologies. In remaining chapter we use this BPEL process (please note that the '.bpel' of the process and the '.owl' files of mapped OWL-S service and *atomic* processes are available with the tool download) to provide some code samples of mapping specifications. In evaluation section (i.e. Section 5), 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. 1) semantic based composition editing and modeling of services 2) semantically enriched interface of the BPEL process as OWL-S SWS). In our evaluation section we enhance the *Process Model* ontology of mapped OWL-S service in semantic environment (e.g. Protégé (John, et al., 2003) (OWL-S Editor) (Daniel, et al., 2005) or even with simple editor like Notepad to develop SWS for scenario 2.

2 Mapping Constraints

Mapping constraints create the base of mapping specifications that can be used to translate BPEL process descriptions to OWL-S suite of ontologies by providing analysis of BPEL process model, OWL-S SWS and their components. Here, we do not mean to provide complete description of these languages as their specifications cover them very well but analytical description of BPEL process models and OWL-S suite of ontologies helps to categorize and to specify that which part of a process should be translated to which construct of OWL-S.

2.1 Analysis of BPEL Process Model

A BPEL process model is set of *primitive* and *structured* activities. Here, we describe functional behavior of BPEL processes and its activities on the basis of which we have defined specifications for translating BPEL process descriptions to OWL-S SWS.

2.1.1 Processes

BPEL allows describing business processes in two ways:

Executable Processes are used to model interaction between participants (i.e. Web services) of a business process. The logic and state of the process determine the nature and sequence of Web services interactions conducted at each business partner, and thus the interaction protocol (Francisco, et al., 2003, p. 9).

Abstract Processes are not typically executable. They are meant to couple Web service interface definition with behavioral specifications that can be used to both constrain the implementation of business roles and define in precise terms the behavior that each party in a business protocol can expect from others (Matjaz, et al., 2004, p. 51).

2.1.2 Primitive Activities

A BPEL process is a set of activities (i.e. *primitive* and *structured* activities). *Primitive* activities are used to perform basic tasks of a process. Some important BPEL *primitive* activities and their behavioral characteristics are as under:

Invoke (<*invoke*>) activity is used to invoke a Web service by sending it some input message and probably by receiving some output message (Example 3 shows a sample *invoke* activity).

Example 3: *invoke* activity that performs a Web service operation (i.e. *getTranslation* operation).

```
<invoke partnerLink="To_Translation_Service_Port_1"
        portType="q2:TranslatorPortType" operation="getTranslation"
        inputVariable="Message1_To_Translation_Service"
        outputVariable="Message1_From_Translation_Service" />
```

In a BPEL process *invoke* activity can have dual behavior i.e. 1) it can be used to perform a Web service operation 2) it can be used to create the interface of an asynchronous BPEL process. Due to its different behavior, mapping of *invoke* activity to OWL-S also varies (as discussed in Sections 3.1.2 and 3.2).

Receive (<*receive*>) activity receives a message from a Web service probably to start a process. Like an *invoke* activity, a *receive* activity also has dual behavior i.e. 1) it can act as an interface of a BPEL process 2) it can be used to receive a message from a Web service in response to an asynchronous Web service operation.

Reply (<*reply*>) activity is used to reply a message in response to a *receive* activity.

Assignment (<*assign*>) activity is used to assign values to message variables. In a BPEL process the Assignment activity can be used to initialize input message of a Web service operation.

Primitive activities are used to perform small tasks within a complex process. Different activities can be combined and their order of execution can be defined by using some *structured* activities.

2.1.3 Structured Activities

BPEL *structured* activities are used to define control flow between sub *primitive* and *structured* activities within a process. Some major structured activities with their functional behavior are described below.

Sequence (<*sequence*>) activity is used to define a set of activities that are performed in a sequence. A *sequence* completes when its last child activity has been performed.

Flow (<*flow*>) activity is used to invoke child activities concurrently. A *flow* activity completes when all activities within *flow* activity have completed.

Switch-Case (<*switch*>) activity is used to perform child activities under some conditional aspects. A *switch* activity can have one or more conditional branches defined by *case* elements. A *case* may have optional *otherwise* branch that is performed when condition statement becomes false.

While (<*while*>) is used to repeatedly perform a child activity. The child activity under the *while* activity is performed as long as the *while* condition holds true.

2.1.4 Some Additional Activities

Wait (<*wait*>) activity is used to wait for some time.

Throw (<throw>) activity is used for throwing exceptions and indicating faults.

Terminate (<*terminate*>) activity is used to terminate a process.

In this section we provided an analytical description of BPEL processes and functional constraints of BPEL activities. With such analytical description of functional constraints of BPEL processes and activities it becomes easier to specify which BPEL activities have matching behavior to which OWL-S control construct (CC). Table 2 summarizes BPEL process components with their short functional description.

Activities	Description
Primitive Activities	
Invoke	Performs WS operation or create process interface
Receive	Receives process input message or response of synchronous WS operation
Reply	Replies in response of some Receive activity
Assignment	Assigns message values
Structured Activities	
Sequence	Performs sub-activities in sequence
Flow	Synchronizes sub-activities
Case-switch	Shows conditional behavior
While	Repeatedly performs a task
Some Other Activities	
Wait	Waits for some time
Throw	Throws exceptions and errors
Terminate	Terminates a process

Note: WS stands for Web service.

Table 2. BPEL process model activities and their description.

2.2 Analysis of OWL-S Ontologies

OWL-S is being developed to describe SWSs. Here, we analyze functional constraints of OWL-S suite and its CCs that can help to specify that which activities of BPEL process can be mapped to which OWL-S CCs on the basis of their matching behavior.

2.2.1 OWL-S: Technical Overview

OWL-S is suite of OWL ontologies (i.e. *Profile*, *Process Model* and *Grounding* ontologies). *Profile* ontology is used to present semantically enriched interface of a process as SWS. Like a workflow language, the *Process Model* ontology can be used to model composition of multiple *atomic* and *composite* processes (services). Figure 6 provides an overview of the OWL-S *Process Model* ontology and relation of *Process* class with child classes. *Grounding* ontology describes about how to access a service by specifying message formats, protocols and transport. *Service* ontology actually acts as an organizer for the *Profile*, *Process Model* and *Grounding* ontologies. Each OWL-S service has one instance of the *Service* class.

