

# A Practical Tutorial on Semantic Web Services

Farshad Hakimpour, Suo Cong, Daniela E. Damm

**Abstract.** This tutorial presents two dominant specifications in domain of Semantic Web Services domain, namely OWL-S (Web Ontology Language for services) and WSMO (Web Services Modeling Ontology). It briefly introduces Web Services and Semantic Web, two major specifications underlying the Semantic Web Services technology and then explains most of the key features of this Semantic Web Services together with simplified examples. We discuss three aspects of Semantic Web Services: specifications for semantic descriptions of services, intelligent discovery and selection of services using semantic descriptions, and finally, building more complex services by composing existing ones. The goal is not only to present an abstract view of this technology but also introduction of the technical details of the two existing specifications.

## 1. Introduction

Semantic Web Services technology lays its foundation on both Web Services (W3C, 2004a) and Semantic Web (Berners-Lee et al. 2001; Decker et al., 2000). Web Services offer a promising approach to accomplish a loose coupling of processes across organizational boundaries. Web Services technologies present specifications that cover the details required for an automated interoperation among client agents and services on the Web, with a minimum interference of human agents. A Web Service may provide any of the following or their combinations:

- static information, e.g. retrieving geographic or statistical data;
- digital processes, e.g. unit conversion or currency exchange; or
- actual services with concrete effects, e.g. booking a flight or selling a book and shipping it to an address.

On the other hand, Semantic Web offers computer interpretable semantic knowledge to facilitate a smarter selection of services and assists combining them to build composite services or applications. Such objectives can be achieved by describing the capabilities of a service using semantic descriptions. Programs on the Web will be able to find each other (other Web Services) by matching their requirements with the capabilities of available services. Semantic Web technologies can be applied to describe provided capabilities and/or desired requirements of a service.

We believe Semantic Web Services technology will improve and facilitate discovery, composition and interaction with Web Services. Semantic Web Services facilitates the process of composing several Web Services to build a more complex service, while it exposes and behaves as one single service to a client agent. That includes both aspects of facilitating automatic service composition as well as

providing specification to describe a composition. The interaction with Web Services not only considers invocation and brokering, but would often follow a specific message interchange protocol. Semantic Web Services technology provides specifications for Web Services to describe their interaction pattern. Description of interaction patterns can be used by client agents during the discovery as well as the execution time.

Our main objective is to introduce the emerging technology of Semantic Web Services. As we introduce this technology, we also discuss the two dominant specifications in this domain, namely OWL-S (Web Ontology Language for services; Martin et al., 2004a) and WSMO (Web Services Modeling Ontology; WSMO, 2004a). We present all essential features of these specifications and provide simplified examples.

This tutorial is organized as following. In the next two sections we give an overview of Web Services technologies, Semantic Web. Section 2 provides the background knowledge for Web Services. Section 3 briefly introduces Semantic Web, the notion of ontology and the Web Ontology Language (OWL) specification. These two sections give the necessary basic understanding of these technologies that is required for the remainder of this paper.

Section 4 motivates the augmentation of Web Services by adding semantics. In addition, it briefly introduces the two major specification in this domain OWL-S and WSMO. We use these specifications to present our example descriptions through the paper. In section 5, we explain how the semantics of Web Services are described by their functional and non-functional properties; also how semantic descriptions are bound to the service descriptions. We discuss intelligent service discovery as a one of the motivation of Semantic Web Service technology in section 6. Section 7 introduces topics relevant to service composition modeling. We explain OWL-S composition model and briefly introduce WSMO Orchestration and Choreography. In section 8, we present a summary and discuss the types of the tools needed for applying Semantic Web Services technology.

In the following sections, concepts or relations specific to OWL-S are denoted by Arial narrow and those of WSMO are capitalized. In figures, rounded corner boxes illustrate concepts in OWL-S and boxes show concepts in WSMO. The arrows in the diagrams ( $\rightarrow$ ) show relations between concepts with the arrows pointing to the range of the relation. The name of the relation and the cardinality (if relevant) appear next to the arrow head. The solid triangular ( $\rightarrow$ ) arrows show the specialization relation with the arrow pointing to the superclass.

## 2. Web Services

A Web Service is a software program that exposes a coherent functionality via an interface described in a machine-processable format (e.g. WSDL) and supports interoperable machine-to-machine interactions with other programs via XML-based messages (e.g. SOAP) conveyed using Web-related standards (W3C, 2004a).

