Towards Integration of Business Processes and Semantic Web Services

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

DISSERTATION

zur Erlangung des akademischen Grades

Doctor Ingenieur
(Dr.-Ing.)

im Fachgebiet

Informatik

vorgelegt

von Muhammad Ahtisham Aslam

geboren am 25 September 1979 in Jaranwala

Die Annahme der Dissertation haben empfohlen:

1. Prof. Dr. Ing. habil. Klaus-Peter Fähnrich
2. Prof. Dr. Ing. Wilhelm G. Spruth
3. Prof. Dr. Yun Yang

Die Verleihung des akademischen Grades erfolgt auf Beschluss des Rates der Fakultät für Mathematik und Informatik vom ..... mit dem Gesamtprädikat ..... .

Dedication

To my brother ”Mian Arshad”
Who, for me is
Like my father
Like my mother
Like my sister
Like my friend
Who is not GOD
But
Like GOD for me

Acknowledgements

I would like to thank a number of people who have guided, assisted and supported me in my research work and in writing this thesis. First of all, I would like to thank my supervisor, Prof. Dr. Klaus-Peter Fähnrich who guided, supported and helped me throughout my PhD work. Prof. Fähnrich provided an excellent and challenging research environment to a number of researchers under the roof of Betriebliche Informationssysteme (Business-oriented Information Systems) at University of Leipzig.

Out of other researchers at Business-oriented Information Systems research group, I would like to say special thanks to Dr. Sören Auer who guided, helped, supported and motivated me in every aspect of my research work. I would feel happy to admit that continuous guidance and encouragement of Dr. Sören Auer helped me in completing my thesis which otherwise was looking very tough and difficult to me.

I would also like to thank Dr. Jun Shen (from the University of Wollongong) and Michael Herrmann (from DaimlerChrysler AG) for their support and co-operation in my research work.

It is important to note that this work is partially supported by the Higher Education Commission (HEC) (www.hec.gov.pk) of Pakistan under the scheme "Partial Support Scholarship for PhD Studies Abroad".

At the end, I would like to thank my family and friends who encouraged me at every stage of my work. Even though they were not able to give me any technical support but it was their moral support which motivated me and encouraged me at every tough time. I would also like to thank some one, to whom I can not mention here and whose expected arrival in my life was a big motivation for me to complete my PhD work.

Muhammd Ahtisham Aslam
25-05-2007

Abstract

Business processes are modeled as syntax based compositions of multiple services to perform tasks that a single Web service alone can not perform. When these processes are exported as services they have same syntactical limitations as traditional WSDL services resulting in clampdown for their dynamic discovery, invocation and composition by other semantic enabled systems. Successfully translating existing business processes to semantic Web services can help to address syntactical limitations of business processes and enabling them for semantic based composition editing, modeling and for dynamic discovery, invocation and composition by other semantic enabled systems.

The aim of this thesis is to bridge the semantic gap between business processes and semantic Web services. Bridging the semantic gap between business processes and semantic Web services can help 1) to edit and model the compositions of Web services on the basis of matching semantics 2) to expose semantically enriched interfaces of business processes that can be used for dynamic and automated discovery, invocation and composition of business processes as semantic Web services.

The approach presented in this thesis describes solutions for bridging the semantic gap between syntax based and semantic based composition of Web services both at architectural as well as technical levels. To meet architectural requirements, a new 4-tier semantic Web service integration and composition architecture has been presented. The proposed 4-tier architecture addresses issues like developing domain ontologies, describing semantics of Web services, interfacing between different layers of integration architecture and semantic enhancements in Web service related machinery (e.g. UDDI). The approach presented in this thesis uses upcoming semantic Web service language (i.e. OWL-S) to address syntactical limitations of traditional business process modeling language (i.e. BPEL) by mapping BPEL processes to OWL-S services. The *Process Model* ontology of OWL-S suite is used to define the semantic based composition of services by translating BPEL process model (which is syntax based composition of Web services) to OWL-S *composite* process (which is semantic based composition of Web services). Each Web service operation with in a BPEL process model is translated to an OWL-S *atomic* process and the resulting OWL-S composite service is composition of these *atomic* processes with defined control and data flow. The *Profile* ontology of mapped OWL-S service can be used to expose semantically enriched interface of the BPEL process as OWL-S service. This semantically enriched interface can be used for semantic based dynamic discovery, invocation and composition of BPEL process as OWL-S service. The *Grounding* ontology of mapped OWL-S service describes how to interact with the service. A tool has also been developed that can be used to map existing business processes to OWL-S services. An important feature of the implemented tool is that it supports the mapping of BPEL process to complete OWL-S suite of ontologies. Also, each Web service operation with in a BPEL process model is mapped to OWL-S *atomic* process with complete OWL-S suite of ontologies (i.e. *Profile, Process Model* and *Grounding*).

The main contributions of this thesis can be summarized as follows: First of all a new 4-tier architecture for semantic Web service composition and integration has been presented. On the basis of 4-tier architecture I proposed a semantic Web service composition and integration life cycle and a framework for semantic based composition of Web services. The framework consists of four components and each component is responsible to perform a specific task (e.g. discovery, selection, composition and execution) in the whole semantic Web service integration and composition life cycle. Second, I describe mapping constraints that can be used to establish the correspondence between syntax based and semantic based compositions of Web services. Third, on the basis of mapping constraints I present mapping specifications and algorithms that can be used to translate existing BPEL processes to OWL-S suite of ontologies. Fourth, a tool (BPEL4WS 2 OWL-S Mapping Tool) has also been developed that can be used to translate existing BPEL processes to OWL-S services. Mapping BPEL processes to OWL-S services overcomes syntactical limitations of BPEL processes and enables them for semantic based editing and modeling of Web services compositions. Also, the BPEL process mapped to an OWL-S service can be used for dynamic and automated discovery, invocation and composition by other semantic enabled systems. Finally evaluation of the proposed work has been provided by implementing it in a use case scenario.

Contents

List of Figures

List of Tables

List of Algorithms

Publications

- M. A. Aslam and Sören Auer: Book Chapter for Semantic Web Methodologies for E-Business Applications: Ontologies, Processes and Management Practices Book, García, R. (ed.), Idea Group Publishing, book scheduled for publication in 2008.
- M. A. Aslam, S. Auer, J. Shen and K. Fähnrich: Bridging the Semantic Gap Between Business Processes and Semantic Web Services. Journal of Internet Technology, Taiwan, ISSN: 1607-9264 Vol.8 No.4 2007-TAAI 2006 Special issue-06, pp 433-443.
- M. Herrman, M. A. Aslam and O. Dalferth: Applying Semantics (WSDL, WSDL-S, OWL) in Service Oriented Architectures (SOA). Proceedings of the 10th Intl. Protégé Conference, July 15-18, 2007, Budapest, Hungary.
- M. A. Aslam, S. Auer, J. Shen, M. Herrmann: An Integration Life Cycle for Semantic Web Services Composition. Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD 07), April 26-28, 2007, Melbourne, Australia, ISBN 1-4244-0962-4, pp 490-495.
- M. A. Aslam, S. Auer, J. Shen, M. Herrmann: Web Services Composition to Facilitate Grid and Distributed Computing: Current Approaches and Future Framework: Proceedings of 4th International Workshop on Frontiers of Information Technology (FIT 2006), December 20-21, 2006, Islamabad, Pakistan.
- M. A. Aslam, M. Herrmann, S. Auer, R. Golden: Real-life SOA experiences and an Approach Towards Semantic SOA. Proceedings of 4th International Workshop on SOA and Web Services in conjunction with ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2006), October 22-26, Portland, Oregon, USA, ISBN 82-997428-0-3, pp. 72-81.;
- M. Herrmann, M. A. Aslam: Mercedes Car Group (MCG) Enterprise Architektur - Ein Ansatz zur semantischen Modellierung der Services in einer SOA. In: Fähnrich, K.-P., Kühne, S. Speck, A. Wagner, J. (Hrsg.): Integration betrieblicher Informationssysteme: Problemanalysen und Lösungsansätze des Model-Driven Integration Engineering, Leipziger Beiträge zur Informatik: Band IV. Leipzig: 2006, S. 145-151, ISBN-10: 3-934178-66-9, ISBN-13: 978-3-934178-66-3.(German Paper).
- M. A. Aslam, S. Auer, J. Shen, M. Herrmann: Expressing Business Process Model as OWL-S Ontologies. Proceedings of the 2nd International Workshop on Grid and Peer-to-Peer based Workflows (GPWW 2006) in conjunction with the 4th International Conference on Business Process Management (BPM 2006), Vienna, Austria, LNCS 4103 , Sept. 4, 2006, pp.400-415.
- M. A. Aslam, S. Auer, J. Shen: From BPEL4WS Process Model to Full OWL-S Ontology. In proceedings of Posters and Demos 3rd European Semantic Web Conference (ESWC 2006), Budva, Montenegro, June 11-14, 2006, pp. 61-62.

Chapter 1

Introduction

The goal of this work is to bridge the semantic gap between business processes and semantic Web services to enable business processes for 1) semantic based editing and modeling of Web services compositions 2) to expose semantically enriched interfaces of business processes that can be used for dynamic and automated discovery, invocation and composition of business processes as semantic Web services. For this purpose a 4-tier semantic Web service composition and integration architecture, semantic Web service composition life cycle and a framework for dynamic and automated Web service composition has been presented. Based on these theoretical concepts I have presented an approach that can be used to shift existing business processes to semantic Web services. A prototypical tool (BPEL4WS 2 OWL-S mapping tool[1]) has also been developed for translating existing business processes (BPEL processes) to semantic Web services (OWL-S services).

In this chapter I give an overview of this document. First of all I describe the motivation for my work. Then I give an example scenario that helps to understand the research problem in broader sense and to set boundaries of my work. Then I discuss the research questions with in defined problem domain. After providing a short background of my work, I describe the proposed solution as in the form of my research contributions. At the end of this chapter I give the outline of my thesis.

1.1 Motivation

Rapidly changing trend of developing business applications as services resulted in quick adoption of Web services. With this wide acceptance of Web services different architectural approaches (e.g. [56]) for integration of Web services and workflow languages (e.g. BPEL4WS [42, 62], MS XLANG [59, 13], IBM WSFL [68]) have been developed. These workflow languages can be used to model business processes as syntax based compositions of multiple Web services to perform complex tasks that a single Web service alone cannot perform. Major drawbacks of these languages are 1) they compose Web

[1] BPEL4WS 2 OWL-S Mapping Tool is open source project and it can be downloaded from: http://bpel4ws2owls.sourceforge.net/

services on the basis of their syntactical information following the traditional 3-tier business application integration architecture 2) when these processes are exported as services they have same syntactical limitations as traditional WSDL [31] services. Modeling Web services compositions and discovering, invoking and composing them on the basis of syntactical information is inefficient and unreliable approach. Semantic Web and semantic Web services (*in remaining thesis I will write the term semantic Web service as "SWS" and semantic Web services as "SWSs"*) community is working on different languages (e.g. OWL-S [71, 72], WSDL-S [15, 77] and WSMO [19, 51]) to provide Web service semantics and approaches to dynamically discover, invoke and compose these services on the basis of matching semantics. In the early stage of my research work I realized that with semantic enhancements in Web services, a new architectural approach is needed that can be used to integrate these semantically enriched business services. Also, in addition with semantic based Web service integration and composition architecture an approach is needed to transfer existing business processes (e.g. BPEL processes which are syntax based compositions of Web services) to semantic based compositions of Web services (e.g. OWL-S composite services). Such an approach will help not only to edit and model the composition of services on the basis of matching semantics but also to provide semantically enriched information about processes (BPEL processes) as OWL-S composite services. This semantically enriched information can be used for dynamic discovery, invocation and composition of processes as SWSs.

Enhancing existing business processes with semantics and enabling them for semantic based composition editing, modeling and for dynamic discovery, invocation and composition needs to address the following major research problems:

- Developing architecture for integration and composition of semantic based Web services.
- Establishing the correspondence between syntactical and semantic based Web services composition. Establishing such a correspondence includes:
  - Providing semantics of individual services with in processes.
  - Modeling the composition of services on the basis of matching semantics.
  - Providing semantics of composite services (processes) that are modeled as composition of multiple SWSs.

By successfully addressing these problems we can:

- Provide a basic architecture needed to integrate and compose SWSs on the basis of matching semantics.
- Provide semantics of processes as composite services for the purpose of dynamic discovery, invocation and composition.
- Composition can further be edited on the basis of matching semantic information rather than syntactical information to model more complex services.

The aim of this thesis is to shift existing business processes from a syntactical to semantic based environment rather than to build semantic enabled business applications (processes) from scratch. For this purpose I analyzed the problem from ground resulting in my research contribution as theoretical approach and implemented tool that can be used to address the mentioned research problem. In next section I provide an example scenario which will help to understand the problem at more concrete level.

1.2 An Example Scenario

In order to understand the problems raised due to syntactical limitations of BPEL processes we consider an example scenario of syntax based Web services composition (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 (available with tool download). *Translator and Dictionary* process is modeled in MS BizTalk Server [13] as syntax-based composition of two services (i.e. *Translator* service and *Dictionary* service). *Translator* service is a Web service that can be used to translate a string from one language to another supported language. *Dictionary* service is a Web service that can be used to get the meaning of an English word in English (i.e. only English language is supported by Dictionary service). Now I define two problem tasks that can not be performed by anyone of these two services (i.e. *Translator* Service or *Dictionary* Service). These two tasks are:

1. How we can get the meaning of a *German* word in *English*? Because *Dictionary* service supports only meaning of an *English* word in *English*, not the meaning of a *German* word in *English*.

2. How we can get meaning of a *German* word in *German*? Because *Translator* service only translates string from one language to other language (not give the meaning of a word) and *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 tasks. As a solution, I model a BPEL process as syntax based composition of these services (i.e. *Translator* and *Dictionary* services) to perform the task defined in first scenario. Then I highlight what are limitations of such a syntax based Web services composition (process). In remaining chapters of this thesis I describe architectural and conceptual aspects of such a syntax based Web services composition and provide their solutions. I also describe a strategy to translate syntax based Web services compositions to semantic based Web services compositions. The process modeled to perform the task defined in first scenario consists of the following steps (as show in Figure 1.1):

1. Process accepts input string (i.e. *German* word) from user (may be another service).

2. Transfer this string as an input of the *Translator* service to translate string from *German* to *English*.

3. Output of the *Translator* service (i.e. *English* translation of input string) is given as an input to the *Dictionary* service.

4. As a last step of the process, the *Dictionary* service returns meaning of the input string.

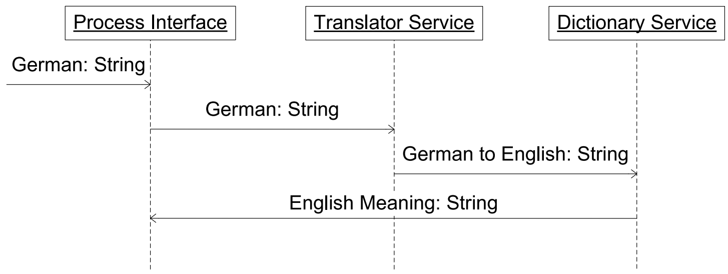

Figure 1.1: Sequence of services in process according to first scenario.

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

1. When a process is exported as a Web service, it has same syntactical limitations as traditional WSDL service (i.e. syntactical interface) resulting in clampdown of process for dynamic discovery, invocation and composition.

2. If we want to extend the process (discussed in first scenario (Figure 1.1)) in a semantic environment to perform the task pointed in second scenario (Figure 1.2) then we will realize that:

   (a) Web services with in composition provide no information for semantic based editing and modeling of process. For example consider input message (Example 1) required by *Translator* service. This message provides no semantic information about message parts (i.e. "inputString", "inputLanguage" and "outputLanguage").

   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>
   ```

   (b) Semantic limitation of Web services with in process restrict to dynamically discover and compose (on the basis of matching semantics) other SWSs (e.g. semantically enriched *Translator* service).

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 "inputLanguage" with ontology concept "SupportedLanguage" defined in appropriate domain ontology. Providing such semantic information can help to:

- Provide semantically enriched interface of the process as an OWL-S composite service that can help in dynamic discovery, invocation and composition of BPEL process as a SWS.
- Translate the process from syntax-based to semantic based composition which provides semantically enriched information about each service involved with in the composition.
- Edit and model the composition on the basis of matching semantics rather than relying just on syntactical information.
- Defining abstract process (semantically enriched Web service request) with in composition to dynamically discover and compose a service on the basis of matching semantics defined in abstract process (according to the approach discussed in [100]).
- Using an AI planning for automated composition by mapping OWL-S *composite* and *atomic* processes to tasks and operators of the planning languages (e.g. HTN planning).

Example 2. Semantically enriched message part.

```
<process:Input rdf:ID="inputLanguage">
 <process:parameterType rdf:datatype="&xsd;#anyURI">
  &languages;#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 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 (tasks) for modeling processes as Web services composition. For first scenario we modeled a BPEL process in MS BizTalk Server (BPEL file of the process is attached in Appendix A). Then I highlighted limitations of such syntax based process modeling. In Chapters 4 and 5, I provide a detail analysis of BPEL process models and OWL-S SWSs and then on the basis of this analysis I define the mapping specifications. In remaining chapters I use this BPEL process to provide some code samples of mapping specifications. Until we reach the evaluation chapter (Chapter 7) the whole BPEL process is mapped to OWL-S service. Then I use this mapped OWL-S service to answer the problem questions (discussed above). In evaluation chapter (Chapter 7) I enhance the mapped OWL-S service (*Process Model* ontology) in semantic environment (e.g. Protégé [55, 12] (OWL-S Editor [47, 10]) or even with simple editor like Note pad) to develop OWL-S composite service (SWS) for second scenario (i.e. getting the meaning of *German* word in *German*) by editing and extending mapped OWL-S service (on the basis of matching semantics) by the following steps (as shown in Figure 1.2):

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

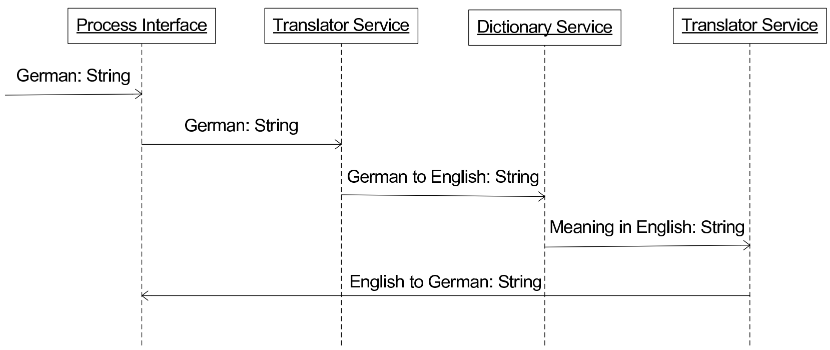

Figure 1.2: Sequence of services in process according to second scenario.

The evaluation section also describes how the *Profile* ontology of the mapped OWL-S service provide semantics of BPEL process as OWL-S service for the purpose of business process automation.

1.3 Research Questions

The overall research question that I tried to answer in this thesis is:

How existing business processes can be shifted from a syntax based to semantic based environment to enable them for semantic based composition editing, modeling and dynamic discovery, invocation and composition by other semantic enabled systems?

To answer this question in detail I define a set of research questions. These research questions actually are sub part of the main research problem. Answers to these questions help to understand the main research problem in small steps.

RQ 1. What Web service is and how we can provide Web service semantics?

- What Web services and its related standards are?
- How we can add semantics to Web services?

RQ 2. Is existing application integration architecture and framework enough for semantic based dynamic integration and composition of business processes as SWSs?

- How workflow and AI planning can effect semantic based composition of business processes as SWSs?
- Is traditional syntax based application integration architecture is enough for composition and integration of SWSs?
- In a semantic enabled composition process, how business and technical constraints can be used?

RQ 3. How correspondence can be established between syntax and semantic based composition of Web services (i.e. BPEL process model and OWL-S composite service)?

- How we can semantically express components of a business process (which is composition of syntax based Web services)?
- How a SWS language (e.g. OWL-S) can be used to model the composition of Web services on the basis of matching semantics?

RQ 4. How a BPEL process model can be mapped and expressed as OWL-S SWS?

- How we can extract information about interface of BPEL process model and express it semantically?
- How the information about control and data flow can be extracted from a BPEL process model and expressed in SWS language (i.e. OWL-S)?
- How interaction protocol and complex messages can be extracted from BPEL process model and defined in OWL-S?

RQ 5. Is translation of BPEL process models to OWL-S ontologies can help for semantic based discovery, invocation and composition of BPEL processes as OWL-S services?

- How semantics of a Web service can be used for reasoning?
- How the execution of business processes as OWL-S services can be supported?

1.4 Background

The general research question states that my research efforts aim at addressing syntactical limitations of business processes and enhancing them with semantics to enable them for dynamic discovery, invocation and composition. There were different research directions for choosing this research area. These research directions are described below as motivational background of my work.

- First of all, I was motivated by *Web service initiative* and its definition at an early stage of Web service project. *According to this definition Web service is described as platform independent technology that can be interacted in a computer understandable way.* Web service community was successful in providing a platform independent service technology as WSDL [31] services and its related standards (i.e. WSDL [31], UDDI [41] and SOAP [58]) but seamless interaction between Web services is still an open question. Complex business tasks that a single Web service alone is not able to perform requires to compose multiple services together to perform that task. Such requirements resulted in development of different workflow languages (e.g. BPEL) that can be used to compose multiple Web services to perform a required task. Different tools such as MS BizTalk Server [13], IBM WebSphere [26], SAP Netweaver [17] etc. support the modeling of business processes as composition of multiple services. The major limitation at this stage is lack of semantics in Web services descriptions that result in manual discovery, invocation and composition of these services.

- Second, I was inspired by research efforts in the area of *semantic Web* and *SWSs* to provide computer understandable meanings of Web services. Research initiatives to provide semantics of Web services fall at two levels 1) developing domain ontologies to provide domain specific information 2) developing languages that use this domain specific information to provide Web service semantics. At first level (i.e. developing domain ontologies) OWL is the major language that can be used to develop domain ontologies. Different efforts (e.g. OWL-S, WSMO and WSDL-S) started for the development of SWS languages. The ultimate goal behind all these efforts is to develop a language that can be used to provide semantically enriched descriptions of Web services by annotating them with domain ontologies.

- Third, while talking about developing and integrating business applications, different architectural approaches for integrating business applications have been presented (e.g. [106, 103]). Among these approaches *3-tier architecture* for integration of business applications meets demands of most of integration scenarios. Rapidly changing trend of developing business applications as business services and furthermore semantic enhancements in service technology resulted in some initiatives to develop *SWS integration and composition architectures* (e.g. [56, 39]). Part of my research work was inspired by these initiatives to develop architecture for integration and composition of semantically enriched applications (services).

- Fourth and the most important inspiration of my work was different efforts that have already been done by different research groups to *establish correspondence* *between syntax based and semantic based composition of Web services.* None of these efforts were able to successfully address above discussed research questions. Since, among different SWS languages, OWL-S is the language that supports modeling a composite service by composing different services on the basis of their matching semantics therefore, these research groups worked on transforming the business processes to OWL-S composite service. These efforts were only able to establish the transformation between partial components of these languages. For example the work discussed in [80, 90, 92, 78, 38] describe some efforts to create correspondence between individual components of syntactic and semantic based languages. These efforts resulted as an initial point as well as provided me some future directions for more research work to be done to support different research projects (e.g. SwinDew [89, 91, 94]) to enable them with semantic support and to help existing legacy systems shifting to semantic based environment.

- Fifth motivational background for my work is that a considerable number of research projects are working in this area. For example, *OWL-S API* [96, 99] (by "mindswap") has been developed and is under continuous improvement with the coming versions of SWS language (OWL-S). *OWL-S API* can be used to programmatically read, write and execute SWSs. Currently going on esearch projects (e.g. *Transitioning Applications to Ontologies* (TAO)[2] [82, 29]) also aims at developing methodologies for transitioning existing or legacy systems into reusable, semantically described services. Also, tools like *OWL-S Editor* [48] has been developed to visually edit and model composite services (composition of SWSs). Semantically enriched information about Web services capabilities can be provided by annotating them with domain ontologies (OWL ontologies) developed in semantic Web tools like *Protégé* [55]. *Future work of these research groups also point out needs to develop approaches and their implementation to map Web services composition from syntactical language (e.g. BPEL) to SWS language (e.g. OWL-S).* The resulting OWL-S service can be edited in semantic based visual tool like *OWL-S Editor*. Also, there exist some approaches (e.g. [100]) that can be used to define abstract processes with in a composite service so that services matching to semantics of abstract processes can be dynamically discovered and composed in resulting composite service.

My research work is an effort to achieve business process automation by providing semantically enriched descriptions of business processes. Larger semantic Web frameworks (e.g. Protégé (OWL-S Editor) and SwinDew [91, 94]) have shown their interest in this work to improve their tools and systems to provide richer support for semantic enabled processes. To accomplish research targets and to answer above discussed research questions I use these initiatives and research work that already has been done (as discussed above) in this area. My work uses the most important and industry wide accepted standard (i.e. OWL-S) to bridge the gap between syntax based and semantic based composition of Web services. Also, utilization of this work in collaboration with projects and systems (e.g. SwinDew and Protégé (OWL-S Editor)) provided motivation for this

[2]http://www.tao-project.eu/

work. Also the new SWS integration and composition architecture discussed in this work provides a good base for future development of SWS and semantic based integration tools.

1.5 Research Contributions

I have approached the problem and addressed the above discussed research questions by proposing a new 4-tier SWS integration and composition architecture and a life cycle for SWS composition. A general framework at an abstract level for dynamic and automated composition of Web services has also been presented. I also have presented an approach which addresses issues to establish correspondence between business processes (BPEL processes) and SWSs (i.e. OWL-S services). Mapping specifications and algorithms have been presented to map existing business processes (i.e. BPEL processes) to OWL-S services. A prototypical implementation of the proposed approach have been presented that can be used to map BPEL processes to OWL-S services. My thesis has resulted in following research contributions:

- First of all, a new *4-tier architecture has been proposed to meet integration and composition requirements for integration and composition of business processes as semantically enriched Web services.* The proposed architecture addresses issues (e.g. semantic based Web services interfaces and queries, bridging semantic gap between different integration layers, UDDI enhancements with semantics etc.). The proposed 4-tier architecture has been discussed in my work **[25, 76]**. *On the basis of 4-tier architecture I propose a SWS integration and composition life cycle [24] and a general framework at an abstract level for dynamic and automated composition of business process as SWSs* **[23]**. The composition framework follows the approach of newly proposed 4-tier SWS integration and composition architecture and SWS integration and composition life cycle. **Chapter** 3 covers the proposed architectural approach in more detail.
- Second, mapping constraints on the basis of matching functional characteristics of BPEL activities and OWL-S control constructs have been described in Chapter 4. *Process modeling and semantic capabilities of BPEL process model and OWL-S suite of ontologies have been analyzed in detail and mapping constrains have been defined to establish a correspondence between BPEL and OWL-S.* Mapping constrains also addresses mapping issues very well for activities which have dual behavior with in BPEL process model.
- *Third, mapping specifications and mapping algorithms have been described in Chapter 5.* Mapping specifications shows that how OWL-S suite of ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies) can be extracted from BPEL process model. It also aims at describing that how control flow and data flow can be defined between child processes with mapped OWL-S composite service. Mapping algorithms show that how efficiently different BPEL activities can be mapped to OWL-S control constructs.
- *Fourth, I have developed a tool (BPEL4WS 2 OWL-S Mapping Tool) as an implementation of above mentioned mapping strategy. BPEL4WS 2 OWL-S Mapping*

Tool can be used to map BPEL processes to complete OWL-S suite of ontologies. In implementation the tool I have used the research work that has already been done in this area by other research groups. For example, I have used OWL-S API [96] to write the resulting OWL-S ontology for mapped OWL-S service. A component of the tool (i.e. OWL-S Mapper) uses OWL-S API for writing resulting OWL-S services. Since, OWL-S API uses Jena reasoner [9] for reasoning the mapped OWL-S ontology therefore, I have also used the Jena reasoner as part of my implementation. I have also explored (as discussed in Section 1.4) and criticized some initial work done by other research groups in this area. In our work **[20, 22, 21]** I have pointed out limitations and drawbacks of previous work done by other research groups in this area and have shown how our work provide a more consistent and practical approach. **Chapter** 6 discusses the implementation and architecture of the tool in detail.

- *Fifth, in **Chapter** 7, I provide an evaluation of proposed approach and its prototypical implementation.* In this chapter I describe that how the approach presented in this thesis addresses syntactical limitations of process modeling language (i.e. BPEL) that have pointed out in Section 1.2 and enable existing business processes for semantic based composition editing, modeling and for dynamically discovering, invoking and composing them on the basis of matching semantics. In Chapter 8 I point out some limitations and give future directions to make this work more useful for SWS and process modeling communities.