2.2.2 Processes

OWL-S has three kinds of processes:

Atomic Processes are processes that can be executed in a single step and they have no sub process. *Atomic* processes are somehow like Web services operations that can be performed in a single step. An *atomic* process is described by using the class *AtomicProcess* that is sub class of the *Process* class (as shown in Figure 6).

Simple Processes may be used either to provide a view of (a specialized way of using) some *atomic* process, or a simplified representation of some *composite* process (for purposes of planning and reasoning) (David, et al., 2006, p. 1).

Composite Processes are processes that can have sub *atomic* and *composite* processes. Like a workflow modeling language we can use *composite* processes to model the composition of multiple atomic and composite processes. A *composite* process allows defining the control flow between sub *atomic* and *composite* processes by using different CCs (e.g. *sequence, split, split-join* etc.).

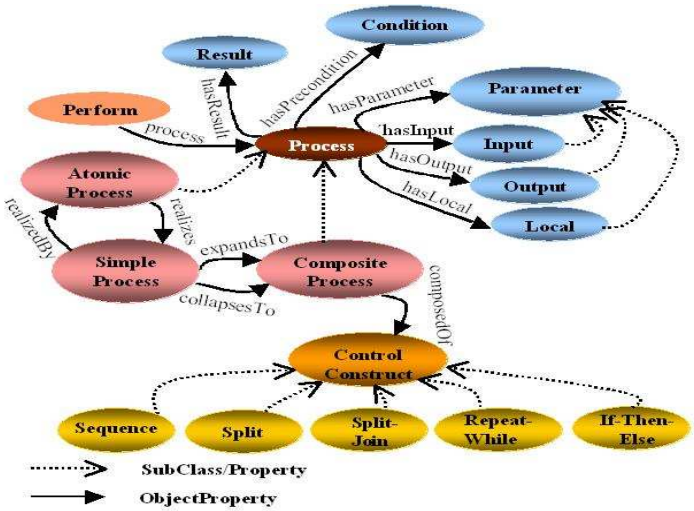

Figure 6. OWL-S *Process Model* ontology.

2.2.3 Control Constructs

OWL-S defines a number of CCs that can be used to define control flow between sub processes within *Process Model* ontology. Discussion about capabilities of these CCs is necessary because they are used to define control flow of BPEL process in the mapped OWL-S service. OWL-S defines many CCs that can be used to define control flow between process components. Some of these CCs are as under:

Sequence, components of a *Sequence* CC are performed in a sequence. *Sequence* class is sub class of the class *ControlConstruct* (as shown in sample code below) that holds other CCs as sub classes.

```
<owl:Class rdf:ID="Sequence">
 <rdfs:subClassOf rdf:resource="#ControlConstruct"/>
  ..........
 </rdfs:subClassOf>
</owl:Class>
```

Split CC is used to perform its process components in parallel. Also, a *Split* CC completes as soon as all of its process components are scheduled for execution.

Split-Join CC is used for concurrent execution of process components with partial synchronization. A *Split-Join* CC completes as soon as all of its process components have been performed.

If-Then-Else CC can be used to implement conditional behavior within a *composite* process. It has three properties (i.e. *ifCondition, then, else*). Execution of *then* and *else* depends on either *ifCondition* is true or false (i.e. if *ifCondition* is true, perform *then* part and if *ifCondition* is false then perform *else*).

Repeat-While CC is used to repeatedly perform its process component (i.e. as long as *Repeat-While* condition holds true). Condition is important part of OWL-S CCs (e.g. *If-Then-Else, Repeat-While, Repeat-Until* CCs).

2.2.4 Some Other OWL-S Mapping Constraints

Some other constraints that need to be addressed while mapping BPEL process descriptions to OWL-S are follows:

Performing Individual Processes: As, we discussed before that a *composite* process is composition of sub *atomic* and *composite* processes, these processes can be performed by using *Perform* CC. The invocation of a process is indicated by an instance of the *Perform* class. The *process* property of class *Perform* indicates the process to be performed.

Condition Expressions: We use SWRL expressions (Peter, 2005) being the most recommended standard to define conditions for OWL-S CCs. OWL-S API (Evren, 2006) (developed by MIDSWAP Lab) also supports the execution of conditions defined by using the SWRL.

Data Flow and Parameter Binding: OWL-S defines a class *Binding* to define the data flow between process components. For sure, OWL-S specifications allow defining hard code values (e.g. 5, "hello" etc.) as inputs of processes.

Parameters and Results: In OWL-S specifications, parameters are what we call variables in general programming languages. Parameters can be expressed by using *Parameter* class.

Table 3 summarizes important components of OWL-S ontology and their analytical description. On the basis of capabilities and limitations of components of BPEL and OWL-S we define mapping specifications for shifting BPEL process model to OWL-S ontology.

OWL-S	Description
Profile	
Input/Output	Provides functional semantics of service as inputs and outputs
Pre-condition/effect	Describes functional semantics as conditions before and after service execution
Result	Conditional output of service
Service category, provider, location	Non-functional semantics
Process Model	
Atomic process	Executes in single step
Simple process	Gives multi view of same process
Composite process	Executes in multiple steps
Sequence	Performs process components in sequence
Split	Concurrently executes process components
Split-Join	Synchronizes process components
If-Then-Else	Shows conditional behavior
Repeat-While	Repeatedly perform sub component
Grounding	
WsdlGrounding	Describes process grounding
hasAtomicProcessGrounding	Provides reference of atomic process grounding
xsltTransformationString	Transform XML document to other

Table 3. Analytical description of OWL-S ontology constructs.

3 Translating BPEL Process Descriptions to OWL-S

In the previous section (i.e. Section 2), we have discussed in detail the process modeling and semantic capabilities of BPEL and OWL-S and components of these languages. Since, OWL-S is suite of three ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies) therefore, we describe the translation (mapping) of BPEL process descriptions to OWL-S at three levels (i.e. mapping of BPEL process model to OWL-S *Profile, Process Model* and *Grounding* ontologies). Table 4 describes specifications for translating the BPEL process descriptions to the OWL-S suite of ontologies. The specifications are described from an abstract level to components and activities level translation of BPEL processes to OWL-S services. Areas, where direct mapping is not possible or requires some additional manual interaction, are discussed as well.