A primary contribution of Web Services toward conquering the limitations of conventional middleware and proprietary EAI/EDI infrastructures is to enforce

standardization in defining, describing, and discovering services. Such standardization should support interactions with other programs in a peer-to-peer fashion based on middleware protocols within the service-oriented paradigm leading to a design strategy that everything could be exposed and used as a service (Alonso, 2004). The most distinct feature of using Web Services is the designed machine-interpretability supporting a Web Service discovery and invocation by other software systems, and consequently interaction with the service. The foundation of the machine-interpretable Web Services is to express the knowledge required for properly interacting with a Web Service in a format that can be processed automatically by any service requester. A requester analyzes a service description to determine whether a Web Service is qualified for fulfilling a given request and to acquire the details of how to use a Web Service. The rest of this section contains a brief background to Web Services and their three main pillars, namely SOAP, WSDL, and UDDI technologies.

## **2.1. SOAP**

The Simple Object Access Protocol (SOAP) is a specification for interactions among Web Services across the Internet. SOAP uses XML to exchange structured and typed information. It defines bindings to actual transport protocols such as HTTP or SMTP (W3C, 2003). Most software vendors support SOAP as the common specification for interacting with a Web Service.

What makes SOAP different from prior technologies, such as CORBA/ORBs or Java RMI, are few following characteristics (W3C, 2003):

- SOAP is an XML-based protocol. Instead of passing objects of complicated structure, which may vary in different implementations, SOAP employs a simple messaging approach. Packaged XML messages are passed between the interacting applications. This makes it easier to achieve common standards among different vendors. Furthermore, SOAP message processors can easily use an underlying XML processor.
- SOAP extensively leverages the HTTP protocol. SOAP can use the HTTP protocol (the protocol underlying the World Wide Web) as its transport protocol. In other words, an HTTP server (i.e. Web server) can recognize a request containing a SOAP message and pass it to a SOAP processor or take the necessary action. However, SOAP is a neutral messaging protocol and does not rely on any underlying protocol, including HTTP, and it can use any other transport protocol.
- SOAP is about messaging. SOAP messages can carry the application semantics. It is neutral towards representation of application semantics. Therefore, it leads to an infrastructure of interoperability and extensibility.
- SOAP is a W3C recommendation and not a vendor dependent messaging protocol, unlike messaging underlying different CORBA/ORB vendor implementations or Java RMI.

SOAP defines a simple messaging framework to transfer XML messages between an initial sender and an ultimate receiver. For a successful interaction

between the sender and the receiver, the receiver must understand how the sender encodes the message. A particular encoding form is proposed in the SOAP protocol. However, SOAP is not limited to use any specific encoding mechanism.

A SOAP message is composed of two parts: an optional SOAP header element and a mandatory SOAP body element. The SOAP header is used to carry “control” information to indicate how to handle the message such as routing, authentication, and transactions which are not included in the application payload. The SOAP body element carries the actual message to be delivered to the ultimate receiver via any number of intermediaries. Only the ultimate receiver is expected to understand the semantics of the application payload.

SOAP provides two kinds of interactions between the sender and the receiver: RPC-based and Document-based. In RPC-based paradigm, the messages are translated into corresponding RPC method signatures and the result/output parameters. In Document-based paradigm, the interactions are realized by exchanging the documents from one application to another.

## 2.2. WSDL

In principle, a service description is defined to express the information required to invoke a service properly. A service invoker needs to understand the complete description of a Web Service to determine whether it is qualified to fulfill a specific purpose or task and how to use it. A complete description of a Web Service normally involves multiple layers (e.g. in Alonso, 2004). In order to distinguish the concept of complete description with the concept of basic description typically at communication level, we consider a complete service description as a service comprehension that consists of three levels of information of using a service properly: communication level, semantic level, and business level.

At the communication level, each Web Service must have a machine-processable description to specify the necessary information, including schematic information (i.e. message formats and data types) and transport protocols, to enable a client agent to invoke and interact with a service. The dominant specification at this level is the Web Services Description Language (WSDL; W3C, 2005). WSDL documents offer service requesters the potential to discover Web Services autonomously and automatically with reduced human intervention.

The information in WSDL model can be divided in two parts: an abstract part which describes a Web Service in terms of the messages it sends and receives, and a concrete part which specifies the details of how to access a Web Service. We focus here on the abstract part and describe how it can be augmented by semantic descriptions.

### Listing 1. Example source code for a service calculating a freight service cost.

---

```
1 package swsExamples.services;
2 public class Freight {
3     public int cost (String destination,
4                     swsExamples.physicalMeasures.Volume size) {
5     . . .
```

---

---

```

6     }
7   }
8
9   package swsExamples.physicalMeasures;
10  public class Volume {
11      public int value;
12      public String unit;
13  }

```

---

An example of a WSDL description is shown in Listing 2. The WSDL description is automatically generated for the service in Listing 1 and consists of the following elements:

- **Messages:** Client agents communicate to the service through two messages: `costRequest` and `costResponse` (lines 24 to 30).
- **Types:** Complex data types can be also described in the WSDL description. In this example `Volume` is describe in lines 12 to 23.
- **Port Types:** Port types are analogous to the interface definitions. Our Port Type between lines 31 to 36 describes the interface for the `Freight` class.
- **Operations:** A Port Type contains a set of operations. In our example, `Freight` contains only the `cost` operation. Operations are described in terms of the messages that can be used to invoke them.
- **Bindings:** Several bindings can be defined for a Port Type. Bindings define encoding and transport protocol for messages used in the operations of a Port Type. In our example, one binding is defined for `cost` operation, using SOAP.
- **Services:** A Service consists of one or more Port Types and at least one binding for each Port Type (lines 58 to 64). Service also specifies an end point or the location where the operations reside on the Web (lines 62 and 62).