1.6 Thesis Outline

This thesis describes the architecture and framework for dynamic composition of business processes as SWSs. A theoretical approach and its prototypical implementation have been developed to shift existing business processes to SWSs rather than to build them in a semantic enabled environment from scratch. Figure 1.3 describes map of this thesis. An overview of chapters of this thesis is as under:

Chapter 2 describes the state of the art in the area of Web service, semantic Web, SWS and process modeling. This chapter also includes some literature about Web service standards and SWS languages.

Chapter 3 analyzes existing approaches for SWS composition and highlights some issues that need to be addressed for dynamic and automated composition of Web services. To address these SWS composition issues new architectural approach and framework has been described in Chapter 3.

Chapter 4 describes mapping constraints that can be used to establish correspondence between syntax based (BPEL processes) and semantic based composition of Web services (OWL-S composite services).

Chapter 5 presents mapping specifications and algorithms that can be used to translate BPEL process descriptions to OWL-S suite of ontologies.

Chapter 6 describes the tool (BPEL4WS2OWL-S Mapping Tool) developed on the basis of architectural concepts and mapping specifications discussed in previous Chapters 4. This chapter also describes the architecture of the implemented tool in detail.

Chapter 7 provides an evaluation of the proposed work by implementing it in a sample use case scenario.

Chapter 8 concludes and describes future directions for my work.

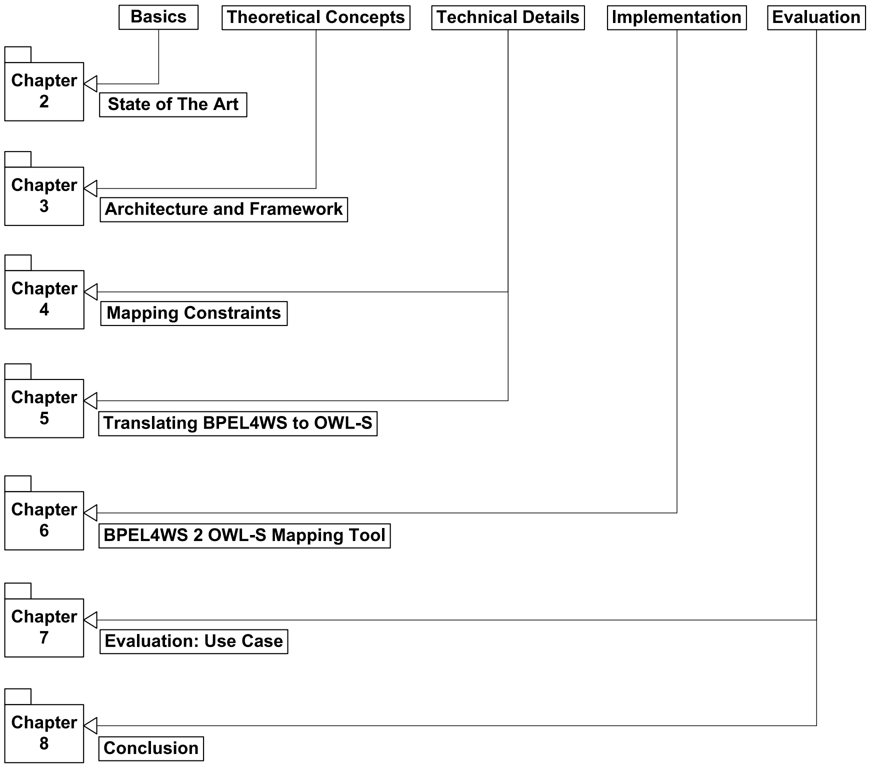

Figure 1.3: Overview of this document.

Chapter 2

Semantic Web Service: State of The Art

In this chapter I describe state of the art in the area of Web services, semantic Web, SWSs and business process modeling. This chapter does not aim to provide complete description of these technologies (as specifications of these technologies provide their detailed technical descriptions) but to describe basic concepts, introduce necessary terminologies and to provide technical overview of important technologies that are major part of this work.

2.1 Introduction

Investigating capabilities and limitations of Web services, SWSs and SWS languages that can be used to overcome syntactical limitations of process modeling languages (e.g. BPEL) was a preliminary step of my research efforts. For this purpose I inquire in detail capabilities of Web service with its related standards (i.e. WSDL [31], SOAP [58] and UDDI [41]) and Web service working architecture (i.e. Service Oriented Architecture (SOA) [30, 49]). I argue that with semantic enhancements in Web service, semantic enhancements in Web service related machinery (e.g. service requester, service provider and service registry) are also needed. Also, the approach followed by traditional 3-tier application integration architecture is not enough for integration and composition of semantically enriched services. I also 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 compositions of Web services. Then I describe the vision of the semantic Web and provide a short overview of semantic Web languages (e.g. RDF [64], RDF-S [33] and OWL [73]). I provide some technical details about semantic Web language (i.e. Web Ontology Language (OWL)) and how OWL ontologies can be used to provide machine understandable meanings of data. I also describe that how SWS community makes use of semantic Web language (i.e. OWL) to provide machine understandable meanings of Web services. I provide short technical descriptions of SWS languages (e.g. OWL-S, WSDL-S, WSMO) and compare them with respect to their semantic and workflow modeling capabilities. By analyzing and comparing existing SWS languages I argue that semantic and process modeling capabilities of OWL-S are much batter as compare to other SWS languages and it can be used to address semantic limitations of traditional process modeling languages (e.g. BPEL). Understanding the terms and technologies described in this chapter will help to better understand the work discussed in remaining chapters of this thesis.

The remaining chapter is organized as follows: Section 2.2 describes Web service technology and highlights its capabilities and limitations. SOA and semantic enhancements in SOA are discussed in Section 2.3. Section 2.4 describes workflow modeling and standard workflow modeling language (i.e. BPEL). Semantic Web and semantic Web languages that can be used to provide machine understandable descriptions of data have been discussed in Section 2.5. Section 2.6 describes emerging SWS languages and provides a comparison of capabilities and limitations of these languages. Section 2.7 describes AI planning for automated composition of semantically enriched Web services. Section 2.8 summarizes this chapter.

2.2 Web Service

Web services are being used to develop applications as reusable services (Web Services) due to number of benefits of Web service technology. IBM encloses capabilities of Web services by defining it as:

Web Services are self-contained, modular applications, accessible via the Web through open standard languages, which provide a set of functionalities to businesses or individuals [61].

According to W3C definition of Web services:

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions thanks to the use of XML. They can be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services [5].

W3C definition adds more to Web services capabilities highlighting it as standard means of interoperating between different software applications, running on a variety of platforms (platform independent), interoperable and extensible. The work discussed in [6] describes a number of definitions for Web services provided by different IT experts and industrial partners (e.g. IBM, Intel, Microsoft, SUN etc.). Generally describing, the following capabilities make Web services an efficient business applications development and integration technology:

Self-contained: A Web service is a complete set of functionalities. An application when published as a Web service, provides APIs that can be used to avail its functionality by sending and receiving appropriate messages. A Web service in itself can be used to perform a specific task supported by that service. In some cases, Web services need to be composed with other services to provide a combine functionality that a single Web service can not perform (as discussed in Section 2.4).

Interoperability: Web service provides interoperability between applications as well as big vendors. Broad vendor agreement on standards and proven interoperability have set Web services apart from integration technologies of the past [95]. Interoperability feature of Web services make it most suitable technology to develop integrated applications.

Platform Independent: Platform independence is important achievement of Web service technology. Applications developed on any platform can be published and invoked as a service on any other platform. Traditional component development technologies (e.g. COM [1], DCOM [2], CORBA [8] etc.) do not provide with such platform independence. A COM component developed on windows can only be used on windows platform. Web services technology overcomes such platform dependence.

Loosely Coupled: Different technologies (e.g. COM, DCOM, CORBA etc.) were introduced to develop software system as components and modules. The whole software application is combination of these components and modules which are tightly dependent and coupled with each other. Web service as compare to these technology is self-contained and loosely coupled. Emerging semantic Web service technologies aim at developing more dynamic and loosely coupled services that can be dynamically discovered, invoked and composed.

Standard Languages: Another important success factor for Web service is that it is described by using XML [32] based language (i.e. Web Services Description Language (WSDL)). WSDL being XML documents inherits the powers of XML (e.g. flexibility, extensibility etc.). Also, Web services use SOAP as message exchange paradigm and can also be used to create complex interaction patterns (e.g request/response, request/multiple responses etc.).

Universally Accessible: Web services are universally accessible. We can develop applications as services, publish them, discover them and invoke them. Any service provider can develop a service and publish it on global network (possibly some UDDI registry). The provider of the service himself and other business partners can discover and invoke a Web service on defined address regardless of the platform and framework on which Web service was developed and on which it is being used.

Above discussed Web service capabilities played an important role in the success and rapid adoption of Web services. With this rapid adaption of Web services different issues like 1) discovering required services 2) composing Web services to perform a complex task that a single Web service alone can not perform 3) describing Web services in computer understandable way so that Web services can be dynamically discovered, invoked and composed by computer agents 4) development of new semantic based architecture and frameworks for semantic based integration and composition of SWSs raised. To address these issues different workflow modeling, semantic Web and semantic Web service languages were developed and are under continuous development. Figure 2.1 gives an overview of the evolution and relation between these syntax and semantic based languages and we discuss them in coming sections in detail.

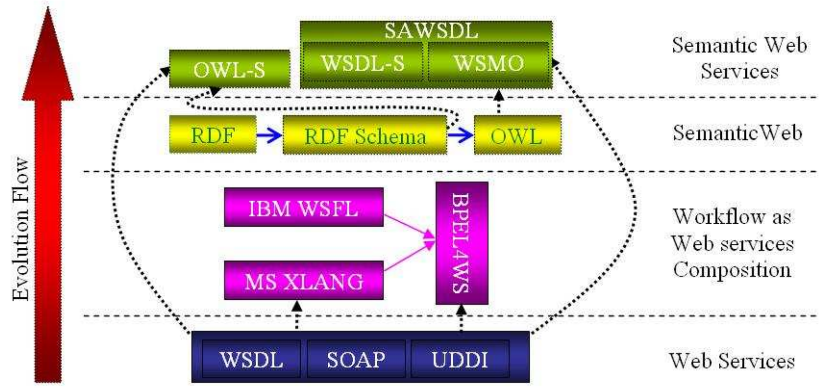

Figure 2.1: Evolution and relation between Web service, workflow, semantic Web and semantic Web service languages.

2.3 Semantic Service Oriented Architecture

Service Oriented Architecture (SOA) [30, 49] is working architecture for Web services and has three participants (i.e. service requester, service provider and service registry). A service provider can develop applications on any platform and in any language and can export them as WSDL services. These WSDL services can be registered on Web service registries (e.g. UDDI registries). A service requester can search for a Web service in Web service registries by using key word based searching. After discovering a required service the service requester can directly interact with the service to utilize its functionality. Limitations of current SOA and service registries are that they support only to publish syntax based WSDL services and required services can be discovered manually on the basis of keyword based searching. Semantic enhancements in Web service resulted in some efforts to semantically enrich Web service related machinery (e.g. semantically enriched Web service registries [108]) so that SWSs can be published and dynamically discovered and composed by computer agents on the basis of matching semantics rather than to find a service manually.

Adding semantics in SOA aims at providing shared meaning of business services within an organization and probably across the organizational boundaries. Traditional SOA has three participants- *Service Provider, Service Requester* and *Service Registry* and semantic enhancements improve the role of these participants of SOA as:

- SWS Provider

- SWS Requester

- SWS Registry

SWS provider can develop and advertise a Web service that provides its machine understandable meaning. Using a SWS language can provide such machine understandable description of Web services. Three major candidates for SWS standards are OWL-S, WSDL-S and WSMO (as shown in Figure 2.2). The SWS provider annotates services with domain ontologies to provide shared meaning of their Web service functionality by using any of these languages. Publishing SWS supposes that the service registry supports such SWS advertisements. The requester of a WS is interested in finding a service that fulfills his functional and non-functional requirements. A requester can find a service manually or can define a Web service request annotated with domain ontologies to provide request semantics (e.g. OWL-S *Profile* ontology). Such semantic requests can be used by computer agents to dynamically find required services (e.g. [102] describes an approach to annotate and discover web services by matching semantics). The work discussed in [63, 66, 69] describe some approaches to automatically locate and discover Web services. Current UDDI structure supports only key word based searching of required services. Such keyword based searching is inefficient and not precise because it finds those services also which are not offering the required functionality. Semantic enhancements in service registries demand more efficient mechanism to discover required services on the basis of matching semantics. Locating required services efficiently (semantically) is required by the semantic enhancements in the service registries. A good work has been done and is continuously improving the semantic base discovery of Web service by improving search algorithms [35] and enhancing the registry architecture [79, 16, 108].

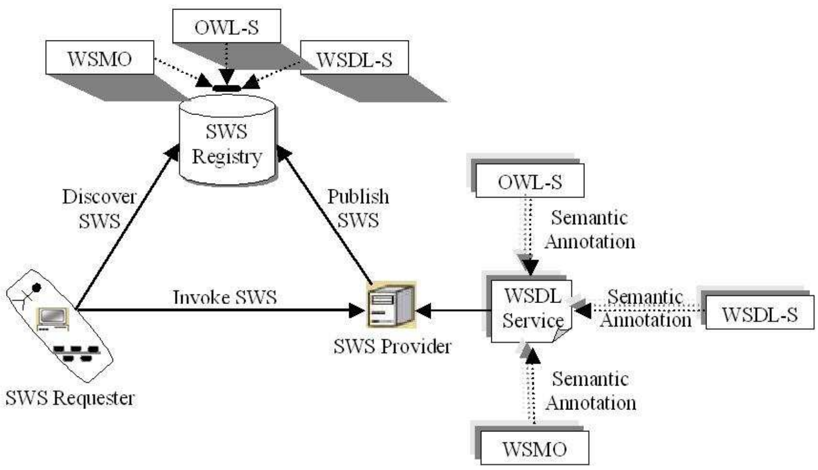

Figure 2.2: Semantics based Service Oriented Architecture.

2.4 Workflow Modeling

Different workflow modeling languages like Web Services Flow Language (WSFL) [68], MS XLANG [59] and Business Process Execution Language for Web services (BPEL4WS, shortly known as (BPEL)) [42] have been developed to define workflows. IBM's WSFL addresses workflow on two levels: 1) it takes a directed-graph model approach for defining and executing business processes 2) it defines a public interfaces that allows business processes to advertise as Web services [88]. The XLANG is an XML [32] based business process language that can be used to orchestrate Web services. An XLANG service description is a WSDL service description with an extension element that describes the behavior of the service as a part of a business process [7]. MS XLANG is a language that is used in MS BizTalk Server (which is Microsoft's business process modeling tool). However, the business processes modeled in MS BizTalk server can easily be imported and exported to BPEL.

BPEL4WS. BPEL is a mature business process modeling language and is the 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. A BPEL process 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 [88]. Several characteristics of BPEL make it the language of choice for modeling business processes. For example, BPEL is a language which 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.) provide support for importing and exporting BPEL processes from one framework to other. In presence of all these capabilities it has many shortcomings resulting in limitations for seamless interoperability of business processes. A BPEL process being a syntax based composition of Web services fails even if a single Web service with in composition is not available or changed with the passage of time. Also, when these BPEL processes are exported as services they expose syntactical interfaces which no more enable them to be dynamically discovered, invoked and composed by other semantic enabled systems. These limitations can be addressed successfully by getting across semantic gap between process modeling languages and upcoming semantic Web and SWS languages (as shown in Figure 2.1 and discussed in remaining chapters of this thesis).

2.5 The Semantic Web

The semantic Web [87, 28] is an extension to the current Web (WWW) to present more meaningful data that is easily and efficiently processable and understandable for humans as well as for machines. It aims at providing common formats for exchanging data and languages for describing relations between data objects.

Different semantic Web languages (e.g. RDF [64], RDF-S [33] and OWL [73, 18]) have been developed to present information as resources on the Web. Uniform Resource

Identifiers (URIs) [27] can be used to uniquely identify entities as resources on the Web. For example, we can assign URI to a student, a university, an address etc. and relation (as shown in Figure 2.3) between these resources can be defined by using semantic Web languages for better and efficient processing of information by human and computer agents.

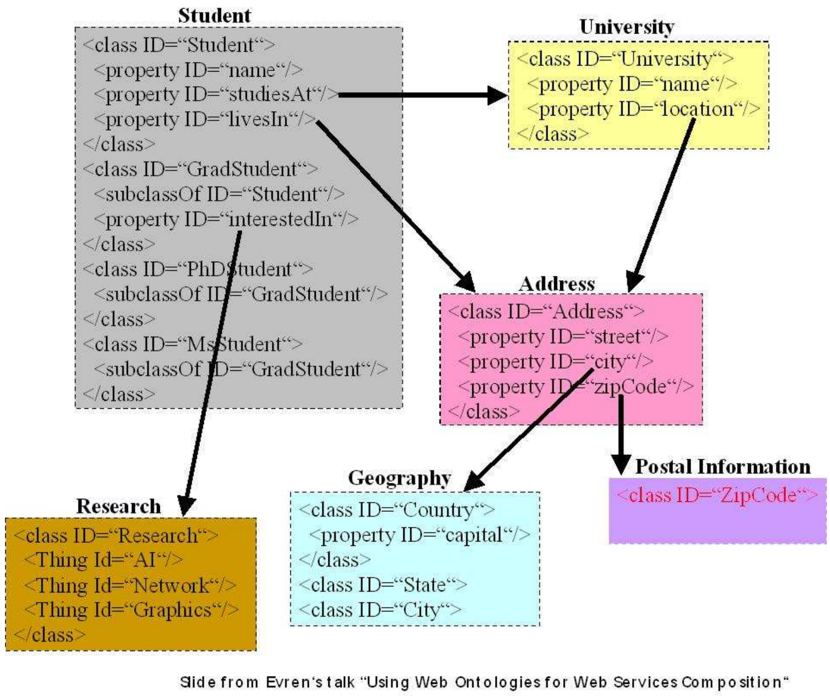

Figure 2.3: Relational semantics defined with OWL ontology.

Among the bundle of semantic Web languages available as W3C recommendations, 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. The 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. Lack of information expression capabilities of RDF-S (e.g. defining properties of properties, necessary and sufficient conditions for class membership, equivalence or disjointness of class etc.) 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 contents only need to be presented to humans [11]. Figure 2.3 (taken from Evren Sirin's talk "Using Web Ontologies for Web Service Composition" [97]) gives a very interesting and easy to understand example of an OWL ontology. This sample ontology defines the relation of a student with his geographical location, university, course etc. This information can be used by computer agents for reasoning and finding suitable student records.

2.6 Emerging Semantic Web Service Languages

The semantic Web introduced the vision of providing machine understandable data and OWL emerged as a language to provide universal meaning to data over the Web. Web services and SWSs communities used the semantic Web vision and semantic capabilities of OWL to make Web services machine understandable for the purpose of dynamic and automated discovery, invocation and composition. Currently different efforts are going on to develop SWS language (e.g. WSDL-S, WSMO and OWL-S). All of these SWS languages working groups are using OWL to provide domain specific semantics of Web services even though their approaches of using OWL ontologies differ from each other.

2.6.1 WSDL-S

WSDL-S is a SWS development language that is jointly developed by the University of Georgia and IBM. The WSDL-S approach is to enhance WSDL tags to provide machine understandable meanings of services. For example, WSDL-S extends WSDL *operation* and *message* tags by annotating them with domain ontologies to provide Web service semantics (as shown in Example 3). WSDL-S approach helps not only to discover a right service but also the right operation among multiple operations supported by the same service. Instead of using XML schema [104] of complex input and output messages of Web service operations, these messages are mapped to domain ontological concepts to provide their shared meaning.

Example 3. WSDL-S extensions to WSDL message tag.

```
<wsdl:messagename="TranslatorRequest">
 <wsdl:part name="in0" type="tns1:inputLanguage"
       LSDISExt:onto-concept="LSDISOnt:SupportedLanguage"/>
</wsdl:message>
```

In addition with extending WSDL, the WSDL-S also adds new tags (i.e. *LSDISExt:precondition* and *LSDISExt:effect*) to WSDL specifications. These tags are used to describe pre-conditions and effects of a Web service operations. WSDL-S does not provide any mechanism to model the composition of SWSs but extends and uses BPEL for this purpose. It means, WSDL-S describes semantically, *what does the service provide* but not *how to use the service*. Figure 2.4 summarizes WSDL-S approach to semantically describe Web service capabilities. Also, WSDL-S concepts are being feeded to the upcoming SWS language (i.e. Semantic Annotation for WSDL (SAWSDL) [50]) 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.

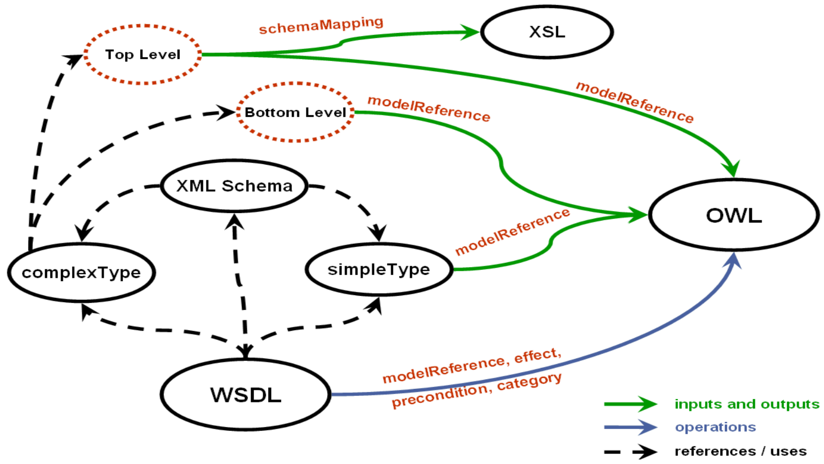

Figure 2.4: Overview of WSDL-S approach.

2.6.2 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. It provides a conceptual framework and formal language for describing Web services semantically to facilitate dynamic and automated discovery, invocation and composition of services. Web Service Modeling Language (WSML) [44] is formal language used to describe WSMO services. The Web Service Execution Environment (WSMX) [36] is execution environment for dynamic discovery, invocation and composition of WSMO services. Like other SWS languages the WSMO also uses domain ontologies (OWL ontologies) to provide domain specific semantics of Web services.

2.6.3 OWL-S

OWL-S is another language being developed to provide Web service semantics to facilitate dynamic and automated discovery, invocation and composition of Web services. OWL-S is suite of OWL ontologies (*Profile, Process Model* and *Grounding* ontologies) and each of these ontologies play a specific role to dynamically perform discovery, invocation and composition tasks. *Profile* ontology provides semantically enriched information about Web service capabilities which helps to publish and discover a service dynamically on the basis of matching semantics. *Process Model* ontology describes how to use a service and can be used for modeling semantic based composition of multiple 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. Like a workflow language, the *Process Model* ontology has very expressive capabilities to model the composition of multiple services but based on their semantic descriptions. Two major reasons for choosing OWL-S to semantically describe BPEL process models are 1) *Profile* ontology

Table 2.1: Comparison of SWS 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	Supported	Not Supported	Not Supported
Invocation	WSDL Grounding	WSDL Grounding	WSDL
Development Tool	Available	Available	Available

of OWL-S service can be used to provide semantically enriched meaning of a process as OWL-S service 2) *Process Model* ontology of OWL-S suite can be used to edit and model composition of multiple SWSs (like a workflow language). Table 2.1 describes a comparison of these SWSs languages.

2.7 AI Planning for Web Service Composition

Planning is about producing changes through actions [109]. The need for planning arises naturally when an agent is interested in controlling the evolution of its environment. Algorithmically, a planning problem has as input a set of possible courses of actions, a predictive model for the underlying dynamics, and a performance measure for evaluating the courses of action. While contributing to solve the problem of automatic Web services composition, AI community has provided different solutions for automatic Web services composition by using different AI planning techniques (e.g. Classical planning [45], HTN planning [107], GOLOG [67] etc.) (I shall describe and evaluate some of these approaches in Section 3.2).

Classical planning is useful in static environments, in which planner has complete information about the problem and the surrounding world. A planner focuses on two major issues: (1) modeling the actions and state change with actions (2) sequencing the actions towards the planning goal.

Hierarchical problem solving method reduces the planning complexities by focusing on one task at one time and ignoring others for the time being. A more sophisticated method of hierarchical planning is Hierarchical Task Network planning (HTN) [107]. In HTN planning a planning problem is organized into a set of tasks. A high level task in HTN plan can be reduced to sub tasks until a planner reaches to a primitive task that can be used to perform a single step operation by using the planning operator. The HTN planning already has a very closer match with OWL-S. For example, the OWL-S composite process can be mapped to HTN task that can be divided into sub-tasks and HTN operators refer to OWL-S atomic processes that can be performed in a single step.

In addition with developing new languages and tools for Web services and SWSs, new architectural approaches are also highly needed to shift existing legacy systems to upcoming semantic based environment. According to the IBM definition of web services architecture:

We believe that applications will be based on compositions of services discovered and marshaled dynamically at runtime (just-in-time integration of services). Service (application) integration becomes the innovation of the next generation of e-business, as business move more their existing IT applications to the Web, taking advantage of e-portals and e-marketplaces and leveraging new technologies, such as XML. The concept of Web services, described here, is our view of what the next generation of e-business architectures for the Web will look like [61].

In my research work I have also adopted a bottom-up approach to achieve business process automation. For this purpose I present a new 4-tier SWS integration architecture that addresses integration and composition issues of SWSs. Then I describe that how business processes can be enriched semantically by mapping them from a syntax based process modeling language (i.e. BPEL) to SWS language (i.e. OWL-S). The process models mapped to OWL-S composite services describe composition of services on the basis of matching semantics as well as semantically enriched interfaces of mapped OWL-S services can be used for dynamic discovery, invocation and composition to achieve the aim of business process automation as OWL-S services.

2.8 Summary

In this chapter I described state of the art in the area of Web service, semantic Web, SWS and process modeling languages. I highlighted Web service capabilities (e.g. interoperability, platform independence, loose coupling, standard languages etc.) which resulted in rapid adoption of Web service in academia and industry. Some process modeling languages (e.g. WSFL, MS XLANG and BPEL) that can be used to model business processes as syntax based compositions of Web services have also been discussed. This chapter described that current Web service architecture for publishing, discovering and composing Web services on the basis of their syntactical interfaces is not enough and new architectural approach is needed that addresses the syntactical limitations of Web service architecture (Chapter 3 describes a new architectural approach that addresses semantic based Web services composition and integration issues in detail).

Secondly, processes modeled by using traditional workflow modeling languages compose Web services on the basis of syntactical information of WSDL services. When these processes are exported as services they also expose syntactical interfaces which no more enable business processes to be dynamically discovered, invoked and composed by other semantic enabled systems. Syntactical limitations of process modeling languages (e.g. BPEL) is key factor that needs to be addressed to achieve the goal of business process automation as dynamic and automated composition of business processes as SWSs. For this purpose different SWS languages (e.g. WSDL-S, WSMO and OWL-S) and process modeling and semantic capabilities of these languages have been analyzed in this chapter. Analysis of capabilities and limitations of these SWS languages shows that SWS language (i.e. OWL-S) can be used to successfully address semantic limitations of process modeling language (i.e. BPEL). The *Process Model* ontology of OWL-S suite can be used to model the composition of Web services on the basis of matching semantics and *Profile* ontology of OWL-S suite can be used to expose semantically enriched interfaces of BPEL processes as OWL-S services. I have also discussed some AI planning techniques that can be used for automatic composition of OWL-S services. This chapter concludes that semantic enhancements in Web service architecture and efficient translation of business processes (BPEL processes) to OWL-S services can help in business process automation by enabling business processes for dynamic discovery, invocation and composition as OWL-S services.

Chapter 3

SWS Composition: Architecture and Framework

In this chapter I present a new 4-tier architecture for integration of business processes as SWSs compositions. The proposed 4-tier architecture is result of differentiation between architectural components (services) and those components that interact with services (orchestration). On the basis of 4-tier SWS integration and composition architecture an integration life cycle and a framework for SWS integration and composition has been presented. In this chapter I describe common architectural issues and problems that arise when we try to translate the composition of Web services from a syntax based to semantic based environment. I have adopted a bottom-up approach in this thesis therefore, before describing in detail about shifting of business processes to SWSs I describe the architectural and technical aspects needed for automation of business processes as SWSs composition.