Algorithm 1 provides a very abstract level description of the recursive algorithm used for extracting OWL-S suite from BPEL process model. It traverses *bpel file* objects tree as long as activities in *bpel file* come to end. An important thing to note is that when an activity is not an I/O *primitive* activity then it is mapped to *perform*

CC (as described in Lines 13 and 33 of Algorithm 1) to perform relevant *atomic* process. In next section we describe the extraction of *Process Model* ontology from BPEL process model.

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 is available in OWL-S for direct mapping.

Table 4. Summary of BPEL4WS to OWL-S mapping specifications.

3.1 Translation to the OWL-S Process Model Ontology

In this section we describe how a BPEL process model is translated 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, SWRL expression and parameters respectively. We also provide some code example for translating BPEL activities to OWL-S CCs. The whole process of translating BPEL process descriptions to OWL-S depends on functional characteristics of BPEL and OWL-S components as described in Section 2. During discussion of mapping specifications we consider the translation of BPEL process (i.e. *Translator and Dictionary* process), which is mapped to OWL-S service. Now we describe step by step translation of BPEL process components to OWL-S CCs.

3.1.1 Process Level Translation

BPEL process model is composition of multiple Web services with defined control and data flow to perform a joint task. A BPEL process model is mapped to OWL-S *composite* process that is a semantic based composition of multiple *atomic* and *composite* processes. 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. An *atomic* process within a *composite* process is result of mapping of a Web service operation that is performed by a *primitive* activity.

We discussed before that a BPEL process is composition of Web services operations that can be performed in a single step. Each Web service operation within a BPEL process is mapped to OWL-S *atomic* process. The mapped *atomic* process consists of complete OWL-S suite of ontologies (i.e. *Profile*, *Process Model* and *Grounding* ontologies). Since, actual tasks within 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 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. 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 (OWL-S Editor)). Since, our sample *Translator and Dictionary* process uses two Web services operations (i.e. *getTranslation* and *getMeaning*, as discussed in Section 1.4) therefore,

these Web services operations are translated to OWL-S *atomic* processes (i.e. *getTranslationProcess* and *getMeaningProcess*) and stored in *getTranslation.owl* and *getMeaning.owl* files (as shown in Figure 7).

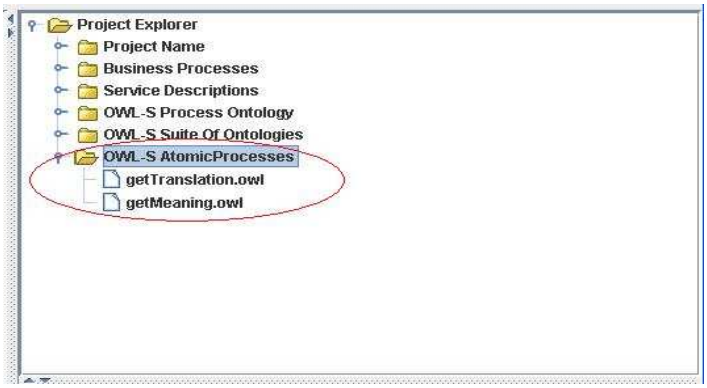

Figure 7. Web services operations translated to OWL-S *atomic* processes and stored in *owl files*.

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 (Algorithm 2)
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 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
33                                 Map it to perform CC to 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             Map structured activity to OWL-S CC (Line 3)
44         end
45     end
46 end
```

Algorithm 1. Abstract level definition of the mapping algorithm.

3.1.2 Activities and Control Flow Translation

In above section we have discussed that a Web service operation performed by a *primitive* activity is mapped to 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 (i.e. *invoke* activity as shown in Example 3) that is used to perform Web service operation (i.e. *getTranslation* operation). 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 above step (i.e. in Section 3.1.1) and stored in *getTranslation.owl* file.

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

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 4 summarizes mapping of BPEL *structured* activities to OWL-S control constructs on the basis of their matching behavior and Algorithm 2 shows it from implementation perspective. We have discussed in detail about behavioral characteristics of BPEL *structured* activities and OWL-S CCs in Sections 2.1.3 and 2.2.3. As sample of mapping these activities we describe translation of two *structured* activities (i.e. *flow* and *switch*) to relevant OWL-S CCs (i.e. *Split-Join* and *sequence* of (*If-Then-Else* CCs)), because mapping of these activities is a little bit tricky.
Synchronization between sub activities and process components is important for defining workflows especially in complex business process integration scenarios. BPEL uses *flow* activity to define synchronization between sub activities. "*A Flow activity completes when all of its sub activities are completed*". OWL-S CCs (e.g. *Split* and *Split-Join*) are used to define synchronization between process components. "*Split-Join completes when all of its process component have completed*". Where as capabilities of *Split* are expressed as: "*Split completes as soon as all of its process components have been scheduled for execution*". Even though both *Split* and *Split-Join* CCs are used for concurrent execution of process components but we map *Flow* activity to *Split-Join* CC on the basis of their matching functional characteristics.
A *switch structured* is used to describe 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 (discussed in next section (i.e. Section 3.1.3)) and *otherwise* part of *case* element is mapped to *else* part of *If-Then-Else* CC. We can summarize mapping of *switch* activity with a list of *case* elements as a sequence (*Sequence*) of *If-Then-Else* CCs mapped with optional *else* part.