### Listing 2. The WSDL definition for the service in Listing 1.

---

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions
3    targetNamespace="http://www.example.com/axis/WSDL/freight.wsdl"
4    xmlns:apachesoap="http://xml.apache.org/xml-soap"
5    xmlns:impl="http://www.example.com/axis/WSDL/freight.wsdl"
6    xmlns:intf="http://www.example.com/axis/WSDL/freight.wsdl"
7    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
8    xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
9    xmlns:wSDLsoap="http://schemas.xmlsoap.org/wsdl/soap/"
10   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
11  <!--WSDL created by Apache Axis version: 1.2-->
12  <wsdl:types>
13    <schema targetNamespace="http://www.example.com/axis/WSDL/freight.wsdl"
14      xmlns="http://www.w3.org/2001/XMLSchema">
15      <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
16      <complexType name="Volume">
17        <sequence>
18          <element name="value" type="xsd:int" />
19          <element name="unit" nillable="true" type="xsd:string" />
20        </sequence>
21      </complexType>
22    </schema>
23  </wsdl:types>
24  <wsdl:message name="costRequest">

```

---

---

```

25 <wsdl:part name="destination" type="xsd:string"/>
26 <wsdl:part name="size" type="impl:Volume"/>
27 </wsdl:message>
28 <wsdl:message name="costResponse">
29 <wsdl:part name="costReturn" type="xsd:int"/>
30 </wsdl:message>
31 <wsdl:portType name="Freight">
32 <wsdl:operation name="cost" parameterOrder="destination size">
33 <wsdl:input message="impl:costRequest" name="costRequest"/>
34 <wsdl:output message="impl:costResponse" name="costResponse"/>
35 </wsdl:operation>
36 </wsdl:portType>
37 <wsdl:binding name="FreightPortSoapBinding"
38 type="impl:Freight">
39 <wsdlsoap:binding
40 style="rpc"
41 transport="http://schemas.xmlsoap.org/soap/http"/>
42 <wsdl:operation name="cost">
43 <wsdlsoap:operation soapAction="" />
44 <wsdl:input name="costRequest">
45 <wsdlsoap:body
46 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
47 namespace="http://www.example.com/axis/WSDL/freight.wsdl"
48 use="encoded"/>
49 </wsdl:input>
50 <wsdl:output name="costResponse">
51 <wsdlsoap:body
52 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
53 namespace="http://www.example.com/axis/WSDL/freight.wsdl"
54 use="encoded"/>
55 </wsdl:output>
56 </wsdl:operation>
57 </wsdl:binding>
58 <wsdl:service name="FreightService">
59 <wsdl:port binding="impl:FreightPortSoapBinding"
60 name="FreightPort">
61 <wsdlsoap:address
62 location="http://www.example.com/axis/services/freight"/>
63 </wsdl:port>
64 </wsdl:service>
65 </wsdl:definitions>

```

---

One of the topics covered in the rest of this paper is how the WSDL descriptions are enhanced by semantics. In other words, we show how Semantic Web Service technology can help us to describe semantics of data types and operations for Web Services. We enhance Messages, Types, PortType and Operation elements of the WSDL description in Listing 2 as we progress in the tutorial.

### 2.3. UDDI

At present, there are two major types of approaches to find service descriptions: search-oriented and storage-oriented. The search-oriented approach employs a crawler to collect description on the Web (Dong et al., 2004; see also <http://www.webservicelist.com>). The storage-oriented approach uses storages to store and organize the service descriptions submitted by service providers actively. The storage of service descriptions can be constructed as a registry, an index, or a peer-to-peer system (W3C, 2004a). To use a service properly, especially in mission-critical business applications, a trusted business service registry is preferred as it can

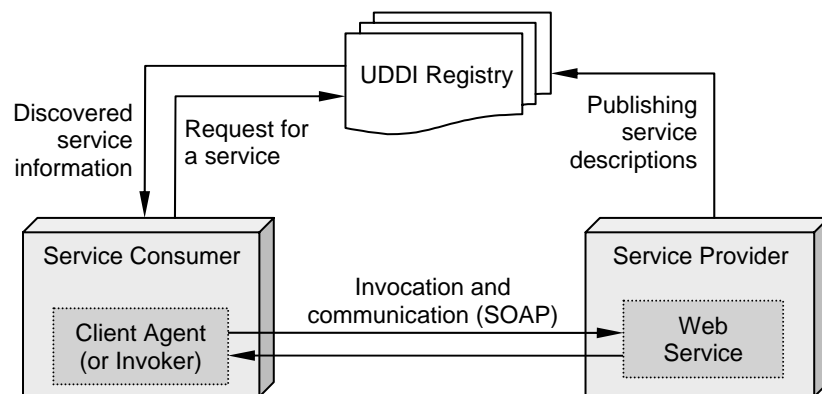
improve the protection of both the service requesters against malicious providers, and the service providers against malicious requesters. One of the most eminent business service registries is the Universal Description, Discovery, and Integration specification (UDDI; OASIS, 2004).

A UDDI registry allows registering of Web Service descriptions for businesses and facilitates their discovery (Figure 1). The core of the UDDI specification consists of a data model defined to represent Web Services as UDDI data, and a collection of *API sets* for manipulating the stored UDDI data. We briefly introduce the UDDI data model as its understanding is necessary in the rest of this paper.

### ***UDDI Data Model***

The canonical UDDI data model defines six major categories of information to represent Web Services. The data model enables the registry to find a qualified Web Service according to a specific request. Each category is defined as an entity expressed in XML.

- **businessEntity**: the description of a business or service provider and the services it provides. It specifies the name of a provider, contact and classification information. **businessEntity** includes service descriptions and technical information, using **businessService** and **bindingTemplate**.
- **businessService**: a logical group of Web Services belonging to a service provider represented by a **businessEntity**. It contains general information of Web Services included in a logical group, such as, names, descriptions and classification information. This structure stands between the level of **businessEntity** and the level of **bindingTemplate** for the sake of assembling a number of services in a logical relationship, for example, services related to travel planning.