3.1 Introduction

In a SOA, interaction between service providers and service consumers takes place in a loosely coupled way, where a service provider can also act as a service consumer. Web services and its related standards like WSDL, SOAP, UDDI and Web service composition languages (e.g BPEL) provide syntax based interaction and composition of Web services in a loosely coupled way. SOA activities and semantic enhancements in SOA (as discussed in Section 2.3) are needed for a common machine-to-machine communication between services with in and across the enterprise boundaries. Guidelines like using at least basic profile of WS-I standards, not publishing overloaded methods in Web services interfaces and creating WSDL first, then implement the service (contract first) are initial steps for reuse of services. But, guidelines in dynamic environment have to be monitored constantly in order to adopt them. Semantic enhancements in Web services, proposed by different research groups and composition of these services on the basis of matching semantics has different barriers to interoperability (e.g. compatible information models, interaction protocols etc.). Dynamically accessible semantic descriptions of service capabilities and utilization protocols, based on shared semantic models published on the semantic Web, are seen as a way to overcome these barriers, but they will require additional infrastructure so that individual software agents can directly interpret published service descriptions (which some times use unfamiliar ontologies) [34].

Dynamic composition of Web services is highly needed in the growing e-business world for the purpose of business process automation as semantically enriched Web services compositions. Composing Web services on the fly can efficiently affect the e-business world both at B2C and B2B levels. For example consider the simple scenario of a B2C interaction in which a client wants to order a pizza for delivery. In such a scenario user has some specifications (e.g. pizza ingredients, specific geographical location to deliver pizza, pizza rates etc.). To perform such a task a client has to manually discover and execute required services one-by-one, which is not an efficient approach. Similarly, B2B interactions in a distributive business environment involve prior agreements and predefined standards between interacting partners. Such prior agreements at different levels of integration can no more motivate efforts for business process automation.

Several current efforts (e.g. OWL-S, WSMO and WSDL-S) aim at providing Web service semantics. In Section 2.6, I have already compared these SWS languages and argued that OWL-S is most suitable SWS language that can be used to overcome syntactical limitations of process modeling language (i.e. BPEL) by translating BPEL process descriptions to OWL-S suite of ontologies so that these OWL-S services can be dynamically discovered and composed on the basis of matching semantics. Different solutions, like enhancing BPEL to create dynamic composition or using AI planning to automate the composition process of required services have been proposed. Most of the methods for business process automation as SWSs composition fall into one of the following two categories: methods based on pre-defined workflow model and methods based on AI planning. The first method uses workflow techniques and second approach is based on AI planning techniques. Both of these methods have their own composition approaches. The workflow method is more meaningful and useful in situations where problem model (e.g. BPEL process model) is already defined. In such a method dynamic composition involves discovery and binding of required services within Web services composition. On the other hand, AI planning method is more suitable in situations where requester has no process model but has a set of constraints and preferences. On the basis of this set of constraints and preferences, final composition can be generated automatically by the program [85]. In this chapter I provide a comparative study of some existing approaches for dynamic and automated composition of Web services and argue that due to lack of semantic support in Web service integration architectures and frameworks, these approaches result in limitations for dynamic and automated composition of SWSs.

As a solution to these problems, I propose a 4-tier SWS integration and composition architecture that provide a new architectural approach to address interoperability and compatibility issues from semantic based Web services composition perspective. On the basis of 4-tier architecture I also describe a SWS integration life cycle and a framework for dynamic and automated composition of business processes when they are translated to SWSs (e.g. OWL-S composite services).

The remaining chapter is organized as follows: Different SWS composition approaches have been discussed in Section 3.2. In Section 3.3 I highlight capabilities and limitations of these approaches. As a step to overcome limitations of existing SWS composition approaches a new 4-tier architecture for Web services integration in semantic service oriented paradigm has been presented in Section 3.4. On the basis 4-tier architecture I propose a SWS integration life cycle that has been discussed in Section 3.5. On top of these architectural concepts I present a novel framework for dynamic and automated composition of business processes as SWSs in Section 3.6. Section 3.7 provides a summary of this chapter.

3.2 SWS Composition Approaches

Dynamic and automated composition by means of Web service semantics is most important and promising task for SWS community to enable business process automation as SWS composition (as shown in Figure 3.1). Different approaches form both workflow and AI communities have been presented for the purpose of SWS composition and in this section I describe some of these existing approaches.

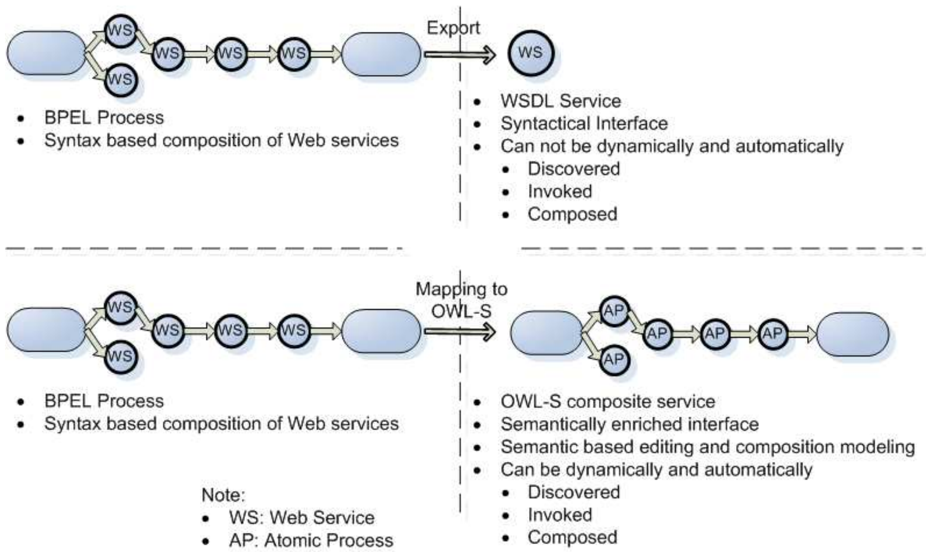

Figure 3.1: Enabling business processes for dynamic and automated discovery, invocation and composition by mapping them to OWL-S SWSs.

3.2.1 A Bottom-Up Approach

The work discussed in [70] presents a bottom-up approach by integrating the semantic Web technology into Web service technology while considering BPEL as composition of Web services. Idea behind this approach is to add semantics in BPEL that provide machine understandable descriptions of required services with in process and extending workflow execution engine (BPWS4J) [43] to realize these semantic descriptions. With these semantic descriptions the bottom-up approach uses Semantic Discovery Service (SDS) to dynamically discover a required service on the basis of matching semantics and bind it with in composition. This approach makes use of DAML Query Language (DQL) to query these repositories of DAML-S Profiles. JTP (Java Theorem Prover) (DAML-S reasoner) is used to find matching service profiles and to compose these services dynamically. In case, if a single service does not meet a service requirements, the SDS uses a recursive back-chaining algorithm to determine a sequence of service invocations or service chain, which takes input provided by the BPWS4J and returns output required by the BPWS4J. However, the system efficiency goes down as the number of service Profiles increases in service chain. One major limitation of this approach is that it doesn't consider pre and post conditions for discovery and composition purposes. Also, chaining multiple services to get required output, in case of long chaining process, affects the efficiency of proposed approach.

3.2.2 METEOR-S Approach

In the METEOR-S project [14], the working group has developed a tool for dynamic composition of Web services. The METEOR-S tool (METEOR-S process designer) allows process designers to design processes on the basis of business and process constraints. Idea behind Web Services Composition Tool is to write required service specifications as an abstract process within BPEL process model and to discover services whose *Profile* matches to defined abstract process. Once required services are discovered, candidate service is selected on the basis of process and business constraints. The process designer uses BPEL for process modeling. A service template is created by using functional as well as QoS specifications of all operations of a Web service in a process [14]. Major drawback of this approach is that end user has to manually select a service for composition among bundle of dynamically discovered matching services.

3.2.3 Template Based Composition

In the work discussed in [100], Evren Sirin uses workflow templates to write abstract activities. These abstract activities can be used to describe required services. On the basis of these activities specifications required services can be discovered to create executable workflows. This approach focuses on value of adding preferences in templates so that services can be ranked to find most suitable one among a bundle of discovered services. Evren Sirin proposes the use of semantic Web technology (OWL) for writing such templates, which allow reasoning for flexible and more consistent match making of required services. This approach focuses on extending the OWL-S process ontology by proposing the addition of abstract process. Evren Sirin proposed that process ontology should have an abstract process that can be used to refer to the *Profile* ontology of an OWL-S service with other specifications that can be used to rank and find best suitable service. The proposed abstract process, unlike to *atomic* process is not connected to specific *Profile* or *Grounding* and unlike to *simple* processes is not connected to any existing process. This approach implements use of AI planning approach (i.e. Hierarchical Task Network

(HTN) planning) with its extended formalism as HTN-Description Logic (HTN-DL) for automatic Web services composition.

3.2.4 Semi-automatic Composition

A semi-automatic composition approach and a prototype SWSs composition tool have been discussed in [98]. The tool discovers semantically matching services from available services repository. These discovered services are then filtered and presented at each step of Web services composition process. End user selects a required service among these available services for the purpose of composition. The service composition tool consists of two components (i.e. inference engine and a composer). Inference engine stores information about all available services in its knowledge base and is capable of finding matching services. Composer is user interface that handles communication between human operator and inference engine. The inference engine discovers matching services on the basis of matching semantics and filters most suitable services on the basis of functional and non-functional attributes. The composition tool doesn't allow the end user to define control flow between discovered services. Also users are not able to define condition statements to have some conditional results of a *composite* process.

3.2.5 WSMO Composition Approach

WSMO community has also developed a tool [86] for dynamic composition of Web services and has integrated it with IRS-III [46]. IRS-III is a framework to develop, publish, compose and execute SWSs developed on WSMO specifications. The composition tool allows users to select goals, mediators and control flow operators to define control flow between components. As already discussed that the composition tool is integrated in IRS-III server therefore composition process starts by selecting a composition goal from the list of available goals defined in the IRS-III server. Data flow between these goals can be defined by specifying the data source as input of goal and the data destination as an output of the goal. Also the tool allows to define values of inputs and outputs of goals at design time of composition. Type mismatch between inputs and outputs of goals can be managed by using mediators. Mediators map and perform transformation between goals. Defining XSL Transformations can support such a data mapping between messages of different types in OWL-S. WSMO composition tool supports semi-automatic composition of Web services by helping users during the process of designing the composition. On the basis of abstractly defined service requirements, services are discovered and invoked dynamically. Non-functional expectations on a service composition are expressions of human will, and consequently should be given by the users instead of letting composition engines to guess or randomly assign the right service [86]. The WSMO community aims at improving the semi-automatic composition process by supporting the discovery and composition on the basis of non-functional semantics.

3.2.6 Some Other Composition Approaches

Planning for Web services Composition by Using SHOP2. The work discussed in [101] describes how an AI planning system (i.e. SHOP2 [3]) can be used with

DAML-S (OWL-S) Web service descriptions to automatically compose Web services. This approach gives partial support for composing services on the basis of their matching functional and non-functional semantics. [101] Does not support the creation of a *composite* process with all OWL-S supported control constructs (e.g. this approach does not support synchronization between process components by implementing the use of OWL-S Split-Join control construct).

SWORD. The method reported in [83] provides a set of tools for composition of a class of Web services. The SWORD implements use of rule-based expert system that determines possibility of automatic creation of composite service from existing services. In case of such possibility a plan is created. Execution of such a plan generates composite service. This approach is limited with respect to selecting Web services for composition just on the basis of input and output and does not handle services that have certain pre-conditions or effects.

Plængine. Plængine [75] is a software system that supports planning for service composition and service enactment. The Plængine uses integrated meta-model approach to plan for Web services composition. The Plængine consists of two components: a composer and an enactor. The composer is responsible to generate composition with the help of its subcomponent ComposerThread that uses search-planning algorithm to perform composition. The enactor is responsible for scheduling and execution of individual services within a composition. This work focuses on overcoming limitations (e.g. handling exceptions, sophisticated support for control flows and extending architecture of meta-models).

Web Services Composition and Execution Framework. The framework discussed in [53] provides mechanism and tools for visual orchestration of semantically well-defined building blocks and semantic invocation of services that match to the user specifications. The dynamic composition approach presented in this work uses predefined flow of complex service extended with abstract functional building blocks. These abstract building blocks define requirements for a service to perform a specific task. The best matching service is discovered and invoked at execution time. A part of this work has been discussed in [79] which describes how to handle BPEL limitations of static Web service binding with late binding by using the idea of "generic Web service proxies". This work presents idea of service ontology based semi-automatically generated activity components, which can be used and manipulated by tools (e.g. for visual modelling of complex services, in deployment phase, in execution phase etc.). Framework proposed in [53] does not fully support dynamic composition on the basis of both functional and non-functional service semantics, which reduces efficiency of proposed framework.

3.3 Limitations of Current Composition Approaches

In this section I highlight major issues that need to be addressed to automate interaction between business processes by translating them to SWSs and composing these services dynamically on the basis of matching semantics. I also summarize above discussed composition approaches by compiling them with respect to their level of support for these composition aspects. Some major issues that help in evaluating existing approaches are as under:

Service Discovery and Selection on the Basis of matching Semantics. This issue ddresses the problem of discovering a service on the basis of matching functional semantics (e.g. input, output, pre and post-conditions) and non-functional semantics (e.g. service response time, geographical location etc.). It is also concerned with selection of a single service from the bundle of semantically discovered services.

Service Binding & Referencing. In case of a workflow language as Web services composition, Service Binding & Referencing describes that how a selected service is bound in final composition and in case of an AI planning approach how a service is referred in final composition generated by an AI plan.

Composition Strategy. This issue addresses the approach used for composition of semantically enriched services. For example, in case of a workflow language as Web services composition, composition strategy describes that either the composition is dynamic or not. In case of AI planning composition strategy describes that either the final composition is generated automatically (automatic) or semi-automatically (semi-automatic).

Execution. Focuses on execution support in the proposed work for the execution of final composition.

Semantic Web Technology. Describes the approach used to add semantics to Web service technology and SWS language used to write semantic based Web service request (e.g. OWL-S, WSDL-S or WSMO etc.).

Table 3.1 summarizes capabilities and limitations of above discussed approaches with respect to these (as discussed above) SWS composition issues. Table 3.1 shows that none of the above discussed approaches addresses all these composition issues. For example, in bottom-up approach (discussed in Section 3.2.1) QoS semantics, pre and post conditions of services play no role in discovery and composition mechanism. In this approach process designer handles pre and post conditions at design time. Similarly, the approach discussed in Section 3.2.2 also defines the basic workflow in BPEL and dynamically discovered services are binded in final process at design time. Semi-automatic composition approach discussed in Section 3.2.4 involves human interaction at each step of service composition. But this approach discovers and filters available services on the basis of matching functional and non-functional semantics.

To enable collaboration between business processes (e.g. BPEL processes) in a dynamic and automated fashion by translating them to SWSs (e.g. OWL-S services), SWS integration and composition approach should provide solution for above discussed issues. With such an approach we can avoid different composition problems, for example, selecting and composing required services dynamically and at run time on the basis of both matching functional and non-functional semantics to avoid problems that occur when

Table 3.1: Comparison of some existing dynamic and automated Web service composition approaches.

Composition Approach	Service Discovery		Service Selection		Service Binding & Referencing	Composition Strategy	Execution	SWS Language
	Functional Semantics	Non-Functional Semantics	Functional Semantics	Non-Functional Semantics				
Bottom-Up Approach	Partial	No	Partial	Yes	Run-time	Dynamic	Yes	OWL-S
METEOR-S	Yes	Partial	Yes	Partial	Deployment /Design time	Dynamic	Yes	WSDL-S
Template Based Composition	Partial	Partial	Partial	Partial	Dynamic	Automatic	Yes	OWL-S
OWL-S Composition Approach	Yes	Partial	Yes	Partial	Dynamic	Semi-automatic	Yes	OWL-S
WSMO Approach	Yes	Partial	Yes	Partial	Dynamic	Semi-Automatic	Yes	WSMO
HTN Planning Using SHOP2	Partial	Partial	Partial	Partial	Dynamic	Automatic	Yes	OWL-S
SWORD	Partial	No	Partial	No	Off-line/ Composition time	Semi-automatic	Yes	Independent of Standards
Plængine	Yes	No	Yes	No	Dynamic	Automatic	Yes	Integrated Meta-Model
Ontology Derived Activity Components Approach	Partial	Partial	Partial	Partial	Dynamic	Dynamic	Yes	OWL-S

a single service within composition is not accessible, or when its functional and non-functional semantics no longer match to the required service semantics. An architectural approach which addresses issues like orchestrating, querying for required services and composing them on the basis of matching semantics can be a positive step towards solution. In next section (Section 3.4), I propose such an architectural approach that uses semantically enriched interfacing between different layers of SWS integration architecture.

3.4 4-Tier Integration Architecture

As the Web becomes more semantic and applications become more agile need for an additional architectural layer becomes more prevalent. This new architectural layer choreographs the business rules and orchestrates services by using ontologies. Figure 3.2 outlines how the choreography and orchestration layer(CO-layer), and services layer in the "new" 4-tier architecture evolved from the business logic layer of the current 3-tier application integration architecture. This new architectural layer is derived from the natural evolution of the business logic layer. The four layers in the proposed 4-tier integration architecture cooperate in order to provide overall functionality. The invocation relationship between four layers is strict top-down invocation relationship. That is, components in upper layers invoke components in lower layers in order to accomplish their functionality and lower layers cannot invoke components in upper layers. This avoids circular invocation dependencies and ensures that the functionality separation is followed [35].

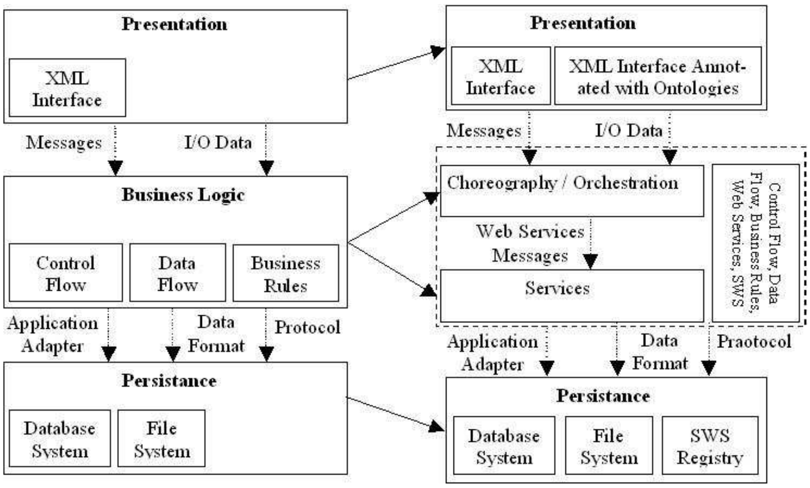

Figure 3.2: 4-tier semantic Web services integration architecture.

The 4-tier SWS integration architecture consists of the following four layers (as shown in Figure 3.2):

- Presentation Layer
- Choreography and Orchestration Layer (Business Logic Layer)

- Services Layer (Business Logic Layer)
- Persistence Layer

Presentation layer provides interface to interact with integrated applications. Different interfaces can be provided to meet different integration requirements. For example, XML provides a cross-platform standard for creating interfaces. It enables better reuse of user interfaces - presentation layer - in complex integration scenarios. Also, it can bypass some WSDL complexities when annotated with domain ontologies to provide data semantics. Different XML messages and input/output data creates the interface between presentation layer and business logic layer. Execution of business logic (process) is closely dependent on these messages and data bridging the co-ordination between presentation layer and business logic layer (CO-layer and services layer). The resulted integrated application (service) can also present its interface (XML interface) for further co-operation with other services and applications. This interface can also be annotated with ontologies. Such a semantic interface helps in further dynamic and automated discovery, composition and invocation of these integrated services by semantic enabled systems.

Business logic layer contains the components that implement the integration functionality. In traditional business application integration scenarios, business logic layer define the control and data flow between integrated applications and implement the business rules and business logic. Components in business logic layer interact with preceding layer components by using some application adapters, data transport protocols and or formats (as shown in Figure 3.2). As long as business applications development trend has changed to business service development (extended with domain specific semantics) and the business logic layer comes in to focus, we begin to differentiate between architectural components that provide services and those components that either orchestrate services (service composition on the basis of matching semantics and aggregation) or choreograph them (business rules and workflow). This difference between components that realize specific use-cases (services) and components that organize those use-cases into dynamic business rules (choreography and orchestration) is emphasized by splitting the business logic layer in two. These two layers jointly play the same role as business logic layer (i.e. event management, process management, data management) and managing the control and data flow between services. When talking about integration architecture for integration and composition of services which expose semantically enriched interfaces, I describe the role of CO-layer and services layer individually in the whole SWS integration and composition architecture.

The CO-layer choreographs and orchestrates services in services layer with business rules and semantics. This is agile layer of the 4-tier software architecture model. This layer is required to be dynamic - to meet the changing requirements of the business enterprise. CO-layer also needs to be adaptable as the enterprise grows through merger and acquisitions. Business logic and business rules can be implemented here by separating them from the underlying infrastructure of the system's operation. These rules can be implemented in some structure language. Even though CO-layer and services layer are emerged from business logic layer but components in these layers coordinate in such a way that they are invoked precisely and in the right order by sending and receiving messages between CO-layer and service layer components. Chapter 4 describes it in detail that when BPEL process is mapped to OWL-S service individual processes with in a composite process are invoked in right order defined with in mapped composite process by sending and receiving semantically enriched messages which are annotated with domain ontologies.

Services layer is not new, for many it remains equivalent to the business logic layer and contains a business rule service. The services layer is the realization of business processes in terms of discrete service definitions. This layer is inherently static as services are tightly coupled to their implementations. However, when consistently defined in terms of IOPE with domain ontologies, services begin to reveal patterns of behavior that can be modeled and orchestrated. Enhancing the services layer with semantic support no more keep it static, as, required services can be defined here semantically and can be discovered and composed dynamically on the basis of matching functional and not-functional semantics (e.g. as discussed in Sections 3.2.2 and 3.2.3).

As discussed above that the proposed integration architecture is a top down approach in which components in upper layer can invoke components in lower layer therefore, business logic layer can be interfaced with persistence layer by using some application adapters, protocols and data transport formats to exchange messages. Splitting the business logic layer in to CO-layer and services layer results in an additional interface to query for SWSs and getting its response. Reliability of the application integration architecture is intimately dependent on persistence components (persistence layer). In the whole integration architecture, database systems are used to store and to manage data. The data includes the messages, events, processes and configuration data. One possibility is to store all the information in some database but file systems can also be used to store data in files as part of persistence layer. The newly emerged layer (i.e. services layer) in the integration architecture added an additional component to persistence layer (i.e. SWS registry). The CO-layer, services layer and persistence layer co-ordinate by sending query for a required SWS (SWS request) and getting its response (semantic Web service response). The 4-tier SWSs integration architecture supports integration of semantically enriched Web services but it is not enough for dynamic, semi-automatic and automatic annotation, advertisement, discovery, selection, composition and execution of inter-organization business logic, making the Internet become a global common platform where organizations and individuals communicate among each other to carry out various commercial activities and to provide value-added services [37]. For these purposes SWS integration and composition life cycle is presented in next section.

3.5 Integration and Composition Life Cycle

SWS integration and composition life cycle (as shown in Figure 3.3) describes an engineering and development cycle to fully harness and sharpen the power of business processes as SWSs. The proposed life cycle is based on a top down approach starting from modeling business processes (composite services) as Web services compositions and ending with their execution. It consist of multiple modules including developing business processes, adding technical and business constraints to processes, annotating the composition workflow with domain ontologies to prepare semantic based service requests in the workflow and deploying and executing the final process (composite service). Each phase of the SWS integration life cycle is responsible to perform a specific task. I herein describe functional aspects of these phases individually.

Business Process Modeling. Business departments define how a single process steps are combined with each other and control and data flow between these process steps - business logic can be defined. In fact, they do not know technical aspects and implementation of these processes and how Web services work, but they are able to design and model the business logic. Different methods like Value Chain Diagrams [57], Event Driven Process Chains [74, 52] and UML Activity Diagrams [52] can be used to model business processes. These methodologies are more useful for business experts to describe business logic as business processes that are annotated with management requirements. Deliverables of such business analysis and design processes are not readable for computers. They need some technical descriptions (e.g. BPEL process or OWL-S *Process Model* descriptions) to become readable and executable by machines.

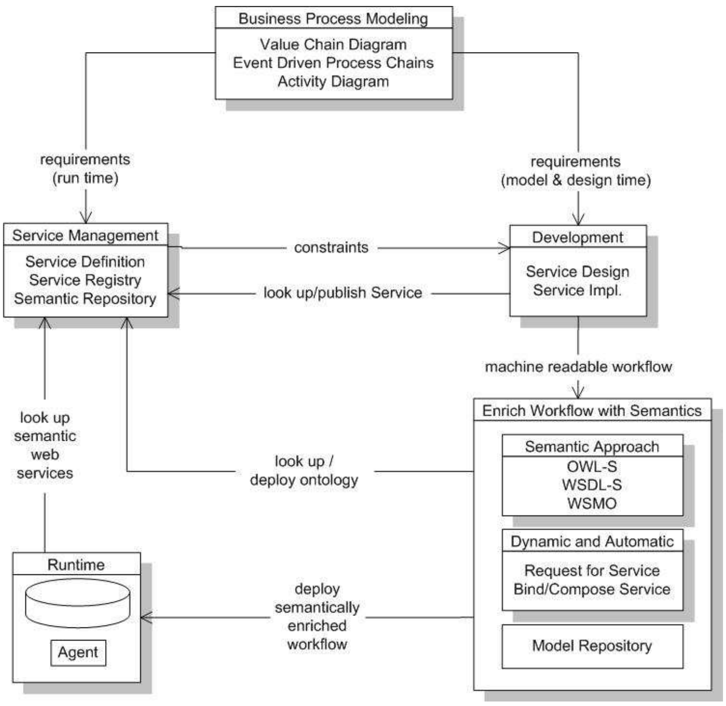

Figure 3.3: SWS integration and composition life cycle.

Development. Once defined, business processes are developed as a composition of Web services with their technical implementation. Technical descriptions of business processes make them machine readable for the purpose of deployment and execution. Machine-readable descriptions of processes can also refer to some existing services available in service registry to perform a specific part of the total business goal. Business constraints (e.g. business rules, data exchange format, communication protocols etc.) are applied to the business process to meet management aspects of integration process. Even though technical descriptions of processes have been implemented to make them executable for machines at this phase, but implementation of semantic descriptions of required services is still needed for the purpose of dynamic discovery and composition.

Semantics Enrichment of Workflow. Instead of binding required services within the composition at design-time (development phase), required services can be described in processes semantically. These semantic service requests can be annotated with domain ontologies. Domain ontologies are managed in the service management scope. The final business process is a process defined in some workflow language (e.g. OWL-S *composite* process or BPEL enriched with process semantics). The process of preparing and sending a request for SWS, discovering a service on the basis of matching semantics and getting its response is dependent on semantic enhancements in participants (service provider, requester and registry) of SOA.

Runtime Phase. Semantically enriched workflows can be deployed on semantic enabled execution engine (i.e. execution engine capable of understanding workflow semantics). Execution engine is capable of invoking Web services that are statically bond in the process during the development phase of life cycle. Also, services defined semantically in the workflow are searched in the semantic services registries. Services discovered on the basis of matching semantics are bound in the workflow at run time. Discovering a service just on the basis of matching functional semantics (input, output) may not always acquire right services, therefore, a semantic service request with in process is defined on the basis of both functional and non-functional semantics. At the end, final process as a composition of services is executed with defined control flow and data flow.

Service Management. As described before, the service management phase is the always-on and helping phase within the life cycle. Managers and developers can manage service publishing and serving requests for semantic and syntax based Web services. Web services registries are enhanced to SWSs registries for publishing and querying for semantically enriched services. Domain ontologies are also managed in this phase. These domain ontologies can be used to annotate Web services and business processes to provide data semantics. Business processes can be managed for the deployment and execution in this phase as well. Service management phase helps to provide business constraints for modeling business and technical perspectives of a Web service integration scenario. The Web Service Description Language (WSDL) does not support the specification of various constraints, management statements, classes of service, Service Level Agreement (SLAs) and other contracts and protocols between Web services.

Traditional business integration scenarios follow the 3-tier business applications integration architecture. Web services and semantic enhancements in Web services resulted in the improvement of 3-tier architecture to 4-tier architecture. The additional layer is responsible for integrating business applications as SWSs and the above-discussed life cycle addresses issues emerged in traditional SWSs integration scenarios (e.g. management of domain ontologies, deploying and querying SWSs, composing and deploying these services on semantic enabled execution engines etc.). In next section (Section 3.6) I describe a framework for dynamic and automated composition of business processes as SWSs.

3.6 SWS Composition Framework

In this section I describe a general framework at an abstract level for dynamic and automated integration and composition of business processes as SWSs. On the basis of above discussed challenges and limitations of recent approaches I propose a composition framework, which consists of four modules (as shown in Figure 3.4). Each of these modules is responsible to perform a specific task that, in combination with other modules results in a composition framework to generate semantic enabled composite services. Here I describe each of these modules in detail and discuss which specific composition problem is addressed by each module.