```
   Input: structured activity
   Output: OWL-S CC
 1 begin
 2     if activity equal to sequence then
 3         Map to Sequence CC
 4     else
 5         if activity equal to flow then
 6             Map to Split-Join CC
 7         else
 8             if activity equal to while then
 9                 Map to Repeat-While CC
10             else
11                 if activity equal to switch then
12                     Map switch activity to Sequence of If-Then-Else CCs
13                 end
14             end
15         end
16     end
17 end
```

Algorithm 2. Algorithm to map BPEL *structured* activities to OWL-S CCs.

3.1.3 Translating Condition Statements

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, which depend on conditions to OWL-S CCs, is not useful. We have implemented an efficient algorithm that translates *condition* statements of BPEL activities to SWRL expressions, which are supported by OWL-S specifications. The mapped SWRL expressions can be parsed and executed by execution engines (e.g. OWL-S API). Mapping *condition* statements to SWRL expressions supports all conditional operators (e.g. =, !=, <, >, <=, >= etc.).
Before having a look on condition mapping algorithm we should keep in mind that complexity of *condition* statement could vary with complexity of message variables being used in *condition* statement. The reason is that

extracting message variables and message parts of an *atomic* process that are being used in *condition* statement is a complex task (specially when message variables of complex message types are involved). However, our algorithm handles the translation of *condition* statements to SWRL expressions carefully and efficiently by parsing and tracking the list of *atomic* processes and their messages.

```
   Input: condition statement
   Output: SWRL expression
 1 begin
 2     Parse condition statement
 3     Extract left hand operands of condition statement ( i.e. message1_Name, variable1_Name and
       part1_Name )
 4     if variable1_Name equal to null and part1_Name equal to null then
 5         while list of Local Variables not ended do
 6             if local_Variable_Name equal to message_Name then
 7                 Save reference of local variable as local_Variable1_Name
 8             end
 9         end
10     end
11     Find condition operator
12     if right hand operand is message variable of an atomic process then
13         Find index of "and" operator or "or" operator
14         if "and" operator exists or "or" operator exists then
15             Display message "Multiple conditions are not supported"
16             Extract right hand operand of condition statement
17         end
18         Extract right hand operand of condition statement ( i.e. message2_Name, variable2_Name
           and part2_Name )
19         if variable2_Name equal to null and part2_Name equal to null then
20             while list of Local Variables (local_Variable_Name) not ended do
21                 if local_Variable_Name equal to message_Name then
22                     Save reference of local variable as (local_Variable2_Name)
23                 end
24             end
25         end
26     end
27     Extract right hand operand (i.e. expression)
28     if (local_Variable1_Name equal to null and local_Variable2_Name equal to null ) or (
       local_Variable1_Name equal to null and expression equal to null ) then
29         while condition operands not end do
30             while list of atomic processes not ends do
31                 if operand equal to atomic process input then
32                     Save reference of atomic process input
33                 end
34                 if operand not equal to atomic process input then
35                     Find operand in output list of atomic processes and save its reference
36                 end
37             end
38         end
39     end
40     Generate SWRL expression;
41 end
```

Algorithm 3. Algorithm to parse *condition* statement and to generate SWRL expression.

3.1.4 Data Flow Translation

We can discuss the mapping of data flow at two levels; one is defining the input and output of a *composite* process, second level of defining data flow is passing messages between process components within *composite* process.

To understand data flow definition at first level, consider a BPEL process in which *receive* activity receives a message from 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 *composite* process. In such cases output of sub process is also defined as output of the process *TheParentPerform*.

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 (data) between sub processes within a *composite* process is addressed by using the *Binding* class.

3.1.5 Variables and Local Parameters Translation

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. These local variables can be used to manipulate and share data between sub *atomic* and *composite* processes. In Section 3.1.3 we have discussed that how these local variables are used in *condition* expressions to store and compare values with inputs and outputs of sub *atomic* and *composite* processes. Local variables can be connected with processes by using the property *hasLocal* of the *process* class.
In this section we have discussed the translation of BPEL process model to OWL-S *Process Model* ontology. We also described the logic of translation of BPEL activities to OWL-S CCs on the basis of mapping constraints (discussed in Section 2). Translation of some of BPEL activities to OWL-S CCs have been described with their syntactical information 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 Translation 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. Semantics of these input/output parameters, pre-conditions and effects are provided by annotating them with domain ontologies defined in a separate OWL files. Since, BPEL process model provide no semantic information about a process therefore, *Profile* ontology parameters of mapped OWL-S service are automatically annotated by the mapping tool with dummy ontological concepts (URIs). Since, semantic information about a service capabilities can vary from user to user therefore, it is not possible to judge a user requirements automatically, generate domain ontologies according to that requirements and annotate *Profile* ontology parameters with these ontological concepts. Maximum process of generating *Profile* ontology from BPEL process is performed automatically by the mapping tool but end user can provide semantic of mapped service by annotating input/output parameters of *Profile* ontology with their required domain concepts. In short user can finish up with *Profile* ontology by performing following tasks:

- Developing domain ontologies by using some semantic Web development tool (e.g. Protégé)
- Annotating *Profile* ontology parameters with these domain ontology concepts

How to develop domain ontologies (Matthew, et al., 2004), edit (annotate) and develop SWSs with these domain ontologies ? is not the aim of this chapter. However, we explain these topics to some extent so that the end user can get more clear idea and understanding that how the *Profile* ontology of mapped OWL-S service can be extended to enable it for semantic based publishing and discovering. First of all, we describe the criteria that we used to extract *Profile* ontology from a BPEL process model and automatic annotation of *Profile* ontology parameters with domain ontologies. Then we give a short description about how to develop domain ontologies and to use them to annotate *Profile* ontology parameters of mapped OWL-S service.

3.2.1 Extracting the Profile Ontology

In Section 2.1.2 we have already discussed that *primitive* activities can have dual behavior i.e. 1) to perform a Web service operation 2) to interact with the outer world (i.e. to create interface of 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 3.1.1 and 3.1.2. Here we are concerned with *primitive* activities that can be used to create interface of BPEL process model. A BPEL process can have one or more *primitive* activities (i.e. *receive*, *invoke* and *reply* activities) that are used to interact with outer world. Such activities are declared as input/output (I/O) activities during mapping process. 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 I/O activity and message parts of the message received by this activity are used to create input parameters of resulting *Profile* ontology. Again, consider a *primitive* activity (<*receive*>) and its message that has three parts (i.e. *input_Lang*, *output_Lang* and *input_Str*). These message parts are used to create input parameters of resulting *Profile* ontology (as shown in Example 6).

Example 6: 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#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>
 ……. other input/output parameters
 <rdfs:label>BPEL2OWLS Profile</rdfs:label>
 <service:presentedBy rdf:resource="&bpel4ws2owls#TestService"/>
</profile:Profile>
```

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 does not has corresponding *reply* activity (as you can see in some example BPEL processes available with the tool download) and BPEL process uses *invoke* activity to send output message to the outer world. In this case message of the *invoke* activity is parsed in corresponding WSDL file and its message parts are used to create output parameters of *Profile* ontology of mapped OWL-S service.