- **bindingTemplate**: the necessary technical information to invoke a specified Web Service. Each **bindingTemplate** represents either the access point or a pointer to the access point of an individual Web Service.
- **tModel**: a technical model which can be reused to represent any kind of specification. **tModels** can be used to describe a Web Service classification scheme, a protocol used by a Web Service, or a namespace in a standard way. A



**Figure 1.** Role of UDDI registries and other Web Services Technologies.

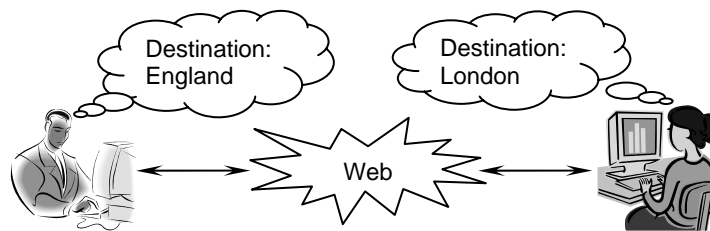
tModel is a ‘technical fingerprint’ used to describe some common characteristics of Web Services.

- publisherAssertion: a relationship between two service providers each represented by a businessEntity.
- subscription: A standing request to receive the notifications of changes of specified UDDI entities.

### 3. Semantic Web

The universal Web provides an immense platform for exchanging and sharing huge amount of data world wide. In order to enable the machines to process the data over the Web automatically and intelligently, a number of relevant metadata should be provided alongside the data. Such metadata can contain *schematic* knowledge, for example in form of XML Schemas (W3C, 2004d) and WSDL (W3C, 2005), or *semantic* knowledge to represent the intended meaning of data.

Inspired by the desire of exchanging machine-processable information, the idea of Semantic Web has been proposed to extend the current Web infrastructure to represent data of well-defined meaning (Berners-Lee, et al. 2001). The Semantic Web initiative is to study the feasible approaches of introducing metadata to describe meanings of Web resources residing at the decentralized Internet. The semantic descriptions are aimed to be interpreted by programs that eventually help users to avoid misinterpretation of available data. Semantic Web technologies assists us in describing terms such as ‘destination’ (in Listing 2) to avoid possible misinterpretation by different people (Figure 2).



**Figure 2.** Various interpretation of a term in a service description.

#### 3.1. Ontologies

Interpretation of terms in data or schematic definitions is usually taken for granted. Ontologies are the means to avoid the misinterpretation of terms by formally describing a conceptualization (Guarino, 1998). They are used to describe the semantics of terms in different domains. As we show in the rest of this tutorial, ontologies play an important role in semantic descriptions of Web Services. Ontologies explicitly describe terms often using logical expressions. Using a formal and logical language, enables computers to process the knowledge encoded in an ontology. Ontologies guarantee that the information in any instance of

communication is consistently interpreted by both involved parties. Using ontologies, communities are able to reduce the risk of misinterpretation while keeping their diversity.

Ontologies may appear in various forms. They can have a simple form of a taxonomy tree that relates terms by specialization and generalization relations; or in a more complicated form, they may use complex logical expressions to describe terms in relation to each other. Formalized definitions of terms in ontologies can be processed by computer programs and help us to improve the reliability of data interpretation.

### 3.2. Web Ontology Language (OWL)

OWL (McGuinness, 2004) defines a formal language for defining ontologies. It is a W3C recommendation for describing ontologies based on other W3C recommendation, RDF and XML. OWL recommendation consists of three sublanguages: OWL-Lite, OWL-DL and OWL-Full. The underlying reason for having different sublanguages is the different levels of details needed in different application (as mentioned in section 3.1).

OWL-Lite is a simple subset of OWL that can be used to describe taxonomy trees. OWL-DL is a more complex language and its expressions can be processed by most Description Logic reasoning systems. OWL-Full is the complete expressive language that offers maximum expressivity but might not be fully processed by the existing systems, due to the complexity of the language.

#### Listing 3. Part of a simplified OWL ontology for quantities describing “volume” (used in our example).

---

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
4   xmlns:owl="http://www.w3.org/2002/07/owl#"
5   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
6   xmlns="http://www.example.com/owl/Hi-Onto/Quantities.owl#"
7 <owl:Ontology about="">
8   <rdfs:comment>
9     An Ontology defining quantities.
10  </rdfs:comment>
11  <owl:imports rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns" />
12  <owl:imports rdf:resource="http://www.w3.org/2000/01/rdf-schema" />
13  <owl:imports rdf:resource="http://www.w3.org/2002/07/owl" />
14 </owl:Ontology>
15 <!-- Definition forVolume -->
16 <owl:Class rdf:ID="Volume">
17   <owl:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
18 </owl:Class>
19 <owl:Property rdf:ID="magnitude">
20   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int" />
21   <rdfs:domain rdf:resource="#Volume" />
22 </owl:Property>
23 <owl:Property rdf:ID="unit">
24   <rdfs:range rdf:resource="#VolumeUnit" />
25   <rdfs:domain rdf:resource="#Volume" />
26 </owl:Property>
27 <owl:Class rdf:ID="VolumeUnit">
28   <owl:oneOf rdf:parseType="Collection">
```