Semantic Service Requester. The first step to perform dynamic Web services composition is to discover and select required services on the basis of matching semantics. This dynamic discovery and selection is a run time process. Because semantic based discovery and selection of required services at design time also involves human interaction, which no more automates the process of Web services composition. The semantic service requester involves interaction with:

- SWS registries (e.g. as discussed in [108]) that are capable of publishing semantically enriched descriptions of Web services and for replying for SWS requests.
- it can also request and interact directly with a known Web service which can accomplish required objectives.

The discovery and selection process of required services is based both on matching functional and non-functional semantics of a Web service. For example, in case of a pizza delivery process, a user sitting in New York requests for a vegetable and mutton pizza. In case of such a request, there would be multiple services that offer vegetable and mutton pizza delivery. But in this case, a service with non-functional matching semantic (e.g. suitable geographical location for a pizza request) is selected for composition. At this stage it is assumed that suitable work has already been done to publish SWSs on semantically enriched registries that have capabilities to reply for SWS queries (as discussed in 4-tier SWS integration architecture (persistence layer)). In the proposed framework, module 1 (i.e. semantic service requester) is responsible to perform such a semantic base service request and to select a service for composition, which has closer semantic match to service request.

Service Binder Like the work supported in bottom-up approach (as discussed in Section 3.2.1), this module is responsible to bind a dynamically discovered and selected service within composition. Runtime binding of required services can help to meet challenges produced by services which change on the fly or which become inaccessible on the network with the passage of time. For example in case of composition of services as a workflow each partner service is bound in workflow at run time so that only those services become part of composition which are currently accessible and meet service functional and non-functional requirements. Similarly, in case of AI planning approach a single service performing some action in a single step (*atomic* process) becomes a part of final composition (complex service) generated by a composition plan. Module 2 of the proposed framework is responsible for run-time binding and referencing of a service within Web services composition

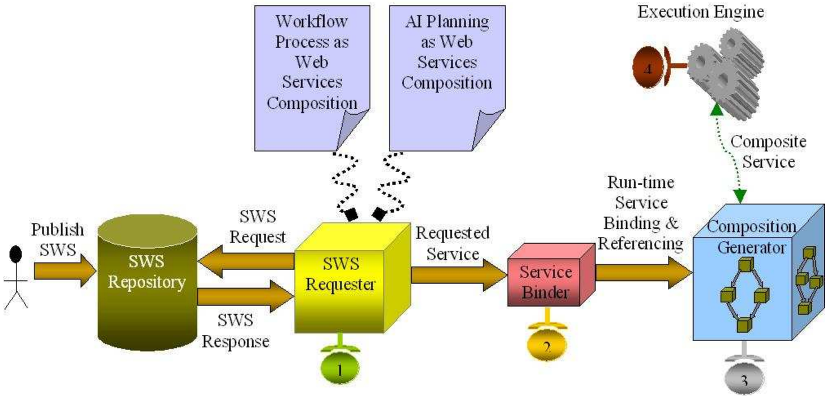

Figure 3.4: Architecture of dynamic and automated Web service composition framework.

Composition Generator. This module (i.e. module 3) is responsible for generating the final composition of SWSs, discovered and bound within composition at run time. In case of a workflow language as a composition of these dynamically discovered and bound services, this module is responsible for generating the final composition process in some workflow language (e.g. BPEL process model or OWL-S *Process Model* ontology as composition of semantically enriched services). In case of an AI planning for automatic Web services composition this module generates the final composition as a complex service (composite service). Composition Generator composes these services with well-defined control flow and data flow within composition. Different approaches have been discussed [86, 101] to automatically compose SWSs defined by using OWL-S descriptions. The automatic composition of OWL-S services can result in an OWL-S composite service. Since, WSDL-S does not support to define composite services therefore, no such approach has been discussed in Section 3.2 that automatically compose Web services by using the WSDL-S service descriptions.

Execution Engine. Finally composition of dynamically and automatically composed services is executed at this stage (module 4). Each service involved with in the composition is executed according to defined control flow. Data flow definition helps to pass data between services with in composition. For example, the approach discussed in Section 3.2.1 uses Semantic Discovery Service (SDS) between process engine (BPWS4J) and Web services to execute the final process with dynamically composed services. Similarly, the approach discussed in Section 3.2.3 uses its execution engine to execute the resulting OWL-S *composite* process.

3.7 Summary

In this chapter I described a comparative study of recent approaches for semantic based discovery and composition of Web services. I highlighted dynamic composition issues and analyzed existing composition approaches with respect to these issues. This comparison resulted in requirements for new architectural approach that can be used to efficiently integrate semantically enriched services. To meet these architectural requirements I have presented a 4-tier architecture that can be used to integrate and compose business processes as SWSs. The 4-tier architecture explains necessary concepts to orchestrate, query and integrate (compose) required services dynamically on the basis of matching semantics.

A SWS integration and composition life cycle has also been discussed in this chapter. The proposed SWS integration life cycle addresses discovery and integration issues and attempts to bring these efforts together. The life cycle starts with adding semantics to Web services and defining business goals that need to be achieved by dynamically discovering and composing SWSs. This chapter described how business processes could be annotated with business logic, rules and constraints in some machine-readable workflow language (e.g. *Process Model* ontology of OWL-S suite) (Chapter 4 describes the constraints that can be used to translate business processes to OWL-S SWSs). I also described the annotation of business processes with domain ontologies to expose their semantically enriched interfaces (e.g. as *Profile* ontology of OWL-S suite). Such semantically enriched workflows (semantically enriched composite services) can be deployed and executed by an execution engine that is capable of understanding process semantics. The success of the proposed Web service integration life cycle and Web services composition framework will be ultimately dependent on the acceptance of emerging standards for SWS. In next chapters of this thesis I propose a strategy and its prototypical implementation that can be used to shift existing business processes (BPEL processes, which are syntax based composition of Web services and expose syntactical interfaces) to OWL-S services (which are semantic based composition of Web services and expose semantically enriched interfaces) for business process automation as dynamic and automated composition of SWSs by utilizing above discussed architectural and structural concepts.

Chapter 4

Mapping Constraints

In this chapter I describe mapping constraints as functional relations between BPEL process models and OWL-S suite of ontologies by analyzing BPEL and OWL-S. Mapping constraints have been defined on the basis of different behaviors that BPEL and OWL-S components show in different situations and relation between these components on the basis of their matching behaviors. These mapping constraints create the base to define mapping specifications that can be used to map BPEL process descriptions to OWL-S suite of ontologies (as discussed in Chapter 5).

4.1 Introduction

Major drawbacks of traditional business process modeling languages are 1) they compose Web services on the basis of their syntactical information 2) when these processes are exposed as Web services they have same syntactical limitations as traditional WSDL services. Consequently, modeling Web services compositions and discovering, invoking and composing them on the basis of syntactical information is inefficient and (due to many single points of failure) unreliable. Different research groups in the semantic Web and SWS community are working on developing standard languages (e.g. OWL-S, WSDL-S and WSMO) to equip Web service with semantics. SWS community has also presented different approaches to dynamically discover, invoke and compose these services on the basis of matching semantics. Due to dynamic and automated behavior of SWSs they are getting more and more attraction of large business organizations. At this stage an approach is needed that can be used to shift existing business processes (e.g. BPEL processes) to SWSs (e.g. OWL-S services) in an efficient and cost affective way rather than to build semantic based business services from scratch. Such an approach will help not only to model the compositions of services on the basis of matching semantics but also to expose semantically enriched interfaces of business processes (BPEL processes) as OWL-S composite services. These semantically enriched interfaces can be used for dynamic discovery, invocation and composition of processes as SWSs.

In this chapter I depict mapping constraints that describe correspondence between business processes (i.e. BPEL processes) and SWSs (i.e. OWL-S services). These mapping constraints can be used to translate BPEL processes to OWL-S services. Mapping constraints have been discussed by providing detailed analysis of BPEL process model and its components and OWL-S suite of ontologies and its control constructs and defining relation between BPEL and OWL-S components on the basis of their matching functional characteristics.

A BPEL process model consists of different activities that can be used to interact with other services, create interface of process and to define control and data flow between services. Similarly OWL-S composite service consists of three ontologies (i.e. *Profile, Process Model and Grounding* ontologies). Analysis of BPEL process model, OWL-S suite of ontologies and their components helps to categorize and to specify that which part of a process should be mapped to which construct of OWL-S on the basis of their matching functional characteristic. Since, OWL-S suite consists of *Profile, Process Model and Grounding* ontologies, therefore mapping constraints are clearly discussed for mapping of BPEL process components to components of these ontologies.

The remaining chapter is organized as follows: Section 4.2 provides an analytical description of BPEL process model and functional characteristics of its components. Section 4.3 describes an analytical view of OWL-S suite and its constructs and describes matching functional aspects of BPEL process components and OWL-S constructs. Section 4.4 summarizes this chapter.

4.2 BPEL4WS Process Model Analysis

BPEL specifications allow to create complex business processes by creating and defining control and data flow between different activities that can be used to perform Web service invocation, passing data between different activities, handling faults and to terminate a process. We can model a process by nesting these activities within *structured* activities to define how to execute them (i.e. either to execute them in *sequence* or *parallel* or depending on some *condition*).

As first part of mapping constraints I analyze functional characteristics of a BPEL process model and its components. During the process of mapping BPEL process to OWL-S service, I create object view of BPEL process and WSDL services involved with in a process. Creating such an object view helps to map these activities of a process to constructs of OWL-S service on the basis of their matching functionality. Here I do not mean to provide detail description of these components of a process (because BPEL specifications cover them in more detail) but to analyze functional characteristics of these activities to create base of their mapping.

4.2.1 Processes

BPEL allows to describe business processes in two ways:

Executable Processes are used to model interaction between participants (Web services) of a business process. The logic and state of the process determine the nature and sequence of Web services interactions conducted by each business partner, and thus the interaction protocol [42].

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 [62]. One thing to note is that executable processes are permitted to use the full power of data selection and assignment but are not permitted to use nondeterministic values. Abstract processes are restricted to limited manipulation of values to reflect the consequences of hidden private behavior [42].

4.2.2 Partner Link

A business process interacts with services that are part of business process by using partner links ($< partnerLinks >$). More than one partner links are characterized by using partner link types ($< partnerLinkType >$) to define relation between two services.

```
<partnerLinks>
 <partnerLink name="To_Translation_Service_Port_1"
  partnerLinkType="q1:To_Translation_Service_Port_1Type"
  partnerRole="portRole"/>
  ..........
<partnerLinks>
```

4.2.3 Primitive Activities

A BPEL process is a set of activities (*primitive* and *structured* activities). *Primitive* activities are used to perform basic tasks of a process. For example:

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

Example 4. *invoke* activity which performs Web service operation *(getMeaning).*

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

When we talk about mapping constraints, *invoke* activity has dual behavior. Its one behavior is to perform a Web service operation by sending it some input message and probably receiving some output message. Its second behavior is that it can be used to create the interface of an asynchronous BPEL process (i.e. to send an output message of a BPEL process to the outer world). In both cases mapping of *invoke* activity to OWL-S varies (Section 5.2 discusses it in detail). *invoke* activity is used to perform both synchronous as well as asynchronous operations. In case of a synchronous (request/reply) operation, the process waits till it get reply from Web service operation. Asynchronous Web service operations are invoked by using *invoke* activity and process continuo to perform other activities. Response of such Web service operation is received by using *receive* activity.

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. For example it can act as an interface of a BPEL process (i.e. to receive a message from outer world (probably from another service) to start a process). Its second behavior is that it can be used to receive a message from a Web service in response to an asynchronous Web service operation (as discussed above). *Discussion about possible behaviors of these activities is important because a single BPEL activity is mapped to different OWL-S control constructs (in remaining chapters of this thesis I write the term control construct as "CC" and control constructs as "CCs") just depending on their behavior.*

Example 5. *receive* activity used to initiate our example process.

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

Reply (< *reply* >) activity is used to reply a message in response to some *receive* activity. A BPEL process (synchronous or asynchronous) receives an initial message by using a *receive* activity to start a process. However, synchronous BPEL process returns results of process by using *reply* activity and an asynchronous process returns the result of a process by using *invoke* activity.

```
<reply partnerLink="Input_Output_Port"
 portType="q1:Input_Output_PortType"
 operation="Operation_1"
 variable="Message_2_From_Translation_Ser"/>
```

Assignment (< *assign* >) activity is used to assign values to message variables. In a BPEL process the Assignment activity can be used to initialize an input message of a Web service by assigning it values of message parts of output message of another Web service operation (i.e. assigning and passing output of one Web service operation as input of the next Web service).

```
<assign>
 <copy>
  <from variable="Input_Message" part="part"/>
  <to variable="Message1_To_Translation_Service"
               part="inputLang"/>
 </copy>
<assign>
```

Primitive activities are used to perform small tasks with in a complex process and can be combined by using some *structured* activities to exactly specify steps of a business process.

4.2.4 Structured Activities

BPEL *structured* activities are used to define control flow between sub *primitive* and *structured* activities with in a process. BPEL provides a number of *structured* activities in which each activity has its own control flow characteristics. Some major structured activities with their functional behavior are described below.

Sequence (< *sequence* >) activity is used to define a set of activities which are performed in a sequence. The code sample (given below), shows that sub activities (i.e. *receive* and *reply*) of main activity (i.e. *sequence*) will be performed in a sequence.

```
<sequence>
 <receive partnerLink="Input_Output_Port"...../>
  ..........
 <reply partnerLink="Input_Output_Port"...../>
</sequence>
```

Flow (< *flow* >) activity invokes child activities in parallel. The following sample shows that child activities (i.e *invoke* activities) are performed in parallel.

```
<flow>
 <invoke partnerLink="Dictionary_Ser_Port"...../>
 <invoke partnerLink="Dictionary_Ser_Port"...../>
</flow>
```

Case-switch (< *switch* >) activity is used to perform child activities under some conditional aspects. The sample *switch* activity (given below) shows that sub activity (*sequence*) will be executed if the message part (i.e. *inputLang*) of the message *Input_Message* is equal to *English*.

```
<switch>
 <case condition="bpel:getVariableData('Input_Message',
                  'part', 'inputLang')= 'English'">
  <sequence>
   ..........
  </sequence>
 </case>
</switch>
```

While (< *while* >) activity performs child activity as long as the *while* condition holds true. The sample code given below shows that sub activity (i.e. *sequence* activity) of *while* activity will be performed as long as value of variable *status* remains "-1".

```
<while name="CREATION_WHILE" condition="bpws:getVariableData(
                  'status', 'status', '//type')=-1>
 <sequence
  ..........
 </sequence>
</while>
```

4.2.5 Some Additional Activities

Wait ($< wait >$) 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.

Analysis of BPEL process model and its components helps to understand functional characteristics of BPEL activities. On the basis of these characteristics of BPEL activities I summarize mapping specifications (as discussed in Chapter 5) that can be used to translate existing BPEL processes to OWL-S SWSs.

In this section I have provided a short description of BPEL processes and functional constraints of BPEL activities. I have also discussed about different roles that a single activity can play in a process. Some syntactical information about these activities has also been provided. This syntactical information is very important from point of implementation of our mapping approach. I have also discussed logical behavior of these activates so that it become easier to specify that which BPEL activities have matching behavior to which OWL-S CCs so that they can be mapped to their relevant CCs.

4.3 OWL-S Ontology Analysis

OWL-S is suite of OWL ontologies developed to describe semantic Web services and consists of *Profile*, *Process Model* and *Grounding* ontologies. Here, I highlight logical aspects of OWL-S ontology to justify that how OWL-S can be used to address syntactical limitations of a BPEL process. I also analyse functional characteristics of OWL-S CCs so that BPEL process and OWL-S SWS components can be mapped on the basis of their matching behavior. The next secttion provides the technical overview and analyzes functional constraints of OWL-S suite and its components.

4.3.1 OWL-S: Technical Overview

As stated above that OWL-S is suite of OWL ontologies and consists of the *Profile*, *Process Model* and *Grounding* ontologies. In this section I describe that what information these ontologies provide and how we can use them to address semantic limitations of BPEL processes.

Service ontology actually acts as organizer for the *Profile*, *Process Model* and *Grounding* ontologies. Each OWL-S service has one instance of the *Service* class and each

Table 4.1: Analytical description of BPEL process model activities with respect to mapping constraints.

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.

Service class has relation with *Profile, Process Model* and *Grounding* classes by using properties *presents*, *describedby* and *supports* (as show in the sample code below). One service can have only one *Service* ontology but can refer to multiple *Profile* ontologies to expose different semantically enriched interfaces for a single *Process Model* ontology. Successfull shifting of BPEL process to OWL-S SWS needs to extract *Profile*, *Process Model* and *Grounding* ontologies from a BPEL process model.

```
<service:Service>
 <service:describedBy>
  <process:CompositeProcess rdf:about="http://www.BPEL2OWLS.org/
           ChangeTestURI.owl#TestProcess"/>
 </service:describedBy>
 <service:presents>
  <profile:Profile rdf:about="http://www.BPEL2OWLS.org/
           ChangeTestURI.owl#TestProfile"/>
 </service:presents>
 <service:supports>
  <grounding:WsdlGrounding rdf:about="http://www.BPEL2OWLS.org/
             ChangeTestURI.owl#TestGrounding"/>
 </service:supports>
</service:Service>
```

Profile ontology is sub ontology of OWL-S suite and can be used to semantically describe Web service capabilities. Against the traditional syntactical interface exposed by a business process or a Web service , *Profile* ontology is used to present semantically enriched information (as input, output, pre-condition and effects (IOPE)) of a process as a SWS. Activities that create interface of a BPEL process model are therefore important so that they can be used to extract information required to create interface of a BPEL process as OWL-S SWS (i.e. *Profile* ontology (Section 5.4 covers it in detail)). In addition with describing capabilities of a service, the *Profile* ontology is also used to describe semantically enriched information about a required service so that a required service can be discovered on the basis of matching semantics (matching *Profile* ontology). Like IOPE, other service attributes (e.g service category, geographical location etc.) are used in the matching process to dynamically discover and compose a required service. The *Profile* ontology is used by the service provider and service requester to publish and discover Web services on the basis of matching semantics. Once a required service is discovered by computer agent the *Process Model* ontology describes how service works.

Process Model describes how to interact with a service. A *Process Model* is not a programme that can be executed but specifies different ways with which a client can interact with a service. Like a workflow language, the *Process Model* ontology can be used to model composition of multiple *atomic* and *composite* processes (services). While talking about shifting a BPEL process to OWL-S service, the *Process Model* ontology is worthwhile to describe control and data flow between sub *atomic* and *composite* processes. Like a BPEL process an OWL-S *Process Model* can have any number of inputs and outputs. Figure 4.1 provides an overview of the OWL-S *Process Model* ontology. It shows that *Process* class describes its Input, Output, Local (local variable) etc. by using object properties *hasInput, hasOutput, hasLocal* etc. where as *hasInput, hasOutput, hasLocal* are sub-classes of *Parameter* class. Figure 4.1 shows that *Atomic Process, Simple Process* and *Composite Process* are sub-classes of *Process* class. *Composite process* uses its object property *composedOf* to use *Control Construct* which has sub-classes *Sequence, Split, Split-Join, Repeat-While* and *If-Then-Else* that can be used to define control flow with in a *composite* process.

Grounding ontology describes about how to access a service by specifying message formats, protocols and transport. The message of complex data types are defined in the *Grounding* ontology by using the XSL Transformation [40]. *Grounding* ontology actually refers to the WSDL implementation of original service. When we discuss about *Grounding* of a *composite* service then it is actually a collection of *Grounding* ontologies of all sub *atomic* and *composite* processes involved in *Process Model* (Section 5.5 addresses this issue in more detail).

4.3.2 Processes

OWL-S has three kinds of processes (i.e. *simple*, *atomic* and *composite* processes) where as BPEL has two kinds of processes (i.e. *executable* and *abstract* processes). Here, we analyze

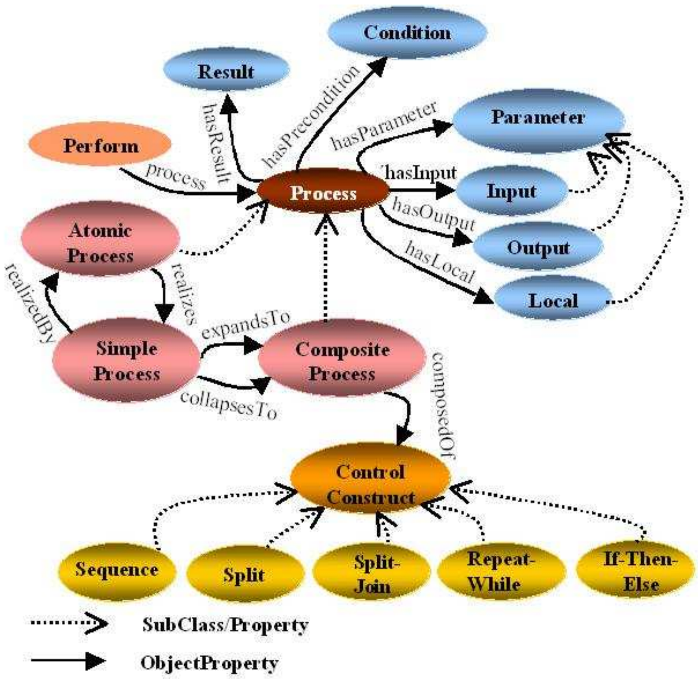

Figure 4.1: OWL-S *Process Model* ontology.

capabilities of these OWL-S processes so that we can have a comparison of capabilities and limitations of these BPEL and OWL-S processes for the purpose of mapping from BPEL to OWL-S.

Atomic Processes are processes that can be executed in a single step. *Atomic* processes are somehow like Web services operations that can be performed in a single step by sending it an input message and probably receiving some output message with in the whole larger BPEL process. OWL-S *atomic* process has no sub *atomic* or *composite* process. An *atomic* process is described by using the class *AtomicProcess* which is sub class of the *Process* class.

```
<owl:Class rdf:ID="AtomicProcess">
 <owl:subClassOf rdf:resource="#Process"/>
</owl:Class>
```

Simple Processes Unlike to *atomic* processes, *simple* processes are not invocable and are not associated with *Grounding* but like *atomic* processes can be executed in a single step. A *simple* process 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) [71].

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 compositions of multiple services on the basis matching semantics. A *composite* process allows to define the control flow between sub *atomic* and *composite* processes by using different CCs (e.g. *sequence, split, split-join* etc.). Since, a *composite* process is composition of multiple processes therefore, output of one process may need to be passed as input of the next process. Such a flow of inputs and outputs can be defined by addressing data flow and parameter binding issues.

4.3.3 Performing Individual Processes

As, I 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.

4.3.4 Control Constructs

I have discussed before that *Process Model* ontology is workflow like language and can be used to define workflow of sub *atomic* and *composite* processes. OWL-S defines many CCs that can be used to define control flow between sub processes with in *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) which 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 complets as soon as all of its process components are performed.

If-Then-Else CC can be used to implement Conditional behavior with in 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 then perform *then* part and if *ifCondition* is false then perform *else*).

Table 4.2: Analytical description of OWL-S ontology constructs with respect to mapping constraints.

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

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).

4.3.5 Condition Expressions

SWRL [81] expressions are most recommended standard to define conditions for OWL-S conditional CCs (e.g. Repeat-While, If-Then-Else etc.). OWL-S API [96] (developed by MIDSWAP Lab) supports the execution of conditions defined by using SWRL expressions. In Section 5.3.5 I explain how BPEL condition statements are translated to SWRL expressions in mapped OWL-S SWS so that process components of conditional CCs can be performed on the basis of true or false statuses of these SWRL expressions.

4.3.6 Data Flow and Parameter Binding

In BPEL process models, output of one Web service operation is passed as input of the next Web service operation by assigning values of their message parts. OWL-S defines a class *Binding* to define the data flow between process components. The *Binding* class can be used to specify that from which process the input is coming and what is input parameter and to which parameter of which process it is to be assigned. For sure, OWL-S specifications allows to define hard code values as input of a process (e.g. 5, "hello" etc.).

4.3.7 Parameters and Results

In OWL-S specifications, parameters are what we call variables in general programming languages and are expressed by using *Parameter* class. The type of OWL-S parameters can be expressed by a URI of a specific OWL class (defined in a domain ontology) which describes that the value of the parameter is of the type of that ontology class. The parameter type of a parameter can also be of usual type (e.g. int, string etc.).

Table 4.2 summarizes important components of OWL-S ontology and their description. On the basis of capabilities and limitations of components of BPEL and OWL-S I define mapping specifications for shifting BPEL process model to OWL-S ontology.

4.4 Summary

Translating business processes to SWSs can efficiently effect the automation of business processes as dynamic and automated composition of SWSs. Successfully shifting a business process (i.e. BPEL process) to OWL-S service is possible by defining relation between BPEL process and its components and OWL-S service and OWL-S CCs. In this chapter I have provided an analytical description of BPEL process model and OWL-S suite and their components. I have also discussed the possible behaviors that a single component can show in different situations and how different components of BPEL and OWL-S can be mapped on the basis of their matching behaviors. Since, OWL-S is suite of OWL ontologies therefore, I have described mapping constraints to translate a BPEL process to complete OWL-S suite of ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies).

I have also provided necessary information about syntax of BPEL and OWL-S components. Different issues, like defining control flow, data flow, condition statements, creating interfaces in BPEL and OWL-S have been discussed in detail. I have also discussed in detail that how individual Web services operations are performed in a BPEL process and how such operations are handled in a SWS language (e.g. OWL-S). On the basis of these mapping constraints I describe mapping specifications and step by step translation of BPEL process model (Appendix A) to OWL-S service (Appendix C) in Chapter 5 so that BPEL processes can be dynamically discovered, invoked and composed as OWL-S SWSs.

Chapter 5

Mapping BPEL Process Descriptions to OWL-S

In this chapter I present a mapping strategy (mapping specifications) that helps to overcome syntactical limitations of BPEL processes by mapping (translating) BPEL processes to OWL-S services. The proposed strategy supports the mapping of BPEL process descriptions to complete OWL-S suite of ontologies (i.e. *Profile, Process Model and Grounding* ontologies). Since, OWL-S is suite of *Profile, Process Model* and *Grounding* ontologies therefore, extraction of these ontologies from a BPEL process model has been discussed in detail.

5.1 Introduction

Providing machine understandable meanings of data and services on the Web is changing the way organization communicate and co-operate with each other in growing e-business world. The existence of established enterprize modeling methods, such as Business Process Modeling (BPM) methods, suggest that they could be exploited by emerging technologies such as semantic Web services to provide a more mature framework incorporating both business and Web application-specific technologies [54].

In previous chapter (Chapter 4) I described mapping constraints in detail which provide complete analysis of functional and non-functional aspects of BPEL and OWL-S and components of these languages. On the basis of these mapping constraints I describe mapping specifications and step by step translation of BPEL process to OWL-S suite of ontologies. Mapping specifications describe translation of a BPEL process description to complete OWL-S suite of ontologies. Another important feature of this work is that not only the BPEL process is translated to OWL-S composite service but all Web services operations within a BPEL process model are mapped to OWL-S *atomic* processes with complete OWL-S suite of ontologies. Consequently, the *Process Model* ontology of the mapped OWL-S service defines a semantic based workflow of sub processes and the *Profile* ontology provides semantically enriched information about capabilities of a process as

OWL-S SWSs that can be used to perform discovery, invocation and composition tasks in a dynamic and automated fashion. I have also implemented the use of multiple algorithms that help in more consistent and efficient translation process. Even though few efforts have already been done to bridge the semantic gap between process modeling languages (e.g. BPEL, FBPML etc.) and OWL-S SWSs, but my approach is unique with respect to its support for mapping of BPEL processes to complete OWL-S suite of ontologies as well as with respect to translation of individual Web services operations to OWL-S *atomic* processes. Such an approach helps to translate legacy systems into reusable, semantically described services. The mapping strategy is a more closer step to practically shift existing business systems from a syntactical to semantic based environment for the purpose of business process automation in semantic service oriented paradigm.

Furthermore, BPEL is mature enough as compare to OWL-S therefor, limitations of mapping specifications are also discussed at stages where direct mapping is not possible or needs some more work from OWL-S community. Since, there is no semantic information with in a BPEL process therefore, some areas are highlighted where end user needs some manual changes in mapped OWL-S service to enable it for dynamic discovery, invocation and composition by other semantic enabled systems.