Till now we have explained that how *primitive* activities are used to create interface of BPEL process and how we use message parts of these I/O activities to create input/output parameters of mapped *Profile* ontology. One more thing that needs to be clarified is that among dual role of BPEL *primitive* activities how a *primitive* activity is declared as an I/O activity so that its message parts can be used to create input/output parameters of *Profile* ontology. The criteria that we used for this purpose is that if a *receive* activity is being used as an initial activity to start a BPEL process (i.e. its *createInstance* attribute value is *yes*) and its *portType* and *operation* is supported by BPEL's corresponding WSDL file.

Another important 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 an interface of BPEL process, only two *primitive* activities are declared as I/O activities and their message parts are used to create input/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 translation of BPEL process description to OWL-S extracts one *Profile* ontology for one *Process Model*.

3.2.2 Developing and Annotating with Domain Ontologies

In above section we have described in detail that how a *Profile* ontology is extracted from a BPEL process model. If we have a deeper look at sample *Profile* ontology (i.e. Example 6) provided in previous section, we see that input/output parameters of *Profile* ontology are mapped to dummy URIs. These dummy URIs need to be replaced with user defined domain ontological concepts (Figure 8 provides a conceptual view of annotating *Profile* ontology parameters with domain ontological concepts). Such annotation provides semantically enriched information about capabilities of mapped OWL-S service.

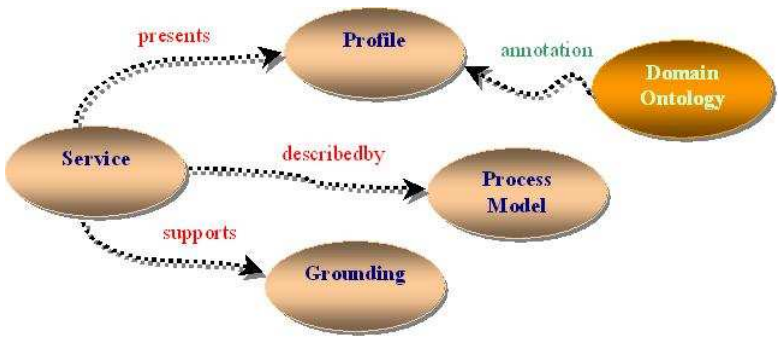

Figure 8. Annotating *Profile* ontology with domain ontology concepts.

Since, OWL-S specifications support to define multiple *Profile* ontologies for one *Process Model* ontology therefore, end user can also define multiple *Profile* ontologies for one *Process Model* ontology to provide different meaning of same service. Protégé with its OWL plugin (Holger, et al., 2004) is an ideal framework for developing domain ontologies. Example 7 gives a simple example of the *Language* ontology that we can use to annotate input/output parameters of our mapped *Profile* ontology to provide semantically enriched information of mapped service.

Example 7: Sample *Language* ontology.

```
<owl:Class rdf:ID="SupportedLanguage">
 <rdfs:comment>Languages supported by the BabelFish translator is an
       enumerated set of the following languages</rdfs:comment>
 <owl:oneOf rdf:parseType="Collection">
  <factbook:Language rdf:about="&factbook;#English"/>
  <factbook:Language rdf:about="&factbook;#Dutch"/>
  <factbook:Language rdf:about="&factbook;#French"/>
  <factbook:Language rdf:about="&factbook;#German"/>
  ..........
  .......... (list of supported languages)
 </owl:oneOf
</owl:Class>
```

Suppose that above language ontology is defined at following address *http:www.uni-leipzig.de/Languages.owl*} (shortly &languages). Then the mapped *Profile* ontology (as show in Example 7) after annotating its parameters with domain ontology looks like:

```
<profile:Profile rdf:about="&bpel4ws2owls#TestProfile">
 <profile:textDescription>This Profile is created by BPEL2OWLS Tool
 </profile:textDescription>
  <profile:hasInput>
   <process:Input rdf:about="&bpel4ws2owls#inputLang">
    <process:parameterType rdf:datatype="http://www.w3.org/2001/
                          XMLSchema#anyURI">&languages#SupportedLanguage
    </process:parameterType>
   </process:Input>
  </profile:hasInput>
    ..........
    .......... (other input/output parameters)

 <rdfs:label>BPEL2OWLS Profile</rdfs:label>
 <service:presentedBy rdf:resource="&bpel4ws2owls#TestService"/>
</profile:Profile>
```

3.3 Translation to the OWL-S Grounding Ontology

Grounding ontology of the OWL-S service describes that how to access a service. Access details 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/outputs of *atomic* processes in some transmittable format is provided in *Grounding* ontology. For this purpose original WSDL services are referred in *Grounding* to access real implementation of services. When a Web service operation within a BPEL process is mapped to OWL-S *atomic* process (during the mapping process) then input/output messages of Web service operation are defined as set of inputs/outputs in the *Grounding* ontology of that *atomic* process. That's why in Section 3.2 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 inputs and outputs in *Profile* ontology. These inputs and outputs 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[1] or can be defined in separate OWL file and can be referred in the type parameter[2].

An OWL-S service *Grounding* is an instance of the *Grounding* class that has sub class *WsdlGrounding*. Each *WsdlGrounding* class contains a list of *WsdlAtomicProcessGrounding* instances that refers to *Grounding* of *atomic* process. *WsdlAtomicProcessGrounding* has properties (e.g. *wsdlInputMessage, wsdlInput, wsdlOutputMessage, wsdlOutput* etc.). *wsdlInputMessage* and *wsdlOutputMessage* objects contain mapping

[1] http://www.mindswap.org/2004/owl-s/services.html BabelFish Translator service provide such example.
[2] http://www.mindswap.org/2004/owl-s/services.html Find Cheaper Book Price service provide such example.