---

---

```
29 <DeliveryType rdf:ID="Liter"/>
30 <DeliveryType rdf:ID="QubicMeter"/>
31 <DeliveryType rdf:ID="Pint"/>
32 </owl:oneOf>
33 </owl:Class>
34 . . .
35 </rdf:RDF>
```

---

Listing 3 and Listing 4 show two examples of ontologies defined in OWL. Both are used in our later examples for describing semantics of the service introduced in section 2.2. Listing 3 illustrate a generic ontology defining terms related to physical quantities. We show how `Volume` is defined in this ontology as a combination of a magnitude and a unit. Listing 4 shows another ontology specific to our service. Note that this ontology is importing another generic ontology, named `geo-spatial`, which defines the destination city as a subclass of European cities.

**Listing 4.** Ontology describing the concepts particularly used in the semantic description of Freight service.

---

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <rdf:RDF
3   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
5   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
6   xmlns:owl="http://www.w3.org/2002/07/owl#"
7   xmlns:geo="http://www.example.com/owl/Hi-Onto/geospatial.owl#"
8   xmlns="http://www.example.com/owl-s/freight/Concepts.owl#">
9   <owl:Ontology rdf:about="">
10    <owl:imports rdf:resource=
11      "http://www.example.com/owl/Hi-Onto/geospatial.owl"/>
12  </owl:Ontology>
13  <owl:Class rdf:ID="Dest_City">
14    <rdfs:label>destination</rdfs:label>
15    <rdfs:subClassOf rdf:resource=
16      "http://www.example.com/owl/Hi-Onto/geospatial.owl#city"/>
17    <rdfs:subClassOf>
18      <owl:Restriction>
19        <owl:onProperty rdf:resource=
20          "http://www.example.com/owl/Hi-Onto/geospatial.owl#inside"/>
21        <owl:hasValues rdf:resource=
22          "http://www.example.com/owl/Hi-Onto/geospatial.owl#Europe"/>
23      </owl:Restriction>
24    </rdfs:subClassOf>
25  </owl:Class>
26  . . .
27 </rdf:RDF>
```

---

It is good practice to separate ontologies specific to the application from more generic ontologies. It is also a common convention in the OWL-S community to name application specific ontology as `Concepts` (in our example, `Concepts.owl` is an ontology specific to our Web Service). It is also a common practice to separate ontologies specific to a domain (also called domain ontology), for example a `Transport` ontology, from more generic ontologies (also called higher-level ontologies), such as the `Quantity` ontology.

## **4. Semantic Web Services**

Web Services technology offers the ability for programs to find, invoke and interact with each other in a dynamic fashion and on the fly. However, there are still limitations on the Web Service technology that Semantic Web Services aims to overcome. In the following subsections, first we show the limitations of Web Services technology that motivated the initiatives such as OWL-S and WSMO. Then we show overviews of both OWL-S and WSMO.

### **4.1. Challenges to Web Services**

From the perspective of service requesters, a conventional process of using Web Services includes three steps: define the request for a Web Service, discover and locate the interface of a qualified service, and invoke the corresponding service implementation via the binding of the published service interface. A typical scenario of developing a client application which makes use of Web Services is as follows. First, the programmer defines what functionality is required to fulfill the application requirements. Then she searches for Web Services qualified for that purpose. The programmer can search manually or use appropriate tools to assist the query processing. The result of the search normally contains a number of candidate Web Services satisfying the search parameters. Then, the programmer has to decide which Web Service is the best candidate. If none of them is qualified, a new query has to be issued. After a qualified service is found, the programmer retrieves the information of the service interface and writes the code to call it in the client application. In this 'traditional' way of using Web Services, only the binding between the service implementation and the service interface is dynamic and could be decided at the run time. The search for services is conducted at the design time and the decision whether a service is qualified or not is made by human agents. In most cases, the call of a service is hard-coded as a call to a service interface. Although the service implementation is decoupled from the client applications, the service interface is still tightly coupled with client applications.

An ideal automated process of using Web Services should enable client applications to handle the three steps automatically by machines instead of a programmer and the switch from one step to another should be seamless without human intervention. However, current Web Services technologies are not capable of supporting such an ideal automated invocation of Web Services. The intervention of human is still inevitable for the grand scenario in which a requester can find and invoke a qualified service on the fly.