Note:*Before it that I proceed with mapping strategy (mapping specifications), I would like you to recall some talks about Web service, semantic Web, SWSs and SWS languages from Chapter 2 and the example scenario (as discussed in Section 1.2). Recalling the SWS literature and example scenario will help to better understand the remaining chapter because in remaining chapter I will describe mapping specifications with respect to motivational scenario that is discussed in Section 1.2 and by using the BPEL process (Appendix A) which is modeled in MS BizTalk Server.*

The remaining chapter is organized as fellows: Mapping specifications have been discussed in Section 5.2. Section 5.3 briefly describes the mapping of BPEL process model to OWL-S *Process Model* ontology. Extraction of *Profile* and *Grounding* ontologies from BPEL process model have been discussed in Sections 5.4 and 5.5 respectively. Sections 5.6 summarizes this chapter.

5.2 Mapping Specifications

In previous chapter (Chapter 4), I have discussed in detail about process modeling and semantic capabilities of BPEL and OWL-S and components of these languages. Since, OWL-S is suite of OWL ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies) therefore, I describe mapping of BPEL process to OWL-S at three levels (i.e. mapping of BPEL process model to OWL-S *Profile, Process Model* and *Grounding* ontologies). Table 5.1 summarizes specifications for mapping BPEL process to OWL-S. Mapping specifications describe from abstract level to components and activities level translation of BPEL process to OWL-S service. Some of BPEL activities can not be directly mapped to OWL-S, as OWL-S does not provide CCs that have matching behavior to thees BPEL activities. Table 5.1 also highlights such activities that can not be directly translated to OWL-S CCs (e.g. Terminate, Throw and Wait activities). On the basis of these specifications I describe the mapping of BPEL process model to OWL-S in detail in remaining chapter.

Table 5.1: Summary of BPEL4WS to OWL-S mapping specifications.

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

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

Algorithm 1 provides a very abstract level definition of the recursive algorithm used for extracting OWL-S suite from BPEL process model according to mapping specifications described in Table 5.1. The algorithm is used to traverse through BPEL file's objects tree as long as activities in *BPEL file* come to end. An important thing to note in Algorithm 1 is that when an activity is non-I/O *primitive* activity then it is mapped to *perform* CC (as shown in lines 13 and 33 of Algorithm 1) to perform relevant *atomic* process and if an activity is an I/O *primitive* activity then it is used to create input/output parameters of *Profile* ontology (as shown in lines 21, 25 and 29 of Algorithm 1). Mapping specifications (as shown in Table 5.1) also describe that an I/O *primitive* activity is used to create input/output parameter of *Profile* ontology and a *primitive* activity which perform a Web service operation is mapped to *Perform* CC (Chapter 4 describes such behavioral aspects of BPEL activities and OWL-S CCs in more detail). In this section I have provided an abstract level definition of mapping specifications and mapping algorithm that can be used to map a BPEL process model to OWL-S suite. In next section I describe the extraction of *Process Model* ontology from BPEL process model on the basis of these mapping specifications.

```
Input: Tree view list of BPEL process and WSDL services
Output: OWL-S suite of ontologies
begin
    Extract BPEL activity from tree
    Map structured activity to OWL-S CC (Algorithm 2)
    Get child activities
    while child activities exist do
        if activity is not structured activity then
            if activity is assignment activity then
                while activity is assignment activity do
                    Traverse activity list
                end
                if activity is non-I/O primitive activity (i.e. invoke activity) then
                    Map it to perform CC to perform atomic process
                    Create data flow
                    Add reference of atomic process Grounding
                end
            end
            if activity is not assignment activity then
                if activity is I/O receive activity then
                    Create composite process input
                    Create Profile input parameters
                else
                    if activity is I/O reply activity then
                        Create composite process output
                        Create Profile output parameters
                    else
                        if activity is I/O invoke activity then
                            Create composite process output
                            Create Profile output parameters
                        else
                            if activity is non-I/O invoke activity then
                                Map it to perform CC to perform atomic process
                                Create data flow
                                Add reference of atomic process Grounding
                            end
                        end
                    end
                end
            end
        end
        if child activity is structured activity then
            Map structured activity to OWL-S CC (Line 3)
        end
    end
end
```

Algorithm 1: Abstract level definition of mapping algorithm.

5.3 Mapping to the OWL-S Process Model Ontology

In this section I describe how a BPEL process model is mapped to OWL-S *Process Model* ontology with defined control and data flow. The *Process Model* mapping specifications describe about how BPEL *primitive* and *structured* activities, condition statements, input/output data passing between different activities, variable etc. are mapped to OWL-S relevant CCs, SWRL expression and parameters. I also provide some code example of mapping of BPEL activities to OWL-S CCs. Whole mapping specifications depend on functional characteristics of BPEL and OWL-S components as described in Chapter 4. During whole discussion of mapping specifications we consider the translation of BPEL process (as shown in Appendix A) to OWL-S composite service (as shown in Appendix C). Now I describe step by step mapping of BPEL process components to OWL-S CCs.

5.3.1 BPEL Process to OWL-S Composite Process

BPEL process model is composition of multiple Web services operations with defined control and data flow to perform a joint task. A BPEL process model (orchestration) is mapped to OWL-S *composite* process which is a semantic based composition of multiple *atomic* and *composite* processes. Control flow and data flow between different Web services operations with in a BPEL process model is mapped to control flow and data flow between process components of an OWL-S composite process defined with in OWL-S *Process Model* ontology. An *atomic* process with in a *composite* process is result of mapping of a Web service operation that is performed by a *primitive* activity.

5.3.2 Web Service Operation to OWL-S Atomic Process

We discussed before that a BPEL process is composition of Web service operations which can be performed in single step. Each Web service operation with in a BPEL process is mapped to OWL-S *atomic* process (as show in Figure 5.1). The mapped *atomic* process consists of complete OWL-S suite of ontologies (i.e. *Profile*, *Process Model* and *Grounding* ontologies). Since, BPEL specifications are used to model the behavior and interaction between Web services and actual tasks 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 with in a BPEL process to OWL-S *atomic* process. As much as I 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 with in a BPEL process to OWL-S *atomic* processes. Appendix B describes a sample translation of a Web service operation (getTranslation) to OWL-S *atomic* process (getTranslationAtomicProcess) according to mindswap Lab's WSDL2OWL-S standards. During the mapping process each Web service operation is mapped to OWL-S *atomic* process and stored in a separate OWL file. An additional benefit of this step is that mapped *atomic* process can also be used with other semantic enabled systems. We can also execute these *atomic* processes by using some execution engine (e.g. OWL-S API) or by importing and executing them in SWSs development tool (e.g. Protégé (OWL-S Editor)).

5.3.3 Primitive Activity to Perform Construct

In above section I have discussed that a Web service operation performed by a *primitive* activity is mapped to OWL-S *atomic* process. The *primitive* activity which performs

Figure 5.1: OWL-S *atomic* processes generated from WSDL operations.

Web service operation is mapped to OWL-S *Perform* CC to perform that *atomic* process with in mapped OWL-S composite service. For example consider the *primitive* activity (*<invoke>*) (as shown in Example 4) that is used to perform Web service operation *getTranslation*. The Web service operation performed by this *primitive* activity is mapped to OWL-S *atomic* process (i.e. getTranslationAtomicProcess, as discussed in previous section (Section 5.3.2)) and stored in *getTranslation.owl* file and *primitive* activity is mapped to OWL-S *Perform* CC to perform relevant *atomic* process (i.e. to perform *atomic* process "getTranslationAtomicProcess" as shown in sample code below).

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

5.3.4 Structured Activity to OWL-S Control Construct

BPEL *structured* activities are used to define control flow between different child activities. OWL-S provides a number of CCs (e.g. *Sequence*, *Split* etc.) for defining control flow between sub processes. Table 5.1 summarize mapping of BPEL *structured* activities to OWL-S control constructs on the basis of their matching behavior. I have described in detail about behavioral characteristics of BPEL *structured* activities and OWL-S CCs in sections 4.2.4 and 4.3.4. Algorithm 2 shows a generic algorithm for mapping of BPEL *structured* activities to OWL-S CCs that are used to define control flow of mapped OWL-S service. As sample of mapping these activities I 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 and involve some important control flow aspects.

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* CC 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 I map *flow* activity to *Split-Join* CC on the basis of their matching functional characteristics.

```
Input: structured activity
Output: OWL-S CC
begin
    if activity equal to sequence then
        Map to Sequence CC
    else
        if activity equal to flow then
            Map to Split-Join CC
        else
            if activity equal to while then
                Map to Repeat-While CC
            else
                if activity equal to switch then
                    Map switch activity to Sequence of If-Then-Else CCs (Algorithm
                    3)
                end
            end
        end
    end
end
```

Algorithm 2: Mapping of *structured* activities to OWL-S CCs.

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 which 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 Section 5.3.5) 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* parts. Let us consider following *switch* activity example:

```
<switch>
 <case condition="bpel:getVariableData('Input\_Message',
                  'part', 'inputLang')='English'">
```

```
   <invoke partnerLink="Dictionary\_Ser\_Port"...../>
  </case>
 </switch>
```

Mapping of above *switch* activity to OWL-S is described in Example 6:

Example 6. BPEL *Switch* activity mapped to OWL-S *Sequence* of *If-Then-Else* CCs and *condition* statement translated to SWRL expression (In all remaining examples bpel4ws2owl[1] and dummyURI[2] are refereed to dummy URIs that are used by mapping tool).

```
<process:Sequence>
 ..........
 <process:If-Then-Else>
  <process:ifCondition>
   <expression:SWRL-Condition>
    <expression:expressionBody rdf:parseType="Literal">
     <swrl:AtomList xmlns:swrl="http://www.w3.org/2003/11/swrl#">
      ..........
      <swrl:BuiltinAtom>
       <swrl:builtin rdf:resource="http://www.w3.org/2003/11/
                         swrlb#equal">
       </swrl:builtin>
        ..........
       <rdf:first rdf:resource="&bpel4ws2owls#inputLang">
       </rdf:first>
        ..........
       <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#
                     "String">'English'</rdf:first>
       ..........
      </swrl:BuiltinAtom>
     </swrl:AtomList>
    </expression:expressionBody>
   </expression:SWRL-Condition>
  </process:ifCondition>
  <process:then>
   <process:process rdf:resource="&dummyURI#getMeaningProcess"/>
  </process:then>
 </process:If-Then-Else>
</process:Sequence>
```

In above example (i.e. Example 6) I have discussed a very simple conditional scenario in which *switch* activity involves single *case* element that is mapped to *If-Then-Else* CC. If a *switch* activity has multiple *case* elements which may have optional *otherwise*

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

branches then following algorithm (i.e. Algorithm 3) is used to traverse through the list of *case* elements and to map each *case* element to *If-Then-Else* CCs with in a *Sequence* CC.

Input: *switch* activity
Output: *Sequence* of *If-Then-Else* CCs in resulting OWL-S service

```
begin
    if activity equal to switch then
        Map switch activity to Sequence CC
        Traverse through switch activity
        while activity equal to case do
            Map case element to If-Then-Else CC
            Map condition statement to SWRL expression (algorithm 4)
            Go to Line 4 of Algorithm 1 to map child activities under case element
        end
        if activity equal to otherwise then
            Create else part of If-Then-Else CC
            Go to Line 4 of Algorithm 1 to map child activities under otherwise part
        end
    end
end
```

Algorithm 3: Algorithm to traverse through *Switch* activity and its *case* elements and to map them to relevant OWL-S CCs.

5.3.5 Condition Statement to SWRL Expression

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. I 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.). In previous section I have given an example (Example 6) of mapping of a *switch* activity to its relevant OWL-S CC (i.e. *If-Then-Else* CC) and its *condition* statement to SWRL expression. Algorithm 4 shows how *condition* statements are parsed and mapped to SWRL expressions (Example 6, lines 3 to 24 show a mapped SWRL expression of *If-Then-Else* CC).

```
Input: condition statement
Output: SWRL expression
begin
    Parse condition statement
    Extract left hand operands of condition statement ( i.e. message1_Name, variable1_Name
    and part1_Name )
    if variable1_Name equal to null and part1_Name equal to null then
        while list of Local Variables not ended do
            if local_Variable_Name equal to message_Name then
                Save reference of local variable as local_Variable1_Name
            end
        end
    end
    Find condition operator
    if right hand operand is message variable of an atomic process then
        Find index of "and" operator or "or" operator
        if "and" operator exists or "or" operator exists then
            Display message "Multiple conditions are not supported"
            Extract right hand operand of condition statement
        end
        Extract right hand operand of condition statement ( i.e. message2_Name,
        variable2_Name and part2_Name )
        if variable2_Name equal to null and part2_Name equal to null then
            while list of Local Variables (local_Variable_Name) not ended do
                if local_Variable_Name equal to message_Name then
                    Save reference of local variable as (local_Variable2_Name)
                end
            end
        end
    end
    Extract right hand operand (i.e. expression)
    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
        while condition operands not end do
            while list of atomic processes not ends do
                if operand equal to atomic process input then
                    Save reference of atomic process input
                end
                if operand not equal to atomic process input then
                    Find operand in output list of atomic processes and save its reference
                end
            end
        end
    end
    Generate SWRL expression;
end
```

Algorithm 4: Algorithm to parse *condition* statement and to generate SWRL expression.

The sample code given below describes some example *condition* statements which involve only input/output parameters of processes or input/output parameters of processes and local variables or *condition* statements that involve only two local variables or static values (e.g. 1, -1, "hello" etc. as shown in sample statements that are given below).

```
<case condition="bpel:getVariableData('Message','part','value')=-1">
```

```
<case condition="bpel:getVariableData('message1_Name',
                'variable1_Name',part1_Name)='English'">
```

An important thing that need here to be discussed is that complexity of *condition* statement can vary with complexity of message variables being used in *condition* statement. Because extracting message variable 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, Algorithm 4 handles the mapping of *condition* statements which involve variables of complex message types to SWRL expressions carefully and efficiently by parsing and tracking the list of *atomic* processes and their messages involved in condition statements.

5.3.6 Message Assignment to Data Flow

We can discuss the mapping of data flow at two levels: one is defining input and output of a *composite* process, second level of defining data flow is passing messages between process components with in *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 (e.g. Appendix A). Such a message which initiates a process is defined as input message of *composite* process with in *Process Model* ontology of mapped OWL-S service. In remaining process this message is referred as a message which 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*).

As an example consider a *receive* activity (as shown in Example 5) in a BPEL process which receives a message to start a process . The definition of message (*Input_Message*) received by *receive* activity is given in relevant WSDL file (as shown in sample code below).

Example 7. Input message schema of Translator service.

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

```
  </xs:element>
 </xsd:schema>
</types>
```

An important thing that need to be noted is that above message definition is based only on syntax and provides no semantic information that can be used by computer agents for the purpose of dynamic and automated discovery, invocation and composition of mapped OWL-S service. When such messages are mapped to OWL-S, they are annotated with domain ontologies to provide semantics for computer agents to reason about them (we discuss these issues in detail in Section 5.4). However, above message is used to define data flow as input of *composite* process (as shown below).

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

This input message of the *composite* process can be passed as input of sub processes (*atomic* or *composite* processes) with in *composite* process. For example, in our mapped OWL-S service (i.e. Appendix C), "getTranslation1" is an *atomic* process with in mapped *composite* process and sample code below (taken from Appendix C) shows that input parameter of *composite* process (i.e. *inputLang*) can be passed as input (*inputLanguage*) of the sub *atomic* process (*getTranslation1*).

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

I have also discussed that with in 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 with in a *composite* process is addressed by using the *Binding* class (as described in above sample code).

5.3.7 Variables to Local Parameters

Like traditional programming languages, we can also declare variables in a BPEL process to store and share data between different activities with in a process. Such variables with in 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 5.3.5 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 (as shown in sample code below).

```
<process:hasLocal>
 <process:Local rdf:ID="Dummy_Local_Var">
  <process:parameterType rdf:datatype="&xsd;#anyURI">
          &xsd;#float</process:parameterType>
 </process:Local>
</process:hasLocal>
```

In this section I have described translation of BPEL process model to OWL-S *Process Model* ontology. I also described the logic of translation of BPEL activities to OWL-S CCs on the basis of mapping constraints (discussed in Chapter 4). 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 evaluation chapter (Chapter 7)).

5.4 Mapping to the OWL-S Profile Ontology

Profile ontology is used to describe semantically enriched information about capabilities of a BPEL process when it is mapped to OWL-S service. 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 sperate OWL file. Since, BPEL process model provide no semantic information about a process therefor, *Profile* ontology parameters of mapped OWL-S service are automatically annotated by the mapping tool with dummy ontological concepts (URIs). Also, 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 semantics 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 tool (e.g. Protégé).
- Annotating *Profile* ontology parameters with these domain ontological concepts.

How to develop domain ontologies [60], edit (annotate) and develop SWSs with these domain ontologies is not the aim of this chapter. However, I 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 I describe criteria about how I extract a *Profile* ontology from a BPEL process model and automatic annotation of *Profile* ontology parameters with ontological concepts. Then I give that how end user can extended mapped *Profile* ontologies with their required domain ontologies.

5.4.1 Extracting the Profile Ontology

In Section 4.2.3 I have already discussed that *primitive* activities can have dual behavior: 1) to perform a Web services operations 2) to interact with outer world (i.e. to create the 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 5.3.2 and 5.3.3. Here, we are concerned with *primitive* activities that can be used to create interfaces of BPEL process models. A BPEL process can have one or more *primitive* activities (i.e. *receive, invoke* and *reply* activities) that are used to interact with the 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 the 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, we consider a *primitive* activity (<*receive*>) and its message that has three parts (i.e. *inputLang, outputLang* and *inputStr* as shown in Example 7). These message parts are used to create input parameters of resulting *Profile* ontology (as shown in Example 8).

Example 8. An example of mapped *Profile* ontology.

```
<profile:Profile rdf:about="&bpel4ws2owls#TestProfile">
 <profile:textDescription>This Profile is created by BPEL2OWLS Tool
 </profile:textDescription>
 <profile:hasInput>
  <process:Input rdf:about="&bpel4ws2owls#inputStr">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/
    XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#string
```

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

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. In sample *Profile* ontology given above (Example 8), we have an output parameter *return* which is part of the message received by *reply* activity used in the process to send output of the process to the outer world (Appendix A).

It is also possible that a *receive* activity do not has corresponding *reply* activity (as you can see in sample BPEL processes available with tool download) and BPEL process uses *invoke* activity to send output message to the outer world (as shown below).

```
<invoke partnerLink="Output_Port"
 portType="q1:Output_PortType_1" operation="Operation_1"
 inputVariable="Message_1_From_Dic_Service" />
```

In this case message (*Message_1_From_Dic_Service*) of the *invoke* activity is parsed in corresponding WSDL file and its message parts are used to create *output* parameters of *Profile* ontology of the mapped OWL-S service.

Till now I 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 need 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 critaria used for this purpose is that if a *receive* activity is being used as an initial activity to start a process (i.e. its *createInstance* attribute value is *yes*) and its *portType* and *operation* is supported by BPEL's corresponding WSDL file, then it is declared as an I/O activity.

Another thing that I think is important to highlight here 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 which 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 mapping from BPEL process model to OWL-S suite extracts one *Profile* ontology for one *Process Model* ontology.

5.4.2 Annotating Profile Ontology Parameters

In previous section (Section 5.4.1), I 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 (Example 8) 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 ontologies (Figure 5.2 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.

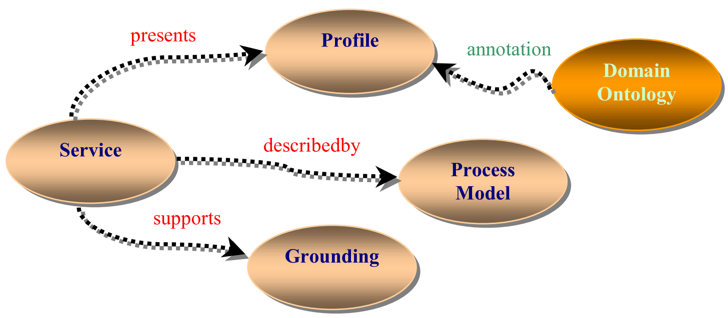

Figure 5.2: Annotating *Profile* ontology with domain ontology concepts.

As discussed above that 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 of mapped OWL-S service to provide different meaning of same service. Protégé with its OWL plugin [65] is an ideal framework for developing domain ontologies. Example 9 provides 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 OWL-S service.

Example 9. 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;#German"/>
  <factbook:Language rdf:about="&factbook;#French"/>
  <factbook:Language rdf:about="&factbook;#Dutch"/>
  ..........
  .......... (list of supported languages)
 </owl:oneOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="canBeTranslatedTo">
 <rdfs:comment>The relation that tells which language can be
       translated to which language</rdfs:comment>
 <rdfs:domain rdf:resource="#SupportedLanaguage"/>
 <rdfs:range rdf:resource="#SupportedLanaguage"/>
</owl:ObjectProperty>

<rdf:Description rdf:about="&factbook;#English"><canBeTranslatedTo
     rdf:resource="&factbook;#German"/></rdf:Description>
<rdf:Description rdf:about="&factbook;#English"><canBeTranslatedTo
     rdf:resource="&factbook;#French"/></rdf:Description>

.......... (list of supported languages)
```

Suppose that above language ontology is defined at following address *http://bis.informatik.uni-leipzig.de/LanguageOntology.owl* (shortly "&Languages"). Then the mapped *Profile* ontology (as show in Example 8) 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#inputStr">
    <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>
```

5.5 Mapping to the OWL-S Grounding Ontology

Grounding ontology of the OWL-S service describes how to access a service. Access details described in *Grounding* ontology include information about protocol, transport and message formats. These details enable *Grounding* to provide concrete level specifications needed to access a service. Concrete level definition of inputs/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 service. When a Web service operation with in 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 5.4 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 with in the same document[3] or can be defined in separate OWL file and can be referred in the type parameter[4].

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

```
<grounding:WsdlGrounding rdf:about="&bpel4ws2owls#TestGrounding">
```

[3] http://www.mindswap.org/2004/owl-s/services.shtml BabelFish Translator service provide such example

[4] http://www.mindswap.org/2004/owl-s/services.shtml Find Cheaper Book Price service provide such example

```
 <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 groundlings of two *atomic* processes (i.e. getTranslationAtomicProcess and getMeaningAtomicProcess) which are sub processes with in mapped *composite* process. I have described in detail in Section 5.3.2 that Web services operations with in a BPEL process model are mapped to OWL-S *atomic* processes and the mapped OWL-S composite service is semantic based composition of these *atomic* processes. The sample code given below provides an example of *Grounding* ontology of the getTranslationAtomicProcess *atomic* process (Appendix B).