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 XSLT Transformation property gives an XSLT script that generates message part 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 which 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>
```

3.4 Implementation of Mapping Tool

We have developed a tool (i.e. BPEL4WS 2 OWL-S Mapping Tool[3]) 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 was downloaded hundreds of times since it was uploaded to the open source platform, sourceforge.net.

3.4.1 Architecture

The overall architecture of BPEL4WS 2 OWL-S Mapping Tool consists of three components (i.e. WSDL Parser, BPEL Parser and OWL-S Mapper) as shown in Figure 9. As it is clear from name that WSDL Parser parses each WSDL file with in mapping project and creates their object view. An important feature of WSDL Parser is that it extracts information about operations supported by a Web services and send this information to OWL-S Mapper which maps each Web service operation to OWL-S *atomic* process. OWL-S Mapperwrites the generated OWL-S *atomic* process in a separate OWL file and saves it in atomic processes directory of mapping project. Atomic processes are used with in *composite* process to define control flow and data flow between process components with in OWL-S composite service.

BPEL Parser traverse through 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 OWL-S Mapper, BPEL parser declares either a *primitive* activity is an I/O activity or not (Section 3.2 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 OWL-S Mapper uses message part of this activity to create input/output of parameters of *composite*

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

process, which ultimately are used to create the *Profile* ontology parameters. If a *primitive* activity is non I/O activity then OWL-S Mapper maps it to *Perform* CC to perform related *atomic* process. Also, BPEL Parser parses *structured* activities in defined control flow of input BPEL process and sends information about these activities to OWL-S Mapper. The OWL-S Mapper translates them to relevant CCs to define control flow of mapped OWL-S composite service. If BPEL Parser sends information to OWL-S Mapper about *Assignment* activity then OWL-S Mapper traverse through list of existing *atomic* processes to extract input/output parameters of these processes matches them with <*copy*> <*to*> and <*copy*> <*from*> parameters of *Assignment* activity and creates data flow between relevant process components. If a BPEL parser comes to some conditional *structured* activity then it simply sends condition string to OWL-S Mapper which creates corresponding SWRL expression (as explained in Section 3.1.3).

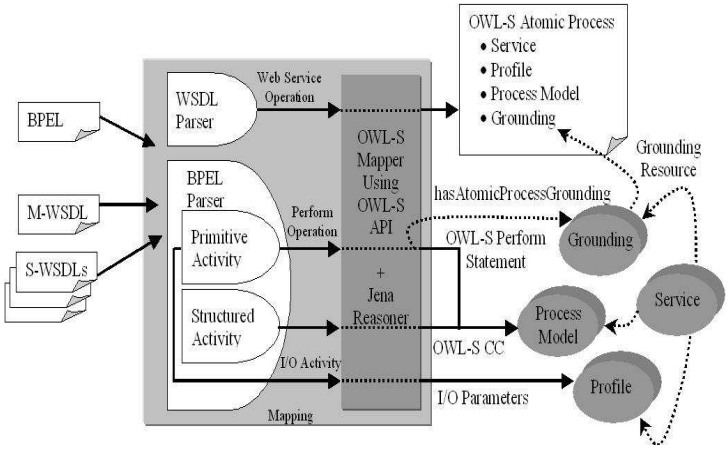

Figure 9. Architecture of the mapping tool.

OWL-S Mapper is actually responsible for writing resulting OWL-S service according to defined mapping specifications. It uses OWL-S API 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 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.

3.4.2 User Interface

The BPEL4WS 2 OWL-S Mapping Tool provides 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 10.

Project Explorer can be used to see project input and output files. Object Explorer provides object view of input BPEL and WSDL files. 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.

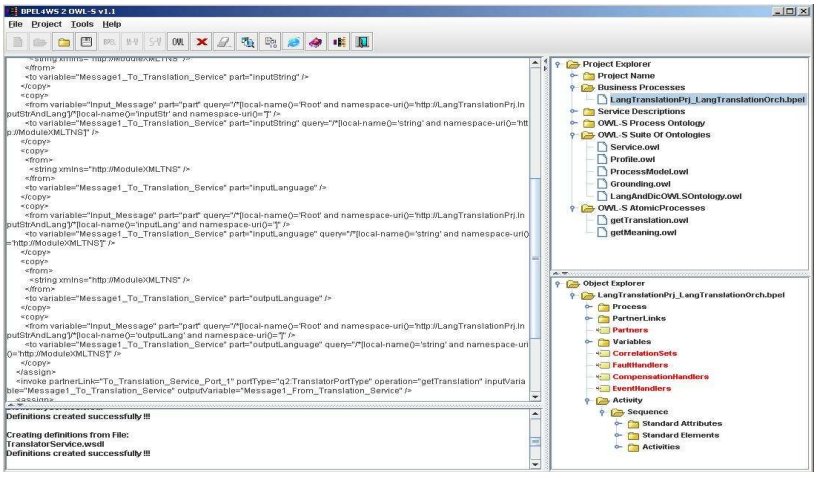

Figure 10. Overview of BPEL4WS 2 OWL-S Mapping Tool.

4 Some Related Efforts

Translating business process descriptions to OWL-S ontologies is a very efficient and cost effective way for enabling business processes with semantics to facilitate dynamic interaction between business partners. Several efforts have already been done to address semantic limitations of process modeling languages. For example, the METEOR-S research group at LSDIS Lab is working on extending BPEL with semantics to compose Web services (i.e. WSDL-S services) on the basis of matching semantics. The work discussed in (Jun et al., 2005; Jun., et al., 2006) describes mapping from the BPEL process model to the OWL-S *Process Model* ontology. We have already criticized and pointed out drawbacks of this approach in our work (Aslam., et al., 2006). Major drawback of (Jun., et al., 2006; Jun., et al., 2005) are that they do not support the *Profile* and the *Grounding* ontologies. Without *Profile* ontology, mapped BPEL process model cannot be advertised as OWL-S SWS that can be discovered, invoked and composed dynamically. The work discussed in (Massimo, et al., 2003) describes a good effort to map WSDL services to DAML-S (updated to OWL-S) services. Another effort (Gayathri, & Yun-Heh, 2007) 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* ontology. The work discussed in (Gayathri, & Yun-Heh, 2007) also supports only the mapping of FBPML process model to OWL-S *Process Model* ontology. It does not support the mapping of *Profile* and *Grounding* ontologies. The work discussed in (Gayathri, & Yun-Heh, 2007) has almost same limitations as that of the work discussed in (Jun., et al. 2006; Jun., et al., 2007) and which I have criticized in (Aslam, et al., 2006). We can summarize that there have many efforts been done to address semantic limitations of process modeling languages by mapping them to semantic Web services 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 translation (mapping) of BPEL process descriptions to complete OWL-S suite of ontologies. We have also well addressed the issues (e.g. conditions mapping, support for complex messages, mapping of *atomic* processes etc.) that have not been addressed by any other research group. Another uniqueness of our work is that we use the OWL-S API in our tool to write the resulting OWL-S service due to which it becomes consistent with execution engines like OWL-S API and semantic Web services development tools (e.g. OWL-S Editor).

5 Evaluation and Benefits