In the past, different service providers have developed their unique style and manner of doing various businesses, and the corresponding enterprise information systems as well as business applications have been developed independently by developers who might think differently. As a result, the prevalence of semantic heterogeneity in Web Services published by different providers is inevitable and thus it leaves some obvious gaps in the steps of using Web Services which needs human intervention to assist the process manually.

The first gap is between step of defining a service request and step of discovering services that is caused by the lack of complete specification for description of Web Services. For the purpose of automated service discovery, the requests for Web Services must be composed in machine-processable format. On one side, it is the request of a Web Service from a service requester. And on the other side, it is the service description published by a service provider. The discovery process is defined to match the service request against a number of service descriptions.

A comprehensive description of a Web Service consists of three levels of knowledge how to use this Web Service: communication, capability and functionality, and business description. The lowest level is the communication level service description, e.g. provided by WSDL description. The middle level of a service comprehension defines the service semantics by expressing the functionality and the capability of a Web Service. In general, the functionality of a Web Service prescribes its intended purpose. The quantitative and qualitative constraints on the functionality of a service are expressed by a capability specification. Business descriptions facilitate the communication and negotiation to set up commercial relationships in order to use services commercially and legally.

The second gap lies between the step of service discovery and the step of service invocation because of the degree of precision of service discovery. Within the process of service discovery, the matching process lacks an accurate measure of how well a service is qualified to fulfill a service request. The result of a discovery is to return all services that are “sufficiently similar” to the request. This implies that human assistance is still required to determine whether a ‘sufficiently similar’ service is the desired one. Therefore, the support of automatic service discovery, invocation, and composition is rather limited. Furthermore, a considerable amount of customer code has to be implemented in the client applications.

### ***Beyond WSDL Descriptions and UDDI Discovery***

Using the Web Services technology, we can describe the data types used in communicating with a Web Service (e.g. through input and output description in WSDL). The UDDI discovery is a keyword based search on service description stored in the registry. Two major types of solutions can improve the discovery of Web Services are emerging.

The first type is to apply mathematical models and methods to analyze the WSDL documents attached with Web Services to infer the underlying semantics. Dong et al. (2004) proposes an effective approach of searching Web Services by clustering parameter names of operations on a set of Web Services into semantically meaningful concepts. These clustering can be used to determine similarity of Web Services, based on statistics computed over a large number of WSDL documents. This trend of research work focus on input/output matching to find a list of operations with similar inputs (outputs) with a given inputs (outputs) of an operation, and operation matching to find a list of operations similar with a given operation.

The second type is to define standard description languages at a higher level of abstraction on top of WSDL and matching algorithms to compare the capabilities provided in such description languages. Semantic Web Services provide the capability that input and outputs can be described by ontologies. Further than describing Services by inputs and outputs, Semantic Web describes services by a set of conditions that must hold before the service execution and will hold after the service execution (see section 5.1). These features provide more expressiveness and precision to describe a service as compared to that of WSDL and potentially improve the discovery process. A number of matching algorithms have been proposed for semantic description models by Paolucci et al. (2002b) and Sycara et al. (2003) The matchmaker proposed by Paolucci et al. (2002b) integrates semantic matching into UDDI. For a request, the proposed algorithm performs a match between all the outputs of the request against outputs of a service advertisement (a service description published in a registry) and a match between all the inputs of a service advertisement against the inputs of the request to determine the degree of match such as exact, plug in, subsumes, and fail.

### ***Beyond Stateless Services***

Web Services technology, particularly WSDL, is based on the assumption that a service is stateless. In other words, a service receives a set of data as an input (input messages) before it starts execution and produces output data (output messages) after its execution. However, in general, a service can be more complicated with a more complex pattern of message exchange. A challenge to Web Services technology is to provide the possibility of defining complex patterns of interactions as compared to one step execution in stateless services.