```
<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>
```

5.6 Summary

To this end, bridging the semantic gap between business process modeling languages and semantic Web services becomes more important to keep existing business processes alive with upcoming semantic Web service technologies. In this chapter I have presented a mapping strategy that can be used to bridge the semantic gap between BPEL process models and SWSs by translating BPEL process descriptions to complete OWL-S suite of ontologies. Such approach helps in business process automation as dynamic and automated composition of business processes as OWL-S services. Extraction of OWL ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies) have been described individually. I have implemented the above discussed mapping strategy as a tool that can be used to map BPEL processes to OWL-S services (as discussed in next chapter (Chapter 6)). Critical mapping issues (e.g. mapping of *condition* statements, translating BPEL activities to OWL-S CCs, generating *Profile* ontology parameter from complex I/O messages etc.) have been addressed by implementing efficient parsing and mapping algorithms.

Profile ontology of mapped OWL-S service can be annotated with user defined domain ontologies to describe service semantics. This *Profile* ontology of mapped OWL-S service can be used to expose semantically enriched interface of BPEL process model as OWL-S service. Computer agents can discover BPEL processes as OWL-S services on the basis of matching semantics (i.e. matching *Profile* ontology). *Process Model* ontology of mapped OWL-S service defines the control and data flow between child *atomic* processes on the basis of matching semantics and can be used to edit the composition on the basis of matching semantics of sub *atomic* and *composite* processes to model more complex services.

OWL-S specifications does not provide CCs for all activity of BPEL processes. Due to these limitations I have also highlighted different areas where direct mapping of BPEL activities to OWL-S CCs is not supported (as described in mapping specifications (Table 5.1)). In order to implement direct translations of BPEL activities (e.g. terminate, fault handling etc.) we need more consistent specifications of OWL-S to address these issues. I have also highlighted areas where inputs from end user are required (e.g. changing parameter types by annotating input/output parameters with domain ontologies etc.).

Chapter 6

Prototype Implementation

In this chapter I present prototype implementation of the approach presented in this thesis. The prototype is the main tool (i.e. BPEL4WS 2 OWL-S Mapping Tool[1]) that provides ways to verify applicability and evaluation of proposed approach. BPEL4WS 2 OWL-S Mapping Tool can be used to map existing business processes (i.e. BPEL processes) to complete OWL-S suite of ontologies. The tool also supports the mapping of individual Web services operations with in a BPEL process model to OWL-S *atomic* processes.

6.1 Introduction

Implementation of the BPEL4WS 2 OWL-S Mapping Tool is an important contribution of this thesis which can be used to enable business processes with semantics by mapping BPEL processes to OWL-S suite of ontologies so that BPEL processes can be dynamically discovered, invoked and composed as OWL-S services by other semantic enabled systems. An OWL-S service can not be discovered dynamically as long as it does not expose its semantically enriched interface as *Profile* ontology. Similarly, the *Process Model* ontology, as described before, is a workflow like language that can be used to define composition of multiple *atomic* and *composite* processes on the basis of matching semantics. The *Process Model* ontology of OWL-S suite can also be edited to model more complex service that can perform a required task. The *Grounding* ontology actually describes how to interact with an OWL-S service (i.e. message format, protocol etc.). Therefore, mapping BPEL process models only to *Profile* or *Process Model* ontologies can not enable business processes to be dynamically discovered, invoked and composed. Hence, to enable business processes for dynamic discovery, invocation and composition a tool should support the mapping of BPEL process to complete OWL-S suite of ontologies.

BPEL4WS 2 OWL-S Mapping Tool is capable of mapping BPEL processes to complete OWL-S suite of ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies). Each of the *Profile, Process Model* and *Grounding* ontologies are stored in separate OWL files for

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

their individual uses as well as the complete OWL-S service of mapped BPEL process is also stored in a separate OWL file. Another important feature of the mapping tool is that not only the business process but all Web services operations with in a BPEL process are mapped to OWL-S *atomic* processes with *Profile, Process Model* and *Grounding* ontologies and are stored in separate OWL files (as shown in Figure 5.1). These mapped *atomic* processes can also be used by OWL-S execution engines (e.g. OWL-S API) to be executed individually or to use with other composite services. BPEL4WS 2 OWL-S Mapping Tool is an open source project and has hundreds of downloads since the time it has been uploaded on the open source project directory (sourceforge.net).

The remaining chapter is organized as follows: Section 6.2 provides an overview of the work that has already been done by other research groups in this area so that we can compare the approach presented in this thesis with these already presented research works and to compare that how my approach and implemented tool is better than already existing approaches and tools. Important features of BPEL4WS 2 OWL-S Mapping Tool have been described in Section 6.3. Section 6.4 describes implementation of the mapping tool. General usage of the tool has been discussed in Section 6.5. Section 6.6 summarizes this chapter.

6.2 Related Work

Semantic enhancements in Web service technology and bridging the semantic gap between business processes and SWSs is increasingly important for interaction between business services and processes in a dynamic and automated fashion. 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 to compose Web services (WSDL-S services) on the basis of matching semantics. The work discussed in [88, 93] describes mapping of BPEL process model to OWL-S *Process Model* ontology. I have already criticized and pointed out drawbacks of this approach in my work [22]. Major drawbacks of [88, 93] are that they do not support *Profile* and *Grounding* ontologies. As I have already discussed before that without *Profile* ontology, *process model* of mapped OWL-S service can not be advertised, discovered, invoked and composed dynamically by other services. The work discussed in [80] describes a good effort to map WSDL services to DAML-S (updated to OWL-S) services.

Another effort [54] 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 [54] also supports only mapping of FBPML process model to OWL-S *Process Model* ontology. It does not support mapping of *Profile* and *Grounding* ontologies. The work discussed in [54] has almost same limitations as that of the work discussed in [88, 93] that I have already criticized in my work [22, 21]. We can summarize that there have many efforts been done to address semantic limitations of process modeling languages by mapping them to semantic Web service languages (e.g. OWL-S) but none of these efforts provide expressive and consistent solution. My work is unique with respect to its support for mapping of BPEL process model to complete OWL-S suite of ontologies and that it addresses some critical 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 my work is that I have implemented the use of OWL-S API in my tool to write resulting OWL-S service. It makes the overall architecture of the tool and its implementation more consistent with other SWS development tools (e.g. OWL-S Editor) and execution engines (e.g. OWL-S API).

6.3 Features of BPEL4WS 2 OWL-S Mapping Tool

In previous section (Section 6.2) I have discussed some efforts that have been done to bridge the semantic gap between process modeling and SWS languages by establishing a mapping between them. Limitations and drawbacks of these existing efforts have also been highlighted in Section 6.2. I have also described in my work [22, 21] that how my approach is better as compare to these existing approaches and how the proposed mapping tool provides more consistent and efficient solution to the prescribed problem. Here, I describe some features of the mapping tool that make this work unique, more consistent and more useful for process modeling and SWS community with respect to other approaches and tools that have already be discussed in Section 6.2.

Support for Complete OWL-S Suite. Since, OWL-S is suite of OWL ontologies (*Profile, Process Model* and *Grounding* ontologies) and each ontology has a specific role in making the SWS vision functional (i.e. dynamic and automated discovery, invocation and composition) therefore, fruitful mapping of BPEL process model to OWL-S needs mapping of BPEL processes to complete OWL-S suite of ontologies. To achieve this goal BPEL4WS 2 OWL-S Mapping Tool supports the mapping of a BPEL process to complete OWL-S suite of ontologies.

Atomic Processes. As I discussed in Section 6.2 that for semantic based composition of Web services each operation with in a BPEL process should be mapped to OWL-S *atomic* process so that these *atomic* processes can be used for semantic based composition with in mapped *composite* process. These *atomic* processes (like Web services operations with in a BPEL process) are used to perform small tasks with in whole mapped composite service. BPEL4WS 2 OWL-S Mapping Tool supports the mapping of each Web service operation involved with in BPEL process model to OWL-S *atomic* process.. *Atomic* processes created during the mapping process are according to WSDL2OWL-S (by MINDSWAP Labs) standards and can be used with other composite services and executed by execution engines (e.g. OWL-S API).

Execution. Mapped OWL-S composite service and *atomic* processes with in *composite* process should be executable by some execution engine (e.g. OWL-S API). Since, I have implemented the use of OWL-S API in my tool therefore, resulting *atomic* and *composite* processes can be executed by OWL-S API.

Efficiency. Mapping of complex business processes with defined control and data flow can affect efficiency of the mapping tool. Parsing and mapping of complex BPEL processes, WSDL services, condition statements, control and data flow etc. has been well addressed by using fast and efficient parsing and mapping algorithms.

Easy to Use. Rapid adaption of the mapping tool depends on how easily and efficiently it can be used by end users. Keeping in mind from end user's perspective, an easy to use interface has been provided and mapping of a BPEL process to OWL-S service can be completed in few steps (as shown in Figure 6.3).

Deployable Services. Mapped OWL-S services can be further edited to model more complex services and deployed on SWS registries which support SWS publications according to OWL-S standards. Such services can be discovered on the basis of matching semantics and can be dynamically composed by other semantic enabled applications and services.

Standards Support. Standards support is another important aspect of mapping strategy to make this work more useful for process modeling and SWS development community. BPEL4WS 2 OWL-S Mapping Tool supports the industry wide accepted standard (i.e. BPEL4WS 1.1, WSDL 1.1 and OWL-S 1.1) for better compatibility with other process modeling and SWS development tools.

Extensibility. BPEL4WS 2 OWL-S Mapping Tool is an open source project and provides a rich and easy to extend set of classes. It can easily be extended with upcoming OWL-S specifications and can be integrated with other SWS development tools. For example, I am working on extending and integrating it as BPEL4WS 2 OWL-S Import Plugin for Protégé (OWL-S Editor) that will help end users to directly import BPEL processes to SWS development environment (i.e. Protégé (OWL-S Editor)).

Implementation of a tool with above described features helps not only to over come limitations of previous works done in this direction by other research groups but also to provide a tool which is more consistent with upcoming technology standards and latest process modeling and SWS development tools. Next section describes implementation of the mapping tool in detail.

6.4 Implementation

BPEL4WS 2 OWL-S Mapping Tool is implemented in JAVA being a platform independent language that is used by most of the research and development community world wide. Since, mapping tool is an open source project therefore, it can be used by other research groups to be extended with upcoming versions of OWL-S for better and more consistent mapping as well as in upgrading their existing systems with semantic support (e.g. SwinDew (a peer-to-peer workflow management system) is upgraded to SwinDew-S) [91, 94, 89]. Another important thing in implementation of the tool is its compatibility with other SWS community tools and applications (e.g. Protégé (OWL-S Editor), OWL-S API etc.).

OWL-S API provides a set of Java APIs for programmatic access to read, write and execute OWL-S services and is developed and maintained by Evren Sirin at MINDSWAP

Lab. Major SWS development tools (e.g. OWL-S Editor) also uses OWL-S API as an execution engine to execute OWL-S services developed in OWL-S Editor. OWL-S API provides an execution engine that can invoke *atomic* processes that have WSDL or Universal Plug and Play Language (UPnP) [4] groundings, and *composite* processes that uses OWL-S control constructs (e.g. *Sequence*, *Split* etc.) [96]. 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 [96]. I have also implemented the use of OWL-S API in my tool to write OWL-S services, for BPEL process models, according to mapping specifications discussed in Chapter 5. Use of OWL-S API make it more easy to integrate my tool with other SWS development tools (e.g. OWL-S Editor).

6.4.1 Architecture

Overall architecture of the BPEL4WS 2 OWL-S Mapping Tool consists of three components (i.e. WSDL Parser, BPEL Parser and OWL-S Mapper) as shown in Figure 6.1. Since, overall functionality of the tool consists of two major steps (i.e. parsing and mapping) therefore, parsing and mapping components work together by passing their inputs and outputs to each other. Here, I describe functional description of these components in detail.

WSDL Parser. As it is clear from name that WSDL Parser parses each WSDL file with in a mapping project and creates their object views. An important feature of WSDL Parser is that it extracts information about operations supported by Web services and sends their information to OWL-S Mapper which maps each Web service operation to OWL-S *atomic* process. OWL-S Mapper writes *atomic* process descriptions in separate OWL files and saves them in atomic processes directory of mapping project.

BPEL Parser. BPEL Parser traverses through the input BPEL file and creates object view of a 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 5.4 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 parts of this activity to create input/output parameters of *composite* process which are ultimately 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, the BPEL Parser parses *structured* activities with in 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

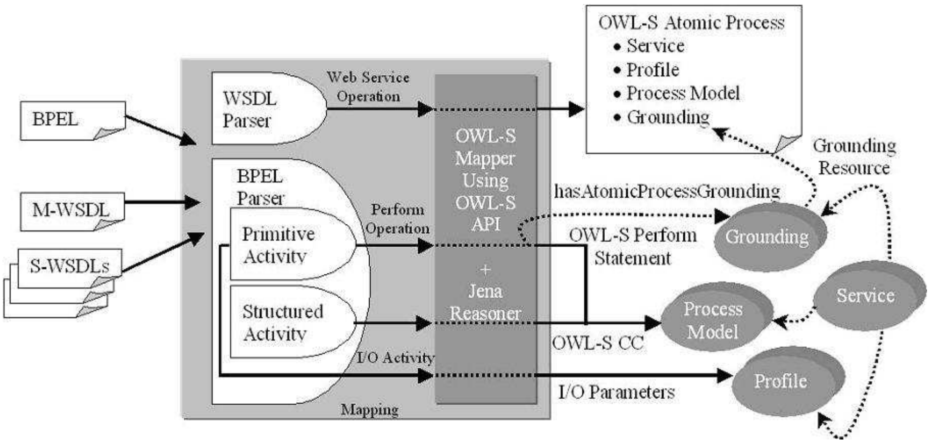

Figure 6.1: Architecture of the BPEL4WS 2 OWL-S Mapping Tool.

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 a conditional *structured* activity during parsing process then it simply sends conditional activity (e.g. Switch, While etc.) and condition string to OWL-S Mapper, which maps it to corresponding OWL-S CC and SWRL expression (as explained in Section 5.3.5) and use it with OWL-S CCs (e.g. If-Then-Else, Repeat-While etc.).

OWL-S Mapper. OWL-S Mapper is actually responsible for writing resulting OWL-S service according to defined mapping specifications. It uses the OWL-S API to write resulting OWL-S composite service. Since, OWL-S API uses a third party reasoner (i.e. jena reasoner) to reason the mapped OWL-S ontology therefore, my tool also uses jena reasoner (as default reasoner) for such reasoning purposes. OWL-S Mapper actually consist of classes that write OWL-S services by using local class structures or by using OWL-S API classes for OWL-S specifications.

6.4.2 User Interface

BPEL4WS 2 OWL-S Mapping Tool provides an easy to use interface which consists of four major parts (i.e. Project Explorer, Object Explorer, Content Window and Output Window), a *Toolbar* and a *Menu bar* as shown in Figure 6.2.

Project Explorer can be used to see project input and output files (i.e. input BPEL file, WSDL files (Master and Slave WSDL files), OWL files of mapped OWL-S suite of ontologies and OWL files of mapped OWL-S *atomic* processes). Object Explorer provides object view of input BPEL and WSDL files. We can look through input BPEL and WSDL files in a tree view control in Object Explorer. Traversing though tree view of input BPEL file also helps to find the sequence of activities with in a BPEL process model. Content window can be used to see contents of any of the input BPEL file, WSDL files and mapped OWL files. User can simply select a file in the Project Explorer and tool will open contents of the file in Content Window. Output of different actions performed (e.g. Validate, Build and Map) can be seen in Output Window.

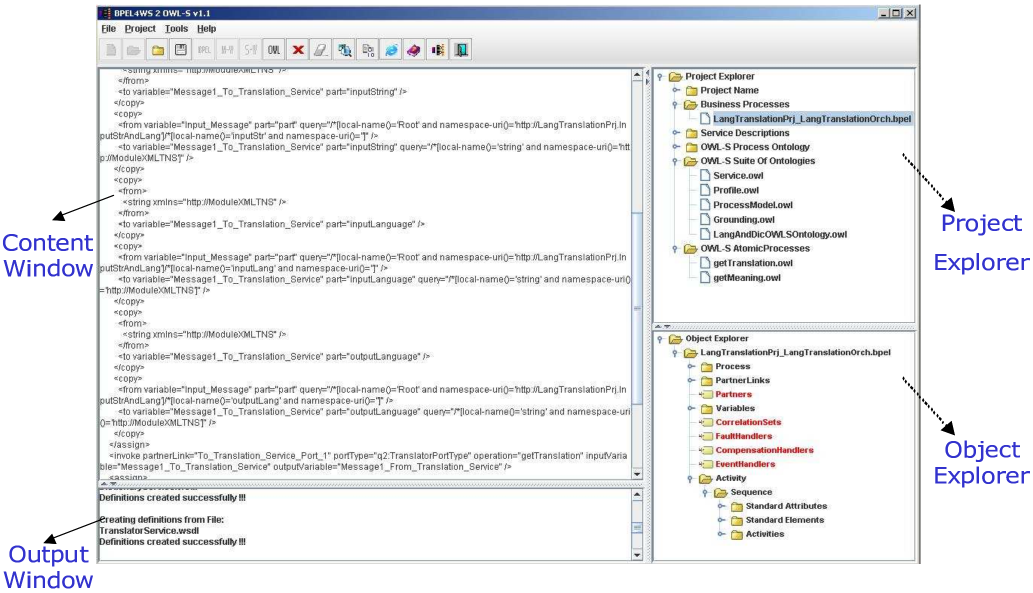

Figure 6.2: Overview of BPEL4WS 2 OWL-S Mapping Tool.

6.5 General Usage

The overall mapping process (as show in Figure 6.3) starts by creating a new project. As an input of the project, tool requires a BPEL file of the process and its corresponding WSDL file (I call it Master WSDL file (M-W)). As I discussed before, that a BPEL process is composition of Web services operations therefore, all Web services WSDL files (I call Web service WSDL file as Slave WSDL file (S-W)) involved in BPEL process are provided as an input of the mapping project. Then these input files are validated by performing validation operation.

Once input files are validated by the tool, next step is to build the project with these input files. Building a project is an important step because it parses BPEL and WSDL files and extracts information about all activities of a process and Web service operations involved in a BPEL process model. This information is used to create object view of activities and components of BPEL process and WSDL services. During mapping process this information is used to write resulting OWL-S service. As a last step, process is mapped to OWL-S, which results in four OWL files (i.e. *Service, Profile, Process Model* and *Grounding* ontologies files) and OWL file which contains complete suite of OWL ontologies. This OWL file can be used to execute by execution engines (e.g. OWL-S API). Also, during the mapping process each Web service operation is mapped to OWL-S

Step		Menu Item	Short Key
1	`Create New Project`	`File->New->Project`	`Ctr+p`
2	`Add BPEL File`	`Project->Add->BPEL File`	`Alt+b`
3	`Add Master WSDL File`	`Project->Add->M-W`	`Alt+w`
4	`Add Slave WSDL File`	`Project->Add->S-W`	`Alt+w`
5	`Validate Project`	`Project->Validate`	`Ctr+v`
6	`Build Project`	`Project->Build`	`Ctr+b`
7	`Map Project`	`Tools->Map->OWL`	`Ctr+m`

Figure 6.3: Sequence of steps (with menu items and short keys) to perform a mapping task.

atomic process and stored in a separate OWL file which is used to perform sub process with in OWL-S composite service.

6.6 Summary

BPEL4WS 2 OWL-S Mapping Tool is an easy to use tool that can be applied by process modeling and SWS development communities to map existing business processes (BPEL processes) to OWL-S services. The mapping tool discussed in this chapter make it easy to enable existing business processes with semantics rather than to build these processes as composite services in a SWS development environment (e.g. Protégé (OWL-S Editor)) from scratch. Its support for industry wide accepted standards and easy to use interface make it a tool of choice for end users. Compatibility and extensibility features of the mapping tool resulted in more interest of the semantic Web and SWS research and development communities in this work. Large projects (e.g. SwinDew) and semantic Web and SWS development tools (e.g. Protégé and OWL-S Editor) have shown their interest in this work. I have also collaborated with SwinDew (a peer-to-peer workflow management system) research and development team on enhancing SwinDew to SwinDew-S by enabling it with semantic support by shifting existing business processes to OWL-S services. OWL-S Editor team has also shown their interest in this work by pointing out need for a tool that can be used to directly import BPEL processes in to OWL-S Editor as OWL-S services. I am currently in touch with OWL-S Editor team in making the tool available as BPEL4WS 2 OWL-S Import Plugin for Protégé (OWL-S Editor).

Chapter 7

Evaluation

In this chapter I provide an evaluation of the approach presented in this thesis by answering research questions that I highlighted in Chapter 1. Most of the results of this thesis that I use to answer research questions have been published in international workshops and conferences. Even though previous chapters provide detail answers of research questions as my research contributions but here I would like to summarize them for the purpose of evaluation. After answering the research questions I provide an evaluation of the proposed approach by answering and evaluating the overall research question (as already described in Section 1.3).

The remaining chapter is organized as follows: In Section 7.1 I summarize answers to the research questions that I raised in Section 1.3. Section 7.2 takes an evaluationary revision of the example scenario (as discussed in Section 1.2) for the purpose of evaluation of the proposed approach. Answer to the main research question and its evaluation is described in Section 7.3. Section 7.4 summarizes this chapter.

7.1 Answers to Research Questions

The overall research question that I described in Chapter 1 is:

How existing business processes can be shifted from a syntax based to semantic based environment to enable them for semantic based composition editing, modeling and dynamic discovery, invocation and composition by other semantic enabled systems?

The main research question has been answered in small research contributions. To answer the main research question and to evaluate my work, first, I provide answers of small research questions that I raised in Section 1.3 and then I describe my overall research contribution for bridging the semantic gap between business processes and SWSs to shift existing business processes from a syntax based to semantic based environment. The overall system for shifting existing business processes from a syntactical to semantic based environment consists of theoretical concepts, approaches and their prototypical implementation. For example, a new 4-tier SWS integration architecture has been presented in Chapter 3 that addresses architectural requirements for business process integration as SWSs composition. A life cycle for SWS composition and a framework for dynamic and automated composition of Web services has also been discussed in Chapter 3. Bringing these theoretical concepts at more concrete level, I presented an approach that can be used to establish correspondence between syntax based and semantic based composition of Web services (as discussed in Chapter 4). I have also presented mapping specifications and algorithms that can be used to map BPEL processes to OWL-S services (as discussed in Chapter 5). An implementation of these theoretical concepts and approaches has been described in Chapter 6.

By using these research contributions, here, I answer to the small research questions which helps to understand and to evaluate the main research contribution.

RQ 1. What Web service is and how we can provide Web service semantics?

- Web services are viewed as platform independent reusable applications that provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web service related standards (i.e. SOAP, WSDL and UDDI) make it accessible and invocable over a variety of platforms.
- Different SWS languages (e.g. WSDL-S, WSMO and OWL-S) have been viewed and discussed in Chapter 2 for the purpose of adding semantics to Web services. Capabilities and limitations of these SWS languages have been discussed which help to evaluate these languages. I have provided a comparison of these SWS languages and proved that semantic and workflow modeling capabilities of OWL-S are much better as compare to other SWS languages. That's why I have choosed OWL-S as SWS language that can be used to overcome semantic limitations of BPEL.

RQ 2. Is existing application integration architecture and framework enough for semantic based dynamic integration and composition of business processes as SWSs?

- I have discussed different approaches for dynamic and automated Web services composition that have been presented both from process modeling and AI communities. Even though these approaches are good initiative towards SWS composition but I have pointed out some requirements that need to be addressed for semantic based integration and composition of business processes as SWSs in real world scenarios. I also evaluated some existing dynamic and automated Web service composition approaches with respect to these requirements and proved that none of existing approach address all of these SWS composition requirements. As a solution of these issues, I have presented a dynamic and automated Web services composition framework at an abstract level.
- Limitations of traditional 3-tier application integration architecture have been discussed in detail in Chapter 3. I also proposed a new 4-tier SWS integration architecture (as discussed in Section 3.4) that addresses syntactical limitations of traditional 3-tier application integration architecture. The newly proposed

4-tier architecture helps to meet semantic based dynamic Web service integration and composition requirements.

 - Modeling Web services composition at design time or to dynamically discover and compose required services needs Web services composition (workflow) to be available in a machine executable language (e.g. BPEL or OWL-S *Process Model* ontology). In such a workflow, management people can add business and management rules with in composition and technical people can use them to be implemented in language that is processable and understandable for machines. For such a collaborative work between business and technical people I have presented a SWS integration and composition life cycle (discussed in Section 3.5) which brings all SWS related efforts in one circle.

RQ 3. How correspondence can be established between syntax and semantic based composition of Web services (i.e. BPEL process model and OWL-S composite service)?

 - As, I discussed before that I have evaluated different SWS languages and comparison of these SWS languages shows that OWL-S has more expressive semantics and workflow modeling capabilities as compare to other SWS languages. I have also elaborated my approach to semantically enrich business processes by expressing business process models (e.g. BPEL process models) as OWL-S services which are semantic based compositions of Web services. Furthermore, in Chapter 4, I have described that different OWL-S control constructs (CCs) can be used to define control flow between child processes with in whole composite service.
 - *Process Model* ontology of OWL-S suite is used in the whole process of establishing correspondence between workflow modeling capabilities of (BPEL) and SWS language (OWL-S). *Process Model* ontology of OWL-S suite can be used to model the composition of multiple services (*atomic* and *composite*) on the basis of their matching semantics. Different OWL-S CCs (e.g. *Sequence*, *Flow* etc.) can be used to define control flow between child *atomic* and *composite* processes with in whole OWL-S composite service. Semantically enriched interface of BPEL process is expressed as *Profile* ontology of OWL-S suite.

RQ 4. How a BPEL process model can be mapped and expressed as OWL-S SWS?

 - In chapter 4, I have described in detail that activities of a BPEL process that interact with outer world are used to create interface of mapped OWL-S service. Also, messages of these activities are used to create input/output parameters of the mapped OWL-S service. These input/output parameters are automatically annotated by the mapping tool with dummy ontological concepts. These dummy ontological concepts can be changed with user defined domain ontologies to expose semantically enriched interface as *Profile* ontology of mapped OWL-S service. This *Profile* ontology is used by other semantically enriched systems to dynamically discover a business process as OWL-S service.
 - It is discussed in detail in Chapters 5 and 6 that a BPEL process is parsed by BPEL Parser to create object view of BPEL process. Activities with in this

object tree of BPEL process are sent to OWL-S Mapper to create control flow of *composite* process with in *Process Model* ontology of mapped OWL-S composite service. BPEL *structured* activities are mapped to OWL-S CCs to define control flow and *primitive* activities are used to create the interface of mapped OWL-S service or to perform sub *atomic* processes with in mapped OWL-S *composite* process. *Process Model* ontology of mapped OWL-S service can be used to edit the composition of services in a semantic enabled environment to model more complex service to perform required tasks.

– Interaction protocol and messages exchanged between partner services are describe in *Grounding* ontology of mapped OWL-S service. The mapping tool extracts information about messages of Web services operations used in BPEL process model and describes them as inputs and outputs of *atomic* and *composite* processes of mapped OWL-S service. As discussed before in Section 5.5 that it is not possible to automatically write XSLT scripts for XSL Transformation of complex Web services messages and end user has to put some manual efforts in this area.

RQ 5. Is translation of BPEL process models to OWL-S ontologies can help for semantic based discovery, invocation and composition of BPEL processes as OWL-S services?

– When a BPEL process model is mapped to OWL-S suite of ontologies, the *Profile* ontology of mapped OWL-S service can be used for reasoning purposes by computer agent to dynamically discover a BPEL process as OWL-S service on the basis of matching *Profile* ontology. Input/output parameters of *Profile* ontology of mapped OWL-S service, when annotated with domain ontologies, provide universally unique meaning to expose a service capabilities. These universal meanings of input/output parameters pre and post conditions of a service make a Web service capabilities understandable for machines.

– Once a BPEL process is mapped to OWL-S service, different execution engines (e.g. OWL-S API) can be used for its execution.

In this section I summarized answers to the research questions raised in Chapter 1 and also referenced to other chapters where readers can find further conceptual and technical details about how a specific research question has been answered. Before summarizing the answer to the main research question and to evaluate the proposed approach we first have an evaluationary revision of the motivational scenario discussed in Section 1.2. It will help not only to recall that what the main problem was but also to understand that how the proposed research approach addressesg the problem.

7.2 Motivational Scenario: An Evaluationary Revision

For evaluation purpose, here, we have an evaluationary revision of the problem scenario (motivational scenario discussed in Section 1.2) and see that how the approach presented in this thesis and its prototypical implementation answers the overall research question i.e.

How existing business processes can be shifted from a syntax based to semantic based environment to enable them for semantic based composition editing, modeling and dynamic discovery, invocation and composition by other semantic enabled systems?

In Section 1.2 I highlighted two problem tasks (as shown in Figures 1.1 and 1.2). For the first problem task (as shown in Figure 1.1) I modeled a BPEL process in MS BizTalk Server as syntax based composition of Web services. Then I pointed out that such a syntax based Web services composition (process) has following limitations:

1. When such process is exported as a Web service, it has same syntactical limitations as traditional WSDL service resulting in clampdown of process for dynamic discovery, invocation and composition.

2. If we want to extend (edit) the process (discussed in first scenario (Figure 1.1)) in a semantic environment (i.e. to edit and model the composition on the basis of matching semantics) to perform the task defined in second scenario (Figure 1.2) then we will realize that:

    (a) Web services with in composition provide no information for semantic based editing and modeling of process.

    (b) Semantic limitation of Web services with in process restrict to dynamically discover and compose (on the basis of matching semantics) other SWSs (e.g. semantically matching *Translator* service).

If I describe these problem at more concrete level and with more precise wording then I can define it as:

1. How we can expose semantically enriched interface of a process to enable it for semantic based dynamic discovery, invocation and composition?

2. How composition of services can be edited and modeled on the basis of matching semantics rather than to compose them just on the basis of syntactical information?

In Chapter 3, I proposed new concepts for architectural changes in Web service related machinery to address dynamic discovery, invocation and composition issues that are raised with semantic enhancements in Web service. In chapter 5, I proposed a more concrete level solution of the problem by presenting a strategy that can be used to map existing business processes (BPEL processes) to OWL-S services. I also described step by step translation (mapping) of a BPEL process (syntax based Web services composition) (Appendix A) to OWL-S composite service (semantic based Web services composition) (Appendix C). In next section I provide an evaluation of my work by describing that how much successfully my approach answer to the main research question (that I have described in two parts, as discussed above).

7.3 Answer to the Main Research Question

In chapter 5, I presented a mapping strategy as mapping specifications that can be used to translate BPEL processes to OWL-S services. I also described step by step mapping of BPEL process (Appendix A) to OWL-S SWS (Appendix C). As a result of this step by step mapping, till the end of Chapter 5 whole BPEL process (Appendix A) was mapped to OWL-S service (Appendix C) with each Web service operation with in BPEL process mapped to OWL-S *atomic* process (e.g. Appendix B). Here, I describe that how a BPEL process (Appendix A) when mapped as an OWL-S service (Appendix C) can be used for dynamic discovery by using the *Profile* ontology of mapped OWL-S service and how the *Process Model* ontology of mapped OWL-S service can be edited to model more complex services on the basis of matching semantics that can perform a required task.

7.3.1 Semantically Enriched Interface

In Section 5.4, I have discussed in detail that how a *Profile* ontology is extracted from BPEL process model and how we can annotate it with domain ontologies to provide semantically enriched interface of BPEL process as OWL-S service. To further understand that how this *Profile* ontology is used to expose semantically enriched interface of BPEL process to facilitate dynamic discovery of BPEL process as OWL-S service, let us consider a small part of *Profile* ontology of mapped OWL-S service (also shown in Example 8 and in Appendix C).