In Section 1.4 we defined two problem scenarios (as show in Figures 4 and 5) and modeled a BPEL process to perform the task defined in first scenario. Then we analyzed BPEL and OWL-S processes and their components and defined step-by-step translation of BPEL process to OWL-S SWS. Till the end of Section 3 the whole BPEL process was mapped to OWL-S SWS with each Web service operation within the BPEL process model mapped to OWL-S *atomic* process.

As a first step to edit the mapped OWL-S service to perform the task defined in second scenario, we replace the 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 (*atomic* and *composite* processes) in OWL-S Editor (even though some compatibility issues between OWL-S Editor and our tool still need to be addressed (as discussed in Section 7) 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. Mapped OWL-S service takes *inputStr*, *inputLang* and *outputLang* as inputs of the OWL-S 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 (Figure 5) (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* (*English*), *outputLang* (*German*) and *inputStr* (output of *atomic* process *getMeaningProcess*). The data flow can be defined by using data binding between *atomic* processes (as discussed in Section 3.1.4) on the basis of matching semantics.

In Section 1.4 we defined two major problems of BPEL process i.e. 1) syntactical interface 2) syntax based process modeling (i.e. Web services composition). We address both of these problems by translating BPEL process description to OWL-S suite of ontologies. *Profile* ontology of mapped OWL-S service provides semantically enriched information about BPEL process as OWL-S SWS and 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 5).

With rapidly growing rate of e-shopping it is becoming very important for e-business companies to keep their business processes and services alive with upcoming semantic Web technologies. Adding semantics will enable existing business processes and services for dynamic co-operation with business partners and for dynamic

interaction with end users. But developing semantic enabled business process and services from scratch is very cost effective and time consuming for both small and large organizations. Our approach provides a very efficient solution with respect to cost and time to shift existing business process to SWSs enabling them for dynamic discovery, invocation and composition by other semantic enabled systems.

6 Risk Assessment

Although, the goal of automatic translation is very appealing, the intention may have some threats in practice for a number of reasons. One of them is that OWL-S with respect to its process modeling capabilities is not as mature as BPEL and mapping of block-structured BPEL to semantic based OWL-S is challenging. Since, BPEL is syntactical language and provides no semantic information therefore, in case of complex business processes it may become hectic to develop domain ontologies from scratch and to annotate mapped OWL-S service parameters with these domain ontologies.

Modelling BPEL processes is supported by a number of tools (e.g MS BizTalk Server, IBM WebSphere, SAP NetWeaver etc.). None of them support to export BPEL processes to OWL-S services. Integrating BPEL4WS 2 OWL-S Mapping Tool with these process modeling tools can enable them to export BPEL processes as OWL-S services but this functionality will neither be fully automated nor support full semantics. End user involvements will be necessary to add meaning to each of the process elements and make them machine-readable and understandable. In addition, it will also allow for reasoning on the process descriptions as OWL-S services. Once, BPEL process description is translated to OWL-S SWS and edited to add semantics, will make it possible to automatically assign Web service (or their composition) to each task and to generate final service that can be deployed and executed by SWS execution engines (e.g. OWL-S API).

7 Conclusion and Future Aspects

In this chapter we presented an approach leading towards semantic business processes. The rationale of the proposed approach is that traditional business processes (e.g. BPEL processes) due to their semantic limitations cannot be dynamically discovered, invoked and composed by other semantic enabled systems. These semantic limitations slow down the process of integration between business partners, business organizations and customers. The methodology that we have used to address these limitations of process modeling languages consists of mapping constraints and specifications that can be used to translate BPEL process descriptions into the OWL-S suite of ontologies (i.e. OWL-S SWSs). The resulting OWL-S services are semantic based compositions of child services and expose semantically enriched interfaces. As a result they can be edited on the basis of matching semantics to model more complex services as well as they can be dynamically discovered, composed and invoked by other semantic enabled systems. We have implemented our approach as a mapping tool (i.e. *BPEL4WS 2 OWL-S Mapping Tool*) that can be used to map (translate) BPEL process descriptions to OWL-S services. Critical issues (e.g. the mapping of *condition* statements, translating activities to CCs, generating *Profile* ontology parameter from complex I/O messages etc.) have been addressed by implementing efficient parsing and mapping algorithms. Since, OWL-S is not as mature as BPEL, we have also highlighted different areas where direct translation of activities is not supported. In order to implement direct translation 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 users are required to manually provide additional information (e.g. changing the parameter type by annotating input/output parameters with suitable domain ontologies etc.).

In order to perform more consistent mapping, it will be necessary to address limitations that we have described in our mapping specifications with upcoming OWL-S specifications. Also, making the tool part of some larger framework like Protégé will support end users more efficiently. Such an effort will enable the end user to directly import BPEL processes as OWL-S services in Protégé (OWL-S Editor). It will also become easier for end users to develop domain ontologies and to annotate *Profile* ontology parameters with domain concepts while working in the same framework (i.e. Protégé).