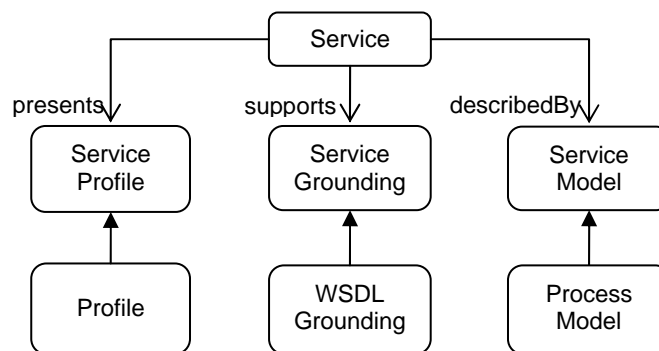
Describing patterns of message exchange is essential when we require building more complex services by combining a set of existing services. The improvement of the discovery process (as described in the last section) allows dynamic composition of the services that can have two different aspects. One is building composed services that its components are dynamically found and invoked on the fly. Second, the descriptions can be used in more intelligent tools to compose Web Services automatically or semi-automatically (section 7).

## **4.2. OWL-S**

OWL-S is a specification for describing semantics of Web Services (Martin et al., 2004a). It is a subsequent development of the DAML-S specifications and developed as part of the DAML program (DARPA Agent Markup Language, see <http://www.daml.org>). While DAML-S was built upon the DAML+OIL, since version 1.0 OWL-S is based on OWL (McGuinness, 2004).

OWL-S lays its basis on a process ontology and benefits from developments in workflow modeling as well as the agent technology (McIlraith et al., 2001). OWL-S describes services by three components namely, ServiceModel, ServiceProfile and ServiceGrounding. The diagram in Figure 3 illustrates the three components and their relations, and their description is as follows:

- The significant part of the three is the ServiceModel that includes descriptions such as, definition of functional parameters, the interaction pattern of a service with the invoker, and its execution mechanism in case of a composite service. To that end, OWL-S presents (but is not limited to) ProcessModel. The ProcessModel describes a service as a Process (Figure 3) by its functional parameters: Inputs, Outputs, Preconditions and Effects (IOPEs). It also specifies the component processes of a composite service and their execution order and binding of the inputs and outputs of component processes.



**Figure 3.** An overview of main concepts in OWL-S ontology for Web Service description.

- The ServiceProfile foresees information that can be required to search for a service in a service registry. An OWL-S Profile consists of information such as, ContactInformation of the service providers, ServiceCategory and other non-functional service parameters. Furthermore, a Profile contains a replication of the functional parameters (IOPEs) presented in its ProcessModel.
- The ServiceGrounding binds the semantic description to the details of accessing and executing a service, such as communication protocol and message format. At present, OWL-S offers specifications for grounding to WSDL descriptions.

Many tools are developed and available based on OWL-S specifications and we briefly introduce them at the end of this paper. At the time of writing, OWL-S 1.1 is also proposed to W3C as a recommendation.

### 4.3. WSMO

The Web Service Modeling Ontology (WSMO) is a specification to describe the various aspects related to Semantic Web Services (WSMO, 2004a; see also <http://www.wsmo.org>). The WSMO specification is mainly developing in relation to SDK project cluster (see [www.sdkcluster.org](http://www.sdkcluster.org)). It is presented in WSML that is a language for formalizing Web Service descriptions (WSML, 2005). The role of WSML for WSMO is comparable to that of OWL for OWL-S. However, WSML has been particularly developed for WSMO.

WSMO lays its foundation on knowledge representation and logical reasoning and benefits from experiences gained in developing UPML (Omelayenko, 2000) and several case studies (e.g. WSMO, 2004b) in different application domains. The main

components of WSMO specifications are Goals, Web Services, Ontologies and Mediators, described in the following:

- WSMO describes Web Services by their Capabilities and Interfaces (Figure 4). The Interface description contains two closely related notions of Choreography and Orchestration. Choreography defines the information required to interact with the Web Service and Orchestration contains information describing a composite Web Service.
- Goals represent the types of objectives that users would like to achieve via Web Services. The WSMO definition of goal describes a request for a service by means of defining the state of the desired information space and the desired state of the world after the execution of the intended service.
- Ontologies provide the definition of the concepts and relations used in the other three component descriptions. For example, a goal can import existing concepts and relations defined in an ontology by either extending or simply reusing them as appropriate.
- Finally, Mediators specify interoperability mechanisms. All component descriptions use mediators to define a valid interaction with any other component.

WSMO is a recent initiative in comparison to the OWL-S specification, consequently, still under development and subject to modification. As a result, we tend to present more detailed discussions along with examples based on OWL-S, since it is a solid specification. Alternatively, our discussion around WSMO is based on the available documents at the time of writing.