```
<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:Profile>
```

In above sample *Profile* ontology, parameter type of input parameter (*inputLang*) is "string". Similarly parameter type of other parameters of the *Profile* ontology (as shown in Example 8) of mapped OWL-S service is also "string" that provide no meaning for computer agents to reason about these *Profile* ontology parameters for the purpose of dynamic discovery. Now if we annotate these input/output parameters with domain ontologies (as discussed in Section 5.4) then above *Profile* ontology of mapped OWL-S service looks as follows:

```
<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>
 .....
 .....
</profile:Profile>
```

Above sample code shows that the input parameter (*inputLang*) is of type "SupportedLanguage" defined in an appropriate domain ontology that is defined at the following address "&languages [1]". This parameter has unique meaning for all computer agents and can be reasoned by other semantic enabled systems to dynamically discover the BPEL process as OWL-S service on the basis of matching semantics.

Before mapping a BPEL process to OWL-S service, the interface exposed by a BPEL process provides only syntax based information. Such syntactical interface can be used by human agents to define interaction between two processes that are exposed as traditional WSDL services but not by computer agents to discover them dynamically. After mapping a BPEL process to OWL-S SWS, the interface exposed by mapped OWL-S service as *Profile* ontology provides semantically enriched information about capabilities of a BPEL process as OWL-S service. Such a semantically exposed interface (*Profile* ontology) of mapped OWL-S service is understandable for human as well as for machines. Now after mapping a BPEL process to OWL-S service it can be dynamically discovered by using *Profile* ontology of mapped OWL-S service by using different dynamic SWS discovery approaches (e.g. [84, 108, 105, 16]).

7.3.2 Semantic Based Composition

In previous section (Section 7.3.1), I described in detail that how the *Profile* ontology of mapped OWL-S service can be used to provide semantically enriched interface of BPEL process as OWL-S service to enable it for semantic based dynamic discovery. Here, I answer the second part of overall research question (i.e. how *Process Model* ontology of mapped OWL-S service can be used to edit and model the composition of Web services on the basis of matching semantics) and how different approaches can be used to utilize *Process Model* ontology of mapped OWL-S service to dynamically compose other services with in composite process.

In Section 1.2 (motivational scenario), I defined two problem tasks and modeled a BPEL process for first task (as shown in Figure 1.1) as syntax based composition of

[1] http:www.uni-leipzig.deLanguages.owl

multiple services (i.e. *Translator* service and *Dictionary* service). Then I claimed that these syntactical limitations of BPEL process model can be addressed by mapping it to OWL-S suite of ontologies, which enables a BPEL process for dynamic composition as OWL-S SWS. For this purpose, in addition with some architectural changes in SWS related machinery I described a strategy to map BPEL processes to OWL-S services. As I have explained in detail in Section **??** that how a *Process Model* ontology is extracted from BPEL process model, here, I show that how the *Process Model* ontology of mapped OWL-S service (i.e. *Process Model* ontology as composition of *Translator* and *Dictionary* service to perform the task defined in first problem scenario) can be edited on the basis of matching semantic information rather than syntactical information to perform the task defined in second problem scenario (as shown in Figure 1.2).

As a first step to edit mapped OWL-S service to perform the task discussed in second scenario (Figure 1.2), we replace dummy URIs of input/output parameters of mapped *atomic* and *composite* processes with domain ontologies (as discussed in Section 5.4). 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 8.4) or in any other editor (e.g. Notepad). Annotating input/output parameters helps to edit and extend the *composite* process with in *Process Model* ontology by defining data flow between sub processes on the basis of matching semantics. Mapped OWL-S service (Appendix C) takes *inputString, inputLang* and *outputLang* as input parameters. Semantically enriched definition about these input parameters is provided by annotating them with domain ontologies (as discussed in Sections 5.4 and 7.3.1). Annotation of input/output parameters of *atomic* and *composite* processes shows that input parameters *inputLang* and *outputLang* are of type "SupportedLanguage" and *inputStr* is of type "string".

The first *atomic* process (*getTranslationProcess1*) with in *composite* process of mapped OWL-S service (Appendix C) translates the input string from input language (i.e. German, which is defined in domain ontology "Languages.owl") to output language (i.e. English). The second *atomic* process (i.e. *getMeaningProcess2*) provides meaning of input word in English language. From here we start editing the *Process Model* ontology of mapped OWL-S service (Appendix C) and add one more *atomic* process (i.e. *getTranslationProcess3*) with in the *Sequence* CC of *composite* process (as shown in Appendix D, Lines 79 to 86). We define data flow for this newly added *atomic* process (i.e. *getTranslationProcess3*) which takes as input (*inputLang*) (value of input parameter *inputLang* is English which is of type "SupportedLanguage"), *outputLang* (i.e. German that is also of type "SupportedLanguage") and *inputStr* (output of *atomic* process *getMeaningProcess2*) (i.e. meaning of German word in English) as shown in Appendix D, Lines 146 to 181. The data flow can be defined by using data binding between *atomic* processes (as discussed in Section 4.3.6). The data flow between atomic processes is defined on the basis of matching semantics. Appendix D shows extended OWL-S service by adding an *atomic* process (*getTranslationProcess3*) with in defined control flow of *composite* process and with defined data flow.

Same process of editing and composing required services with in composite process on the basis of matching semantics can be performed dynamically by using different dynamic composition approaches (e.g. [100, 23, 98, 101, 83] etc.). For example, the dynamic composition approach discussed in [100] can be used to define abstract process for a required service with in *Process Model* ontology of mapped OWL-S service (even though I have highlighted that existing approaches have some open issues to dynamically compose services on the basis of matching functional and non-functional semantics). Then AI planner, as discussed in this work can be used to dynamically discover and compose matching service, which is not possible to do with a syntactical Web service composition language (e.g. BPEL).

In Section 1.2, I defined two major problems of syntax based Web services composition 1) syntactical interface 2) static syntax based Web services composition. I addressed both of these problems by proposing semantic enhancements in Web services infrastructure and by mapping BPEL process to OWL-S with the help of BPEL4WS 2 OWL-S Mapping Tool. The *Profile* ontology of mapped OWL-S service provide semantically enriched information about BPEL process as OWL-S service. Mapped OWL-S service (Appendix C) is edited and extended (Appendix D) on the basis of matching semantic information rather than syntactical information to perform the task defined in second scenario (Figure 1.2).

7.4 Summary

In this chapter I have answered to research questions that are raised during my research work while working on bridging the semantic gap between business processes and SWSs. The main research question has been answered in general, as theoretical approach and its implementation that can be used to map existing BPEL processes to OWL-S SWSs so that BPEL processes can be dynamically discovered, invoked and composed by other semantic enabled systems. I have also provided an evaluation of the approach presented in this thesis. Evaluation of the work discussed in this thesis shows that the presented approach and its compatibility with industry wide accepted standards can be used to easily shift existing business processes from a syntactical to semantic based environment to meet challenges of upcoming dynamic e-business world. During the evaluation, I have also described some limitations of the proposed approach and possible solutions. More friendly environment can be provided to end users by integrating the mapping tool with some other process modeling and SWS development tools (e.g. OWL-S Editor).

Evaluation of the proposed approach shows that business processes when mapped as OWL-S services can be used for semantic based composition editing and modeling of complex services and for dynamic discovery, invocation and composition. The *Process Model* ontology of mapped OWL-S service can be edited and more complex composite service can be modeled on the basis of matching semantics to perform required task. The *Process Model* ontology of mapped OWL-S service can also be used to define abstract processes with in composite service that can be used to dynamically discover and compose required services. AI planning techniques can also be used to automatically compose business processes as OWL-S services. The *Profile* ontology of mapped OWL-S service can be used to expose semantically enriched interface of BPEL process as OWL-S service. This semantically enriched interface enables computer agents to dynamically discover a business process as OWL-S service and to compose it with other services to perform required task.

Chapter 8

Discussion and Conclusion

In this thesis I identified some challenges for business process automation (e.g. syntactical interface, syntax based composition, static binding of services, no computer understandable semantics, lack of reasoning support and lack of architectural approach and framework for semantic enhancements in business process). I addressed these challenges by proposing a new architectural approach and framework for SWSs composition as well as by presenting a strategy that can be used to overcome syntactical limitations of process modeling languages (e.g. BPEL) by translating BPEL process descriptions to OWL-S suite of ontologies. A prototypical implementation of the proposed approach has also been presented as a mapping tool (i.e. BPEL4WS 2 OWL-S Mapping Tool).

The remaining chapter is organized as follows: Section 8.1 provides an open discussion on the raised research problem and its proposed solution. Section 8.2 describes the application area for the work presented in this thesis. Section 8.3 summarizes research contributions and their impact. Some open issues and future work has been discussed in Section 8.4.

8.1 Discussion

Architectural and technological aspects, and tools support is very important for successful semantic enhancements in Web service, so that SWS can be easily and efficiently adopted by industry and academia. As an example we can consider Web service technology and SOA support for Web service and different tools (e.g. MS Visual Studio, MS BizTalk Server, IBM WebSphere, SAP NetWeaver etc.) that can be used to develop Web services and to model business processes as Web services compositions. To address architectural, technological and implementation issues we need a very detailed work to be done by SWS and its related communities. I addressed these architectural and conceptual issues to the extent of the need of my work (i.e. to make business processes enable for dynamic discovery, invocation and composition as SWSs by other semantic enabled systems). I have presented a new architectural approach for integration and composition of business processes as SWSs. In traditional application integration architecture, applications (services) interact with each other by using syntactical interfaces exposed by these services, but the 4-tier architecture proposed in this thesis addresses interaction issues between applications (services) on the basis of their semantically enriched interfaces, which results in dynamic interaction between services. A framework has also been presented at an abstract level which describes the semantic based dynamic integration and composition of business processes as SWSs.

Following the top down approach for my work I have presented a mapping strategy that can be used to easily shift existing business processes to OWL-S services by mapping BPEL processes to OWL-S suite of ontologies (i.e. translating BPEL process descriptions to OWL-S service descriptions). Even though different efforts have already been done by different research groups to shift existing business processes to SWSs through light weight mapping (as discussed in Section 6.2) but to the best of my knowledge none of these efforts have supported the translation of BPEL process descriptions to complete OWL-S suite of ontologies. Another uniqueness of my approach is that it supports not only the mapping of BPEL process model to OWL-S composite service (with *Profile*, *Process Model* and *Grounding* ontologies) but also maps individual Web services operations to OWL-S *atomic* processes with *Profile*, *Process Model* and *Grounding* ontologies.

The proposed mapping strategy is supported by implementing a tool that can be used to map BPEL processes to OWL-S services. OWL-S service, as a result of mapping process can be dynamically discovered by other semantic enabled systems as well as executed by execution engines (e.g. OWL-S API).

8.2 Application Areas

The ultimate use of business processes (e.g. BPEL processes that are modeled as syntax based compositions of WSDL services) is to export them as WSDL services that can be used by other business partners to perform a joint functionality. As long as Web services are being enhanced with semantics to meet demands of dynamism in rapidly growing e-business world it is becoming more and more tough for business processes to survive in such a dynamic world with their syntactical imitations. Also, pre-defined agreements and collaboration between organizations slows down the process of business collaboration and integration of business services in a distributed environment.

To meet these challenges large business organizations are working to enhance their business process descriptions with semantics so that these business processes can be dynamically discovered, invoked and composed by other semantic enabled systems making the process of business services integration more easy and automotive. A big herder to solve the problem is that it is very cost effective and time consuming task to model business processes in a SWS language (e.g. OWL-S) as semantic based compositions of services and which themselves expose semantically enriched interfaces for dynamic integration and composition with partner services. The approach presented in this thesis and its prototypical implementation provides an efficient and easy solution to enable business processes with semantics by translating existing BPEL processes to OWL-S services.

Another application aspect of proposed approach is that with emerging benefits of semantic enhancements in Web services, services are being made available with semantically enriched service descriptions (e.g. OWL-S composite services). Compositions of these semantically enriched services can not be modeled in a syntactical environment (e.g.

MS BizTalk Server) and needs some semantic based environment (e.g. OWL-S Editor). Also, different approaches (as discussed in Chapter 3) can be used to integrate and compose business processes as SWSs in dynamic and automated fashion (even though I have highlighted some issues that still need to be addressed in this regard).

SWS and business process modeling communities can further work in this direction to establish correspondence between syntax based and semantic based compositions of Web services in bidirectional way (i.e. expressing business processes as SWSs and composite services as business processes). Establishing a bidirectional correspondence between business processes and SWSs can help to avoid the overhead of dynamic and automotive implementation algorithms and techniques when business goals and required services are priory known. In Section 8.4 I discuss such application areas as future work.

8.3 Contributions of This Thesis

The contributions of this thesis are as follows:

- First of all a new 4-tier architecture has been presented to meet integration and composition requirements for integration of business processes as semantically enriched Web services. The proposed architecture addresses different semantic based composition issues (e.g. semantic based Web services interfaces, bridging semantic gap between different integration layers, semantic based UDDI query mechanism etc.). The proposed 4-tier architecture has been discussed in my work **[25, 76]**. On the basis of 4-tier architecture I proposed a SWS integration and composition life cycle **[24]** and a general framework at an abstract level for dynamic and automated composition of business process as SWSs **[23]**. The composition framework follows the approach of newly proposed 4-tier SWS integration and composition architecture and SWS integration and composition life cycle. These architectural and theoretical concepts have been discussed in **Chapter** 3 in detail.
- Second, mapping constraints on the basis of matching functional characteristics of BPEL activities and OWL-S CCs have been discussed in Chapter 4. Process modeling and semantic capabilities of BPEL process model and OWL-S suite of ontologies have been analyzed in detail and mapping constrains have been defined to establish correspondence between BPEL processes and OWL-S services. Mapping constrains also addresses translation issues very well for activities which have dual behavior with in BPEL process model.
- Third, mapping specifications and algorithms have been discussed in Chapter 5 to translate BPEL process descriptions to OWL-S suite of ontologies. Mapping specifications show that how OWL-S suite of ontologies (i.e. *Profile, Process Model* and *Grounding* ontologies) can be extracted from BPEL process model. It also aims at describing that how control flow and data flow can be defined between child processes with in mapped OWL-S composite service. Mapping algorithms show that how efficiently different BPEL activities can be mapped to OWL-S CCs.
- Fourth, I have developed a tool (i.e. BPEL4WS 2 OWL-S Mapping Tool) as an implementation of the proposed approach that can be used to bridge the semantic

gap between business processes and SWSs. BPEL4WS 2 OWL-S Mapping Tool can be used to map BPEL processes to complete OWL-S suite of ontologies. I have also explored (as discussed in Section 1.4) and criticized some initial work done by other research groups in this area. In my work [**20, 22, 21**] I have pointed out limitations and drawbacks of previous work done by other research groups in this area and have shown that how my work provide a more consistent and efficient solution of prescribed problem. **Chapter** 6 discusses the implementation and architecture of the tool in detail.

- Fifth, in **Chapter** 7, I have provided an evaluation of proposed approach and its prototypical implementation. In this chapter 7 I describe that how the approach presented in this thesis addresses syntactical limitations of process modeling language (i.e. BPEL) that have pointed out in Section 1.2 and enable existing business processes for semantic based composition editing, modeling and for dynamic discovery, invocation and composition on the basis of matching semantics. In Chapter 8 I point out some limitations and give future directions to make this work more useful for SWS and process modeling communities.

8.4 Open Issues and Future Work

While describing my approach to bridge the semantic gap between business processes and SWSs, I have presented architectural and theoretical concepts, a strategy to map existing business processes to SWSs and its prototypical implementation. During the previous chapters where I have described that how my work distinguishes from other approaches presented in this area by other research groups, I have also pointed out some issues that I have partially addressed in the proposed solution or still need to be solved.

Here I provide a list of open issues that may be addressed in future to make this work more consistent and efficient solution for prescribed problem.

- Semantically enriched registries that can be used to publish and query for semantically enriched services.
- A more clear picture of SWS integration life cycle and framework for SWS composition and its implementation by extending the proposed strategy and its implementation tool.
- More consistent mapping specifications with upcoming versions of OWL-S.
- Support for mapping multiple condition statements to SWRL expressions.
- Algorithms to parse BPEL and WSDL files and to map them to OWL-S more efficiently.
- Extracting multiple *Profile* ontologies for one *Process Model* ontology.
- Synchronization between process components.
- Providing object view of mapped OWL-S *atomic* and *composite* processes

- Extending the mapping tool to import domain ontologies and to annotate mapped OWL-S service with these domain ontologies.

- Evaluating the tool with more complex business process scenarios to make it efficiently usable in large business organizations.

- Implementing the tool as a BPEL4WS 2 OWL-S Import Plugin for SWS development tool (i.e. OWL-S Editor).

- Bidirectional correspondence between business process and SWSs (i.e. translating business processes to SWSs and vice versa, according to situations and requirements of end user).

Regarding future work, it will be beneficial to perform more consistent mapping by addressing limitations that I described in Chapter 5 and above discussed open issues with upcoming OWL-S specifications. Also, making the implemented tool a part of some larger framework like Protégé can make the proposed work more useful for end user. Such an effort will allow to directly import BPEL processes as OWL-S services in Protégé (with its OWL-S Editor plugin). It will also become easier for end user to develop domain ontologies and to annotate the *Profile* ontology parameters with domain concepts while working in the same environment. Hence, I am working on making the tool part of Protégé as *BPEL4WS 2 OWL-S Import Plugin* for Protégé OWL-S Editor.

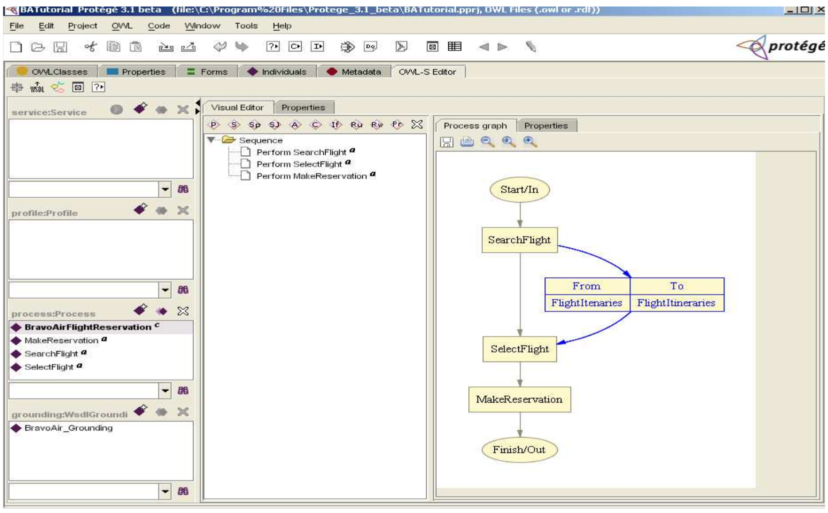

Figure 8.1: An overview of SWSs development tool (Protégé (OWL-S Editor)).

An overview of my ongoing work (future work) is to provide more concrete and practial approach for semantic based discovery, invocation and composition of business processes as SWSs. Especially providing the practical implementation of the SWS integration and composition framework that is discussed in Chapter 3.

During my PhD work I had discussions with many other research groups from the same area and on mailing list, SWS community has appreciated my idea of improvement of mapping tool as BPEL4WS 2 OWL-S Import Plugin for Protégé (OWL-S Editor). BPEL4WS 2 OWL-S Import Plugin will help to easily shift existing business processes from a syntax based to semantic based environment. The plugin will appear as a button in OWL-S Editor tab of Protégé framework. Clicking the *BPEL4WS 2 OWL-S Import Plugin* button will open a wizard which will take as input from the user a BPEL process file, master WSDL file and slave WSDL files (as discussed in Section 6.5 and shown in Figure 6.3) which are part of the process. The BPEL4WS 2 OWL-S import wizard will finish with import of BPEL process as OWL-S service (*composite* process with *Profile, Process Model* and *Grounding*) ontologies. Sub processes (*atomic* or *composite*) will also appear in the *process:Process* window of OWL-S Editor. *Service, Profile* and *Grounding* ontologies will appear under *service:Service*, *profile:Profile* and *grounding:WsdlGrounding* windows as shown in Figure 8.1. End user will be able to simply click and edit any of these ontologies. Specially it will become easier for end user to directly import BPEL process as OWL-S ontology and edit the flow of *composite* process in visual environment of OWL-S Editor. I hope in future I will finish this work which will help for easy shifting of business processes from a syntactical to semantic based environment.

Appendix A

BPEL Process Modeled in MS BizTalk Server

```
<?xml version="1.0"?> <process
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:q2="http://www.mindswap.org/2002/services/Translator.wsdl".../">
 <partnerLinks>
  <partnerLink name="To_Translation_Service_Port_1" partnerLinkType="q1:To_Translation_Service_Port_1Type" partnerRole="portRole" />
  <partnerLink name="Dictionary_Ser_Port" partnerLinkType="q1:Dictionary_Ser_PortType" partnerRole="portRole" />
  <partnerLink name="Reverse_Translation_Port" partnerLinkType="q1:Reverse_Translation_PortType" partnerRole="portRole" />
  <partnerLink name="Input_Output_Port" partnerLinkType="q1:Input_Output_PortType_0" myRole="portRole" />
 </partnerLinks>
 <variables>
  <variable name="Input_Message" messageType="q1:__messagetype_LangTranslationPrj_InputStrAndLang" />
  <variable name="Message1_To_Translation_Service" messageType="q2:TranslatorRequest" />
  <variable name="Message1_From_Translation_Service" messageType="q2:TranslatorResponse" />
  <variable name="Message_1_To_Dic_Service" messageType="q3:DictionaryRequest" />
  <variable name="Message_1_From_Dic_Service" messageType="q3:DictionaryResponse" />
 </variables>
 <sequence>
  <receive partnerLink="Input_Output_Port" portType="q1:Input_Output_PortType" operation="Operation_1"
           variable="Input_Message" createInstance="yes" />
  <assign>
   ..........
   <copy>
    <from variable="Input_Message" part="part" query=.....[local-name()='inputStr' and namespace-uri()='']" />
    <to variable="Message1_To_Translation_Service" part="inputString" query="/*[local-name()='string'....." />
   </copy>
   ..........
   <copy>
    <from variable="Input_Message" part="part" query=.....[local-name()='inputLang' and namespace-uri()='']" />
    <to variable="Message1_To_Translation_Service" part="inputLanguage" query="/*[local-name()='string'....." />
   </copy>
   ..........
   <copy>
    <from variable="Input_Message" part="part" query=.....[local-name()='outputLang' and namespace-uri()='']" />
    <to variable="Message1_To_Translation_Service" part="outputLanguage" query="/*[local-name()='string'....." />
   </copy>
  </assign>
  <invoke partnerLink="To_Translation_Service_Port_1" portType="q2:TranslatorPortType" operation="getTranslation"
          inputVariable="Message1_To_Translation_Service" outputVariable="Message1_From_Translation_Service" />
  <assign>
   <copy>
      <from variable="Message1_From_Translation_Service" part="getTranslationResult" />
      <to variable="Message_1_To_Dic_Service" part="inputString" />
    </copy>
  </assign>
```

```
  <invoke partnerLink="Dictionary_Ser_Port" portType="q3:DictionaryPortType" operation="getMeaning"
          inputVariable="Message_1_To_Dic_Service" outputVariable="Message_1_From_Dic_Service" />
  <invoke partnerLink="Output_Port" portType="q1:Output_PortType_1" operation="Operation_1"
          inputVariable="Message_1_From_Dic_Service" />
 </sequence>
</process>
```

Appendix B

Mapped OWL-S Atomic Process

```
<?xml version="1.0" encoding="windows-1252"?>
 <rdf:RDF
  xmlns:profile="http://www.daml.org/services/owl-s/1.1/Profile.owl#"
  xml:base="http://examples.org/DummyURI.owl">
  ..........
  <service:Service rdf:ID="getTranslationService"/>
   <profile:Profile rdf:ID="getTranslationProfile">
    <profile:hasOutput>
      <process:Output rdf:about="#wsdlFileAddress#return">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         http://www.w3.org/2001/XMLSchema#string</process:parameterType>
      </process:Output>
    </profile:hasOutput>
    <profile:hasInput>
      <process:Input rdf:about="#wsdlFileAddress#outputLanguage">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         http://www.w3.org/2001/XMLSchema#string</process:parameterType>
      </process:Input>
    </profile:hasInput>
    <profile:hasInput>
      <process:Input rdf:about="#filewsdlFileAddress#inputString">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         http://www.w3.org/2001/XMLSchema#string</process:parameterType>
      </process:Input>
    </profile:hasInput>
    <profile:hasInput>
      <process:Input rdf:about="#wsdlFileAddress#inputLanguage">
        <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:Profile>
  <process:AtomicProcess rdf:ID="getTranslationProcess">
    <rdfs:label>getTranslationProcess</rdfs:label>
    <process:hasInput rdf:resource="#wsdlFileAddress#outputLanguage"/>
    <process:hasInput rdf:resource="#wsdlFileAddress#inputString"/>
    <process:hasInput rdf:resource="#wsdlFileAddress#inputLanguage"/>
    <process:hasOutput rdf:resource="#wsdlFileAddress#return"/>
  </process:AtomicProcess>
  <grounding:WsdlGrounding rdf:ID="getTranslationGrounding">
    <grounding:hasAtomicProcessGrounding>
      <grounding:WsdlAtomicProcessGrounding rdf:ID="getTranslationAtomicProcessGrounding"/>
    </grounding:hasAtomicProcessGrounding>
  </grounding:WsdlGrounding>
  <grounding:WsdlAtomicProcessGrounding rdf:about="#getTranslationAtomicProcessGrounding">
    <grounding:wsdlInput>
```

```
      <grounding:WsdlInputMessageMap>
        <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         wsdlFileAddress#outputLanguage</grounding:wsdlMessagePart>
        <grounding:owlsParameter rdf:resource="#wsdlFileAddress#outputLanguage"/>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
    <grounding:wsdlInputMessage rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
     wsdlFileAddress#/Translator.wsdl#TranslatorRequest</grounding:wsdlInputMessage>
    <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
        <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         wsdlFileAddress#inputLanguage</grounding:wsdlMessagePart>
        <grounding:owlsParameter rdf:resource="#wsdlFileAddress#inputLanguage"/>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
    <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource="#wsdlFileAddress#inputString"/>
        <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         wsdlFileAddress#inputString</grounding:wsdlMessagePart>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
    <grounding:wsdlOutput>
      <grounding:WsdlOutputMessageMap>
        <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         wsdlFileAddress#return</grounding:wsdlMessagePart>
        <grounding:owlsParameter rdf:resource="#wsdlFileAddress#return"/>
      </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>
    <grounding:wsdlDocument rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
     wsdlFileAddress#/TranslatorService.wsdl</grounding:wsdlDocument>
    <grounding:wsdlOutputMessage rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
     wsdlFileAddress#/Translator.wsdl#TranslatorResponse</grounding:wsdlOutputMessage>
  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

Appendix C

Mapped OWL-S Composite Service

```
<?xml version="1.0" encoding="windows-1252"?> <rdf:RDF
  xmlns:profile="http://www.daml.org/services/owl-s/1.1/Profile.owl#">
  xmlns:process="http://www.daml.org/services/owl-s/1.1/Process.owl#">
  <service:Service rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestService">
    <service:describedBy>
      <process:CompositeProcess rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestProcess"/>
    </service:describedBy>
    <service:presents>
      <profile:Profile rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestProfile"/>
    </service:presents>
    <service:supports>
      <grounding:WsdlGrounding rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestGrounding"/>
    </service:supports>
  </service:Service>
  <profile:Profile rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestProfile">
    <profile:textDescription>This Profile is created by BPEL2OWLS Tool</profile:textDescription>
    <profile:hasInput>
      <process:Input rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#inputStr">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         http://www.w3.org/2001/XMLSchema#string</process:parameterType>
      </process:Input>
    </profile:hasInput>
    <profile:hasInput>
      <process:Input rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#inputLang">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         http://www.w3.org/2001/XMLSchema#string</process:parameterType>
      </process:Input>
    </profile:hasInput>
    <profile:hasInput>
      <process:Input rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#outputLang">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         http://www.w3.org/2001/XMLSchema#string</process:parameterType>
      </process:Input>
    </profile:hasInput>
    <profile:hasOutput>
      <process:Output rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestOutput0">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         http://www.w3.org/2001/XMLSchema#string</process:parameterType>
      </process:Output>
    </profile:hasOutput>
    <rdfs:label>BPEL2OWLS Profile</rdfs:label>
    <service:presentedBy rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestService"/>
  </profile:Profile>
```

```
<process:CompositeProcess rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestProcess">
  <process:composedOf>
    <process:Sequence>
      <process:components>
        <process:ControlConstructList>
          <list:first>
            <process:Perform rdf:about="http://examples.org/DummyURI.owl#getTranslation1"/>
          </list:first>
          <list:rest>
            <process:ControlConstructList>
              <list:first>
                <process:Perform rdf:about="http://examples.org/DummyURI.owl#getMeaning2"/>
              </list:first>
            </process:ControlConstructList>
          </list:rest>
        </process:ControlConstructList>
      </process:components>
    </process:Sequence>
  </process:composedOf>
  <process:hasOutput rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestOutput0"/>
  <process:hasInput rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#inputLang"/>
  <process:hasResult>
    <process:Result>
      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestOutput0"/>
          <process:valueSource>
            <process:ValueOf>
              <process:fromProcess>
                <process:Perform rdf:about="http://examples.org/DummyURI.owl#getMeaning2"/>
              </process:fromProcess>
              <process:theVar rdf:resource="http://examples.org/DummyURI.owl/wsdlFileAddress#return"/>
            </process:ValueOf>
          </process:valueSource>
        </process:OutputBinding>
      </process:withOutput>
    </process:Result>
  </process:hasResult>
  <service:describes rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestService"/>
  <process:hasInput rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#inputStr"/>
  <process:hasInput rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#outputLang"/>
</process:CompositeProcess>
<process:Perform rdf:about="http://examples.org/DummyURI.owl#getTranslation1">
  <process:hasDataFrom>
    <process:InputBinding>
      <process:valueSource>
        <process:ValueOf>
          <process:theVar rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#inputLang"/>
          <process:fromProcess rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#TheParentPerform"/>
        </process:ValueOf>
      </process:valueSource>
      <process:toParam rdf:resource="http://examples.org/DummyURI.owl/wsdlFileAddress#inputLanguage"/>
    </process:InputBinding>
  </process:hasDataFrom>
  <process:process rdf:resource="http://examples.org/DummyURI.owl#getTranslationProcess"/>
  <process:hasDataFrom>
    <process:InputBinding>
      <process:toParam rdf:resource="http://examples.org/DummyURI.owl/wsdlFileAddress#inputString"/>
      <process:valueSource>
        <process:ValueOf>
          <process:fromProcess rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#TheParentPerform"/>
          <process:theVar rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#inputStr"/>
        </process:ValueOf>
      </process:valueSource>
    </process:InputBinding>
  </process:hasDataFrom>
  <process:hasDataFrom>
    <process:InputBinding>
      <process:valueSource>
```

```
          <process:ValueOf>
            <process:theVar rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#outputLang"/>
            <process:fromProcess rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#TheParentPerform"/>
          </process:ValueOf>
        </process:valueSource>
        <process:toParam rdf:resource="http://examples.org/DummyURI.owl/wsdlFileAddress#outputLanguage"/>
      </process:InputBinding>
    </process:hasDataFrom>
  </process:Perform>
  <process:Perform rdf:about="http://examples.org/DummyURI.owl#getMeaning2">
    <process:hasDataFrom>
      <process:InputBinding>
        <process:valueSource>
          <process:ValueOf>
            <process:theVar rdf:resource="http://examples.org/DummyURI.owl/wsdlFileAddress#return"/>
            <process:fromProcess rdf:resource="http://examples.org/DummyURI.owl#getTranslation1"/>
          </process:ValueOf>
        </process:valueSource>
        <process:toParam rdf:resource="http://examples.org/DummyURI.owl/wsdlFileAddress#inputString"/>
      </process:InputBinding>
    </process:hasDataFrom>
    <process:process rdf:resource="http://examples.org/DummyURI.owl#getMeaningProcess"/>
  </process:Perform>
  <grounding:WsdlGrounding rdf:about="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestGrounding">
    <service:supportedBy rdf:resource="http://www.BPEL2OWLS.org/ChangeTestURI.owl#TestService"/>
    <grounding:hasAtomicProcessGrounding rdf:resource="http://examples.org/DummyURI.owl#getTranslationAtomicProcessGrounding"/>
    <grounding:hasAtomicProcessGrounding rdf:resource="http://examples.org/DummyURI.owl#getMeaningAtomicProcessGrounding"/>
  </grounding:WsdlGrounding>
</rdf:RDF>
```