Acknowledgement: Funding for the research leading to these findings is partially provided by the Higher Education Commission (HEC) of Pakistan under the scheme “Partial Support Scholarship for PhD Studies Abroad”.

References

Aslam, M., A., Sören, A., Jun S., & Michael, H. (2006). Expressing business process model as owl-s ontologies. In E. Dustdar (Ed.), Proceedings of the 2nd International Workshop on Grid and Peer-to-Peer based Workflows in conjunction with the 4th International Conference on Business Process Management, Volume 4103/2006 (pp. 400-415), Vienna, Austria.

Christoph, B., Emilia, C., Dieter, F., Juan, M., G., Armin, H., Thomas, H., Michael, K., Adrian, M., Matthew M., Eyal, O., Brahmananda, S., Ioan, T., Jana, V., Tomas, V., Maciej, Z., & Michal, Z. (2005). Web service execution environment (wsmx). Retrieved April 20, 2007, from http://www.w3.org/Submission/WSMX/.

Dan, B., & Ramanathan, V., G. (2004). RDF vocabulary description language 1.0: RDF schema. Retrieved May 22, 2007, from http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

Daniel, E., Grit, D., David, M., Fred, G., John, K., Shahin, S., & Rukman, S. (2005). The OWL-S editor - A development tool for semantic web services. In A. Gomez-Perez & J. Euzenat (Eds.), The Semantic Web Research and Applications, 2nd European Semantic Web Conference, Volume 3532 (pp. 78-92), Heraklion, Crete, Greece.

David, B., & Canyang, K., L. (2006). Web services description language (WSDL) version 2.0 part 0: Primer. Retrieved April 11, 2007, from http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327.

David, M., Mark, B., Jerry H., Ora, L., Drew, M., Sheila, M., Srini, N., Massimo, P., Bijan, P., Terry, P., Evren, S., Naveen, S., & Katia, S. (2006). Owl-s: Semantic markup for web services. Retrieved April 10, 2007, from http://www.ai.sri.com/daml/services/owl-s/1.2/overview/, March 2006.

Deborah, L., M., & Frank V., H. (2004). OWL web ontology language overview. World Wide Web Consortium, Recommendation REC-owl-features. Retrieved April 17, 2006, from http://www.w3.org/TR/owl-features/.

Evren, S. (2006). Owl-s api. Retrieved May 20, 2006, from http://www.mindswap.org/2004/owl-s/api/.

Evren, S., Bijan P., & James, H. (2005). Template-based composition of semantic web services. In AAAI Fall Symposium on Agents and the Semantic Web, Virginia, USA.

Francisco, C., Hitesh, D., Yaron, G., Johannes, K., Frank L., Kevin, L., Dieter, R., Doug, S., Siebel, S., Satish, T., Ivana, T., Sanjiva, W. (2003). Business process execution language for web services. Retireved April 5, 2007, from ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf.

Frank, L. (2001). Web Services Flow Language (WSFL 1.0). Retrieved May 14, 2007, from http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Gayathri, N, & Yun-Heh, C. (2007). Translating Fundamental Business Process Modelling Language to the Web Services Ontology through Lightweight Mapping. IET Software Journal, 1, 1-17.

Graham, K., & Jeremy, J., C. (2004). Resource description framework (RDF): Concepts and abstract syntax. Retrieved May 25, 2007, from http://www.w3.org/TR/2004/REC-rdf-concepts-20040210.

Holger, K., Mark, A., M., & Alan, L., R. (2004). Editing description logic ontologies with the Protégé OWL plugin. In V. Haarslev and R. Moller (Eds.), Proceedings of the 2004 International Workshop on Description Logics volume 104 (pp. 70-78), Whistler, BC, Canada.

Joel, F., & Holger, L. (2006). Semantic annotations for wsdl. Retrieved March 26, 2007, from http://www.w3.org/TR/2006/WD-sawsdl-20060928/

John, H., G., Mark, A., M., Ray, W., F., William, E., G., Monica, C., Henrik, E., Natalya, F., N., & Samson, W., T., (2003). The evolution of protégé: an environment for knowledge-based systems development. International Journal of Human-Computer Studies, 58(1), 89-123.

Jos D., B., Holger, L., Axel, P., Dieter, F. (2006). The web service modeling language WSML: An overview. In Y. Sure and J. Domingue (Eds.), Proceedings of 3rd European Semantic Web Conference, Volume 4011 (pp. 590-604). Budava, Montenegro.

Jun, S., Georg, G., Yun, Y., Markus, S., Michael, S., Thomas, R. (2006). Analysis of business process integration in web service context. In The International Journal of Grid Computing: Theory, Models and Applications, 23 (3), 283-294.

Jun, S., Yun, Y., Chengang, W., & Chuan, Z. (2005). From BPEL4WS to OWL-S: Integrating E-business process descriptions. In Proceedings of International Conference on Services Computing, Volume 1 (pp. 181-190), Orland, FL, USA.

Massimo, P., Naveen, S., Katia, P., S., & Takuya, N. (2003). Towards a semantic choreography of web services: From WSDL to DAML-S. In L. Zhang, (Ed.), Proceedings of the International Conference on Web Services (pp. 22-26), Las Vegas, Nevada, USA.

Matjaz J., Benny, M., & Poornachandra, S. (2004). Business Process Execution Language for Web Services: A Practical Guide to Orchestrating Web Services Using BPEL4WS. PACKT Publishing.

Matthew, H., Holger, K., Alan, R., Robert, S., & Chris, W. (2004). A practical guide to building owl ontologies using the protege-owl plugin and co-ode tools edition 1.0. The University of Manchister, UK and Stanford University, USA.

Nigel, S., Tim, B., & Wendy, H. (2006). The semantic web revisited. IEEE Intelligent Systems, 21(3),96-101.

Peter, F., P., (2005). Requirements and non-requirements for a semantic web rule language. In Rule Languages for Interoperability.

Satish, T. (2001). XLANG web services for business process design. Retrieved May 1st, 2007, from http://xml.coverpages.org/XLANG-C-200106.html

Sinuhe, A., Emilia, C., John, D., Cristina, F., Dieter, F., Birgitta, K., Holger, L., Axel, P., & Michael, S. Web service modeling ontology primer (2005). Retrieved June 12, 2007, from http://www.w3.org/Submission/WSMO-primer/.