Appendix D

Semantically Enriched and Extended OWL-S Service

```xml
<?xml version="1.0" encoding="windows-1252"?> <rdf:RDF
  xmlns:profile="http://www.daml.org/services/owl-s/1.1/Profile.owl#">
  xmlns:process="http://www.daml.org/services/owl-s/1.1/Process.owl#"
  xmlns:Languages="http://bis.informatik.uni-leipzig.de/LanguageOntology.owl">
  <service:Service rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestService">
    <service:describedBy>
      <process:CompositeProcess rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestProcess"/>
    </service:describedBy>
    <service:presents>
      <profile:Profile rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestProfile"/>
    </service:presents>
    <service:supports>
      <grounding:WsdlGrounding rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestGrounding"/>
    </service:supports>
  </service:Service>
  <profile:Profile rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestProfile">
    <profile:textDescription>This Profile is created by BPEL2OWLS Tool</profile:textDescription>
    <profile:hasInput>
      <process:Input rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#inputStr">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         http://www.w3.org/2001/XMLSchema#string</process:parameterType>
      </process:Input>
    </profile:hasInput>
    <profile:hasInput>
      <process:Input rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#inputLang">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         &Language#SupportedLanguage</process:parameterType>
      </process:Input>
    </profile:hasInput>
    <profile:hasInput>
      <process:Input rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#outputLang">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         &Language#SupportedLanguage</process:parameterType>
      </process:Input>
    </profile:hasInput>
    <profile:hasOutput>
      <process:Output rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestOutput0">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
         http://www.w3.org/2001/XMLSchema#string</process:parameterType>
      </process:Output>
    </profile:hasOutput>
    <rdfs:label>BPEL2OWLS Profile</rdfs:label>
    <service:presentedBy rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestService"/>
```

```xml
</profile:Profile>
<process:CompositeProcess rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestProcess">
  <process:hasOutput rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestOutput0"/>
  <process:hasInput rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#inputLang"/>
  <service:describes rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestService"/>
  <process:hasResult>
    <process:Result>
      <process:withOutput>
        <process:OutputBinding>
          <process:valueSource>
            <process:ValueOf>
              <process:theVar rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#return"/>
              <process:fromProcess>
                <process:Perform rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#getTranslation3"/>
              </process:fromProcess>
            </process:ValueOf>
          </process:valueSource>
          <process:toParam rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestOutput0"/>
        </process:OutputBinding>
      </process:withOutput>
    </process:Result>
  </process:hasResult>
  <process:composedOf>
    <process:Sequence>
      <process:components>
        <process:ControlConstructList>
          <list:first>
            <process:Perform rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#getTranslation1"/>
          </list:first>
          <list:rest>
            <process:ControlConstructList>
              <list:first>
                <process:Perform rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#getMeaning2"/>
              </list:first>
              <list:rest>
                <process:ControlConstructList>
                  <list:rest rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#nil"/>
                  <list:first>
                    <process:Perform rdf:about="http://examples.org/DummyURI.owl#getTranslation3"/>
                  </list:first>
                </process:ControlConstructList>
              </list:rest>
            </process:ControlConstructList>
          </list:rest>
        </process:ControlConstructList>
      </process:components>
    </process:Sequence>
  </process:composedOf>
  <process:hasInput rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#inputStr"/>
  <process:hasInput rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#outputLang"/>
</process:CompositeProcess>
<process:Perform rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#getTranslation1">
  <process:hasDataFrom>
    <process:InputBinding>
      <process:toParam rdf:resource="http://bis.informatik.uni-leipzig.de/getTranslation.owl#outputLanguage"/>
      <process:valueSource>
        <process:ValueOf>
          <process:theVar rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#outputLang"/>
          <process:fromProcess rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#TheParentPerform"/>
        </process:ValueOf>
      </process:valueSource>
    </process:InputBinding>
  </process:hasDataFrom>
  <process:hasDataFrom>
    <process:InputBinding>
      <process:valueSource>
        <process:ValueOf>
          <process:fromProcess rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#TheParentPerform"/>
          <process:theVar rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#inputLang"/>
        </process:ValueOf>
```

```
      </process:valueSource>
      <process:toParam rdf:resource="http://bis.informatik.uni-leipzig.de/getTranslation.owl#inputLanguage"/>
    </process:InputBinding>
  </process:hasDataFrom>
  <process:process rdf:resource="http://bis.informatik.uni-leipzig.de/getTranslation.owl#getTranslationProcess"/>
  <process:hasDataFrom>
    <process:InputBinding>
      <process:valueSource>
        <process:ValueOf>
          <process:fromProcess rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#TheParentPerform"/>
          <process:theVar rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#inputStr"/>
        </process:ValueOf>
      </process:valueSource>
      <process:toParam rdf:resource="http://bis.informatik.uni-leipzig.de/getTranslation.owl#inputString"/>
    </process:InputBinding>
  </process:hasDataFrom>
</process:Perform>
<process:Perform rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#getMeaning2">
  <process:hasDataFrom>
    <process:InputBinding>
      <process:toParam rdf:resource="http://bis.informatik.uni-leipzig.de/getMeaning.owl#inputString"/>
      <process:valueSource>
        <process:ValueOf>
          <process:fromProcess rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#getTranslation1"/>
          <process:theVar rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#return"/>
        </process:ValueOf>
      </process:valueSource>
    </process:InputBinding>
  </process:hasDataFrom>
  <process:process rdf:resource="http://bis.informatik.uni-leipzig.de/getMeaning.owl#getMeaningProcess"/>
</process:Perform>
<process:Perform rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#getTranslation3">
  <process:hasDataFrom>
    <process:InputBinding>
      <process:valueSource>
        <process:ValueOf>
          <process:fromProcess rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#getMeaning2"/>
          <process:theVar rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#return"/>
        </process:ValueOf>
      </process:valueSource>
      <process:toParam rdf:resource="http://bis.informatik.uni-leipzig.de/getTranslation.owl#inputString"/>
    </process:InputBinding>
  </process:hasDataFrom>
  <process:hasDataFrom>
    <process:InputBinding>
      <process:valueSource>
        <process:ValueOf>
          <process:fromProcess rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#TheParentPerform"/>
          <process:theVar rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#inputLang"/>
        </process:ValueOf>
      </process:valueSource>
      <process:toParam rdf:resource="http://bis.informatik.uni-leipzig.de/getTranslation.owl#outputLanguage"/>
    </process:InputBinding>
  </process:hasDataFrom>
  <process:process rdf:resource="http://bis.informatik.uni-leipzig.de/getTranslation.owl#getTranslationProcess"/>
  <process:hasDataFrom>
    <process:InputBinding>
      <process:toParam rdf:resource="http://bis.informatik.uni-leipzig.de/getTranslation.owl#inputLanguage"/>
      <process:valueSource>
        <process:ValueOf>
          <process:fromProcess rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#TheParentPerform"/>
          <process:theVar rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#outputLang"/>
        </process:ValueOf>
      </process:valueSource>
    </process:InputBinding>
  </process:hasDataFrom>
</process:Perform>
<grounding:WsdlGrounding rdf:about="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestGrounding">
```

```
    <service:supportedBy rdf:resource="http://bis.informatik.uni-leipzig.de/GermanToGermanDic.owl#TestService"/>
    <grounding:hasAtomicProcessGrounding rdf:resource="http://bis.informatik.uni-leipzig.de/getTranslation.owl#
            getTranslationAtomicProcessGrounding"/>
    <grounding:hasAtomicProcessGrounding rdf:resource="http://bis.informatik.uni-leipzig.de/getMeaning.owl#
            getMeaningAtomicProcessGrounding"/>
  </grounding:WsdlGrounding>
</rdf:RDF>
```

Bibliography

[1] Com: Component object model technologies. Online: http://www.microsoft.com/com/default.mspx.

[2] Dcom technical overview. Online: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp.

[3] Shop: Automated planning. Online: http://www.cs.umd.edu/projects/shop/description.html.

[4] Universal plug and play (upnp) language. Online: http://upnp.org/.

[5] Web services activity statement. Online: http://www.w3.org/2002/ws/Activity.

[6] Wie schreibt man eigentlich web service. Online: http://www.jeckle.de/webServices/index.html.

[7] Xlang. Online: http://xml.coverpages.org/xlang.html, June 2001.

[8] Common object request broker architecture: Core specification. Online: http://www-lih.univ-lehavre.fr/d̃utot/enseignement/CORBA/CORBA3_spec.pdf, March 2004.

[9] Jena-a semantic web framework for java. Online: http://jena.sourceforge.net/index.html, 2004.

[10] The owl-s editor. Online: http://owlseditor.semwebcentral.org/, 2004.

[11] Owl web ontology language overview. Online: http://www.w3.org/TR/owl-features/, February 2004.

[12] Protégé: Ontology editor and knowledge-base framework. Online: http://protege.stanford.edu/, September 2006.

[13] Susie Adams, Clifford R. Cannon, Dilip Hardas, Cuneyt Havlioglu, Akhtar Hossein, Charles Kaiman, Tom Lake, Bill Martschenko, Rand Morimoto, Robert Oikawa, Rick Pearson, Kevin Price, Stephen Tranchida, and Larrry Wall. *BizTalk Unleashed*. Sams, Feburary 2002.

[14] Rohit Aggarwal, Kunal Verma, John Miller, and Willie Milnor. Dynamic web service composition in meteor-s. Technical report, LSDIS Lab, Computer Science Department, University of Georgia, USA, 2004.

[15] Rama Akkiraju, Joel Farell, John Miller, Meenakshi Nagarajan, Amit Sheth, and Kunal Verma. Web service semantics - wsdl-s, November 2005.

[16] Rama Akkiraju, Richard Goodwin, Prashant Doshi, and Sascha Roeder. A method for semantically enhancing the service discovery capabilities of UDDI. In Subbarao Kambhampati and Craig A. Knoblock, editors, *IIWeb*, pages 87–92, 2003.

[17] George Anderson and Danielle Larocca. *Sams Teach Yourself SAP in 24 Hours, Second Edition*. Sams, November 2005.

[18] Grigoris Antoniou and Frank van Harmelen. Web ontology language: Owl. Online: http://www.cs.vu.nl/ frankh/postscript/OntoHandbook03OWL.pdf.

[19] Sinuhe Arroyo, Emilia Cimpian, John Domingue, Cristina Feier, Dieter Fensel and-Birgitta König-Ries, Holger Lausen, Axel Polleres, and Michael Stollberg. Web service modeling ontology primer. Online: http://www.w3.org/Submission/WSMO-primer/, June 2005.

[20] Muhammad Ahtisham Aslam, Sören Auer, and Jun Shen. From bpel4ws process model to full owl-s ontology. In *Proceedings of Posters and Demos 3rd European Semantic Web Conference (ESWC 2006)*, pages 61–62, Budva, Montenegro, June 2006.

[21] Muhammad Ahtisham Aslam, Sören Auer, and Jun Shen. Bridging the semantic gap between business processes and semantic web services. *Journal of Internet Technologies*, Under review process, 2007.

[22] Muhammad Ahtisham Aslam, Sören Auer, Jun Shen, and Michael Herrmann. Expressing business process model as owl-s ontologies. In *Proceedings of the 2nd International Workshop on Grid and Peer-to-Peer based Workflows (GPWW 2006) in conjunction with the 4th International Conference on Business Process Management (BPM 2006)*, pages 400–415, Vienna, Austria, September 2006.

[23] Muhammad Ahtisham Aslam, Sören Auer, Jun Shen, and Michael Herrmann. Web services composition to facilitate grid and distributed computing: Current approaches and future framework. In *Proceedings of 4th International Workshop on Frontiers of Information Technology (FIT 2006)*, 2006.

[24] Muhammad Ahtisham Aslam, Sören Auer, Jun Shen, and Michael Herrmann. An integration life cycle for semantic web services composition. In *Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD 07)*, April 2007.

[25] Muhammad Ahtisham Aslam, Michael Herrmann, Sören Auer, and Richard Golden. Real-life soa eexperiences and an approach towards semantic soa. In *Proceedings of 4th International Workshop on SOA and Web Services in conjunction with*

ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2006), pages 72–81, Portland, Oregon, USA, October 2006.

[26] Roland Barcia, Bill Hines, Tom Alcott, and Keys Botzum. *IBM WebSphere: Deployment and Advanced Configuration*. IBM Press, August 2004.

[27] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform resource identifier (uri): Generic syntax. Online: http://gbiv.com/protocols/uri/rfc/rfc3986.html, January 2005.

[28] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[29] Kalina Bontcheva. Project description. Technical report, EU-IST Project IST-2004-026460 TAO (TAO: Transitioning Applications to Ontologies), University of Sheffield, August 2006.

[30] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. Online: http://www.w3.org/TR/ws-arch/, February 2004.

[31] David Booth and Canyang Kevin Liu. Web services description language (WSDL) version 2.0 part 0: Primer. World Wide Web Consortium, Candidate Recommendation CR-wsdl20-primer-20060327, March 2006.

[32] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible markup language (xml) 1.0 (fourth edition). Online: http://www.w3.org/TR/2006/REC-xml-20060816/, September 2006.

[33] Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C recommendation, W3C, Feburary 2004. Online: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[34] Mark Burstein, Christoph Bussler, Michal Zaremba, Tim Finin, Michael N. Huhns, Massimo Paolucci, Amit P. Sheth, and Stuart Williams. A semantic web services architecture. *IEEE Internet Computing*, 9(5):72–81, 2005.

[35] Christoph Bussler. *B2B Integration: Concepts and Architecture*. Springer, 2003.

[36] Christoph Bussler, Emilia Cimpian, Dieter Fensel, Juan Miguel Gomez, Armin Haller, Thomas Haselwanter, Michael Kerrigan, Adrian Mocan, Matthew Moran, Eyal Oren, Brahmananda Sapkota, Ioan Toma, Jana Viskova, Tomas Vitvar, Maciej Zaremba, and Michal Zaremba. Web service execution environment (wsmx). Online: http://www.w3.org/Submission/WSMX/, June 2005.

[37] Jorge Cardoso and Amit P. Sheth. Introduction to semantic web services and web process composition. In Jorge Cardoso and Amit P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2004.

[38] Chen-Burger, Yun-Heh, and Dave Robertson. Mapping a Business Process Model to a Web Services Model. In Li Guo, editor, *Third IEEE International Conference on Web Services 2004*, pages 746–749, July 2004.

[39] Yun-Heh Chen-Burger, Austin Tate, and Dave Robertson. Enterprise modelling: A declarative approach for fbpml. In *Proceedings of European Conference of Artificial Intelligence, Knowledge Management and Organisational Memories Workshop*, Lyon, France, 2002.

[40] James Clark. XSL transformations (XSLT) version 1.1. World Wide Web Consortium, Working Draft WD-xslt11-20010824, August 2001.

[41] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. UDDI version 3.0.2. Organization for the Advancement of Structured Information Standards, UDDI Spec Technical Committee Draft, October 2004.

[42] Francisco Curbera, Hitesh Dholakia, Yaron Goland Bea, Johannes Klein Microsoft, Frank Leymann Ibm, Kevin Liu Sap, Dieter Roller Ibm, Doug Smith, Siebel Systems, Satish Thatte, Ivana Trickovic Sap, and Sanjiva Weerawarana Ibm. Business process execution language for web services, May 2003.

[43] Francisco (Paco) Curbera, Matthew J. Duftler, Rania Khalaf, Nirmal Mukhi, William A. Nagy, and Sanjiva Weerawarana. Bpws4j: A platform for creating and executing bpel4ws processes. Online: http://www.alphaworks.ibm.com/tech/bpws4j, April 2004.

[44] Jos de Bruijn, Holger Lausen, Axel Polleres, and Dieter Fensel. The web service modeling language WSML: An overview. In York Sure and John Domingue, editors, *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, pages 590–604. Springer, 2006.

[45] Yannis Dimopoulos and Pavlos Moraitis. Multi-agent coordination and cooperation through classical planning. In *IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 398–402, Washington, DC, USA, 2006. IEEE Computer Society.

[46] John Domingue, Liliana Cabral, Farshad Hakimpour, Denilson Sell, and Enrico Motta. Irs-iii. a platform and infrastructure for creating wsmo-based semantic web services. In *Proceedings of the Workshop on WSMO Implementations (WIW 2004)*, Frankfurt, Germany, 29-30 September 2004.

[47] Daniel Elenius, Grit Denker, David Martin, Fred Gilham, John Khouri, Shahin Sadaati, and Rukman Senanayake. The OWL-S editor - A development tool for semantic web services. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2005.

[48] Daniel Elenius, Grit Denker, David Martin, Fred Gilham, John Khouri, Shahin Sadaati, and Rukman Senanayake. The owl-s editor- a development tool for semantic web services. In *Proceedings of 2nd European Semantic Web Conference (ESWC 05)*, pages 78–92, Heraklion, Crete, Greece, May/June 2005.

[49] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall, August 2005.

[50] Joel Farrell and Holger Lausen. Semantic annotations for wsdl, 2006.

[51] Cristina Feier, Dumitru Roman, Axel Polleres, John Domingue, Michael Stollberg, and Dieter Fensel. Towards intelligent web services: Web service modeling ontology (wsmo). In *Proceeding of the International Conference on Intelligent Computing (ICIC)*, Hefei, China, August 2005.

[52] Ferdian. A comparison of event-driven process chains and uml activity diagram for denoting business processes. Master's thesis.

[53] Matthias Flügge and Diana Tourtchaninova. Ontology-derived activity components for composing travel web services. In Robert Tolksdorf and Rainer Eckstein, editors, *Berliner XML Tage*, pages 133–150. XML-Clearinghouse, 2004.

[54] Nadarajan Gayathri and Yun-Heh Chen-Burger. Translating fundamental business process modelling language to the web services ontology through lightweight mapping. In *under review process*, August 2006.

[55] Gennari, John H., Musen, Mark A., Fergerson, Ray W., Grosso, William E., Monica Crubezy, Henrik Eriksson, Noy, Natalya F., Tu, and Samson W. The evolution of protege: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[56] Beth Gold-Bernstein and William Ruh. *Enterprise Integration: The Essential Guide to Integration Solutions.* Addison Wesley Professional, 2004.

[57] Colin Gray. Entrepreneurship, resistance to change and growth in small firms. *Journal of Small Business and Enterprise Devel-opment*, 9(1462-6004):61–72, March 2002.

[58] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. SOAP version 1.2 part 1: Messaging framework. World Wide Web Consortium, Recommendation REC-soap12-part1-20030624, June 2003.

[59] Mike Havey. *Essential Business Process Modeling.* O'Reilly, August 2005.

[60] Horridge, Matthew, Knublauch, Holger, Rector, Alan, Stevens, Robert, Wroe, and Chris. A practical guide to building owl ontologies using the protege-owl plugin and co-ode tools edition 1.0. August 2004.

[61] IBM. Web services architecture overview. Online: http://www-128.ibm.com/developerworks/library/w-ovr, September 2000.

[62] Matjaz Juric and Benny Mathew Poornachandra Sarang. *Business Process Execution Language for Web Services: A Practical Guide to Orchestrating Web Services Using BPEL4WS*. PACKT Publishing, October 2004.

[63] Uwe Keller, Ruben Lara, Holger Lausen, Axel Polleres, and Dieter Fensel. Automatic location of services. In *Proceedings of Second European Semantic Web Conference (ESWC 2005)*, pages 1–16, Heraklion, Crete, Greece, May/Jun 2005.

[64] Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210, Feburary 2004.

[65] Holger Knublauch, Mark A. Musen, and Alan L. Rector. Editing description logic ontologies with the protégé OWL plugin. In Volker Haarslev and Ralf Möller, editors, *Description Logics*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[66] Ulrich Küster, Birgitta König-Ries, Mirco Stern, and Michael Klein. Diane: an integrated approach to automated service discovery, matchmaking and composition. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1033–1042, New York, NY, USA, 2007. ACM Press.

[67] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: a logic programming language for dynamic domains. *Journal of Logic Programming*, 1997.

[68] Frank Leymann. Web services flow language (wsfl 1.0), May 2001.

[69] Mohamrnad A. Makhzan and Kwei-Jay Lin. Solutions to a complete web service discovery and composition. In *CEC/EEE*, page 73. IEEE Computer Society, 2006.

[70] Daniel J. Mandell and Sheila A. McIlraith. Adapting BPEL4WS for the semantic web: The bottom-up approach to web service interoperation. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2003.

[71] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila Mcllraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen. Srinivasan, and Katia Sycara. Owl-s: Semantic markup for web services. Online: http://www.ai.sri.com/daml/services/owl-s/1.2/overview/, March 2006.

[72] David L. Martin, Massimo Paolucci, Sheila A. McIlraith, Mark H. Burstein, Drew V. McDermott, Deborah L. McGuinness, Bijan Parsia, Terry R. Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia P. Sycara. Bringing semantics to web services: The OWL-S approach. In Jorge Cardoso and Amit P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 26–42. Springer, 2004.

[73] Deborah L. McGuinness and Frank van Harmelen. OWL web ontology language overview. World Wide Web Consortium, Recommendation REC-owl-features-20040210, Feburary 2004.

[74] Jan Mendling, Gustaf Neumann, and Markus Nüttgens. Yet another event-driven process chain. In *Proceedings of 3rd International Conference on Business Process Management (BPM 2005)*, volume LNCS 3649, pages 428–433, Nacy, France, Septemper 2005.

[75] Harald Meyer, Hagen Overdick, and Mathias Weske. Plængine: A system for automated service composition and process en-actment. In *Proceedings of WWW Service Composition with Se-mantic Web Services*, pages 3–12, University of Technology of Compiègne.

[76] Herrman Michael and Muhammad Ahtisham Aslam. Mercedes car group (mcg) enterprise architektur: Ein ansatz zur semantischen modellierung der services in einer soa. In *Integration betrieblicher Informationssysteme: Problemanalysen und Lösungsansätze des Model-Driven Integration Engineering, Leipziger Beiträge zur Informatik: Band IV. Leipzig*, pages 145–151, September 2006.

[77] John Miller, Kunal Verma, Preeda Rajasekaran, Amit Sheth, Rohit Aggarwal, and Kaarthik Sivashanmugam. Wsdl-s: Adding semantcis to wsdl. white paper, LSDIS Lab, University of Georgia, Georgia, USA, July 2004.

[78] Gayathri Nadarajan and Yun-Heh Chen-Burger. An ontology-based conceptual mapping framework for translating fbpml to the web services ontology. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 158–165, Washington, DC, USA, 2006.

[79] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Importing the semantic web in uddi. In *Proceedings of E-Services and the Semantic Web Workshop*, pages 225–236, 2002.

[80] Massimo Paolucci, Naveen Srinivasan, Katia P. Sycara, and Takuya Nishimura. Towards a semantic choreography of web services: From WSDL to DAML-S. In Liang-Jie Zhang, editor, *IWCS*, pages 22–26. CSREA Press, 2003.

[81] Peter F. Patel-Schneider. Requirements and non-requirements for a semantic web rule language. In *Rule Languages for Interoperability.* W3C, 2005.

[82] Terry R. Payne, Nuria Sanchez, and Domenico Redavid. Semantic web services bootstrapping meghodology. Technical report, EU-IST Project IST-2004-026460 TAO (TAO: Transitioning Applications to Ontologies), August 2006.

[83] Shankar R. Ponnekanti and Armando Fox. SWORD: A developer toolkit for web service composition. January 2002.

[84] Preeda Rajasekaran, John A. Miller, Kunal Verma, and Amit P. Sheth. Enhancing web services description and discovery to facilitate composition. In Jorge Cardoso

and Amit P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 55–68. Springer, 2004.

[85] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In Jorge Cardoso and Amit P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2004.

[86] Denilson Sell, Farshad Hakimpour, John Domingue, Enrico Motta, and Roberto C. S. Pacheco. Interactive composition of wsmo-based semantic web services in irs-iii. In *In proceedings of the First AKT Workshop on Semantic Web Services (AKT-SWS04) KMi*, The Open University, Milton Keynes, UK, December 2004.

[87] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[88] Jun Shen, Georg Grossmann, Yun Yang, Markus Stumptner, Michael Schrefl, and Thomas Reiter. Analysis of business process integration in web service context. In *FGCS: The International Journal of Grid Computing: Theory, Models and Applications*, number ISSN:0167-739X. Elsevier Publishers, May 2006.

[89] Jun Shen, Jun Yan, and Yun Yang. Swindew-s: Extending p2p workflow systems for adaptive composite web services. In *Proceedings of Australian Software Engineering Conference (ASWEC 2006)*, pages 61–69, Sydney, Australia, April 2006.

[90] Jun Shen, Yun Yang, and Bharat Lalwani. Mapping web services specifications to process ontology: Opportunities and limitations. In *Proceedings of the 10th International Workshop on Future Trends of Distributed Computing Systems (FTDCS 04)*, pages 229–235, Suzhou, China, May 2004.

[91] Jun Shen, Yun Yang, and Quang Huy Vu. Swindew-b: A p2p based composite service execution system with bpel. In *Proceedings of 3rd International Conference on Service Oriented Computing (ICSOC) Workshop on Dynamic Web Processes (DWP 2005), post-proceedings as IBM RC23822*, pages 73–84, Amsterdam, Netherlands, December 2005.

[92] Jun Shen, Yun Yang, Chengang Wan, and Chuan Zhu. From bpel4ws to owl-s: Integrating e-business process descriptions. In *International Conference on Services Computing (SCC 2005)*, pages 181–188, Orlando, FL, USA, July 2005.

[93] Jun Shen, Yun Yang, Chengang Wan, and Chuan Zhu. From BPEL4WS to OWL-S: Integrating E-business process descriptions. In *IEEE SCC*, pages 181–190. IEEE Computer Society, 2005.

[94] Jun Shen, Yun Yang, and Jun Yan. Adapting p2p based decentralised workflow system swindew-s with web service profile support. In *Proceedings of 9th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2005)*, pages 535–540, Coventry, UK, May 2005.

[95] Payam Shodjai. Web services and the microsoft platform. Online: http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/wsmsplatform.asp, June 2006.

[96] Evren Sirin. Owl-s api. Online: http://www.mindswap.org/2004/owl-s/api/, August 2001.

[97] Evren Sirin. Using web ontologies for web service composition. Online: http://www.mindswap.org/ evren/docs/WebServices.pdf, June 2004.

[98] Evren Sirin, James A. Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In Jean Bézivin, Jiankun Hu, and Zahir Tari, editors, *WSMAI*, pages 17–24. ICEIS Press, 2003.

[99] Evren Sirin and Bijan Parsia. The owl-s java api. In Third International Semantic Web Conference (ISWC2004), November 2004.

[100] Evren Sirin, Bijan Parsia, and James Hendler. Template-based composition of semantic web services. In *AAAI Fall Symposium on Agents and the Semantic Web*, Virginia, USA, November 2005.

[101] Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, and Dana S. Nau. HTN planning for web service composition using SHOP2. *J. Web Sem*, 1(4):377–396, 2004.

[102] Kaarthik Sivashanmugam, Kunal Verma, Amit Sheth, and John Miller. Adding semantics to web services standards. In *Proceedings of the 1st International Conference on Web Services (ISWC'03)*, pages 395–401, Las Vegas, Nevada, June 2003.

[103] Pervasive Software. Pervasive integration architecture. white paper, Pervasive Software, 2005.

[104] Kilian Stillhard and Erik Wilde. XML schema compact syntax (XSCS) version 1.0. Technical Report TIK Report No. 166, Computer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland, March 2003.

[105] Michael Stollberg, Uwe Keller, and Dieter Fensel. Partner and service discovery for collaboration establishment with semantic web services. In *ICWS*, pages 473–480. IEEE Computer Society, 2005.

[106] Andreas Terzis, Jun Ogawa, Sonia Tsui, Lan Wang, and Lixia Zhang. A prototype implementation of the two-tier architecture for differentiated services. In *In RTAS99*, Vancouver, Canada, 1999.

[107] Carsten Ullrich. Course generation based on HTN planning. In Mathias Bauer, Boris Brandherm, Johannes Fürnkranz, Gunter Grieser, Andreas Hotho, Andreas Jedlitschka, and Alexander Kröner, editors, *LWA*, pages 74–79. DFKI, 2005.

[108] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller. Meteor-s wsdi: A scalable infrastructure of registries for semantic publication and discovery of web services. *Journal of Information Technology and Management*, 2004.

[109] Qiang Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach to Classical Planning.* Springer-Verlag, Berlin, 1997.