It is worth mentioning that WSMO and OWL-S have different technical coverage. For example, the WSMO initiative presents specifications for architecture and execution environment as well as a language for ontology definitions, while these lay outside the scope of the OWL-S initiative. In fact, DAML recently started initiatives on these topics (see <http://www.daml.org/services/swsa/>). Our focus here is limited to the specifications for describing semantics of services.

## **5. Semantic Service Descriptions**

The first step in the development of Semantic Web Services is to define the semantics of services. Later steps are concerned with the usage of the semantic descriptions. The next section shows how semantics are being used for intelligent discovery while this section is dedicated to the aspects of the former step.

We divide this section to three parts: first describing the properties defining the functionality of a service; then, describing properties that do not affect the functionality but contributing to the service descriptions; finally, describing the association of the semantic descriptions to schematic definitions. We discuss how these descriptions are presented in OWL-S and WSMO, along with examples from OWL-S that are related to our examples of WSDL descriptions and OWL ontologies in last sections.

## 5.1. Functional Properties

Describing semantics of a service in terms of its input and output parameters is an intuitive approach. WSDL descriptions provide a schematic description of these parameters. For example, in a service for selling books, credit card information and book information are inputs and purchase acknowledgement is the output. Furthermore, services can be described by (1) the conditions required for a service to perform successfully –e.g. validity of the credit card; and (2) the conditions that would hold after the successful execution of a service –e.g. the charged credit card and the shipped book. Both these types of conditions can be expressed in a logical language. The former conditions are called Preconditions and the later are named Effects, in OWL-S. The collection of the above descriptions (Input, Output, Precondition and Effect) is generally referred to as *functional properties* or *capabilities*.

OWL-S describes a service by a process model (Figure 3). A Process is described in terms of its Input and Output types as well as Precondition and Effect expressions (IOPEs). We show a simple example of the semantic description of a process in Listing 5. This process description defines the semantics for the WSDL description in Listing 2. The input and output types in Process descriptions are defined by ontologies unlike WSDL, where input and output types are defined by their structure and representation (schemata). Later in section 5.3, we explain how the semantic descriptions (in Listing 5) are bound to the WSDL descriptions (in Listing 2).

Listing 5 shows the input parameters defined in OWL-S (Lines 15 to 26). The input type “Destination” is defined as a city that is in turn defined in the OWL ontology in Listing 4 and the `size` is of type `Volume` defined in Listing 3. The output parameter `costReturn` is also defined as charge that is also defined in `Quantity` ontology (Listing 3). Using such description, a client agent can find out that the `destination` is the name of a city and not a country.

We can also see the third type of parameters in OWL-S Process descriptions called Locals. Their only use is to describe semantics of the services. Local parameters are neither provided by the invoker as input to a service, nor are they being returned to the invokers as output. They may be evaluated by any other means or they may only being used in logical expressions without any value being assigned to them at run time (as in our example). Local parameters appeared in OWL-S since version 1.1.

The Local parameter in our example is defined as the city of origin (lines 35 to 40). Our example shows how the semantic service description assumes the origin of the Freight-Cost operation. This is the answer to a question that possibly appears to anyone who reads the code in Listing 1. In fact, our service provider offers the service only from London. Precondition in lines 50 to 74 shows the assumption about the origin of the Freight service taken into account when it calculates the cost. A second precondition (lines 75 to 82) states that the submitted destination must be one of the cities of Rome, Berlin or Madrid.

### Listing 5. OWL-S process definition for the example in Section 2.2.

---

1 <rdf:RDF

---

---

```

2  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
4  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
5  xmlns:owl="http://www.w3.org/2002/07/owl#"
6  xmlns:expr=
7  "http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#"
8  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
9  xmlns:process="http://www.daml.org/services/owl-s/1.1/Process.owl#"
10 xmlns:geo="http://www.example.com/owl/Hi-Onto/geospatial.owl#"
11 xmlns="http://www.example.com/owl-s/freight/Process.owl#"
12 xml:base="http://www.example.com/owl-s/freight/Process.owl">
13 <!-- Atomic Process : Freight_cost -->
14 <!--Inputs-->
15 <process:Input rdf:ID="Freight_destination">
16   <process:parameterType
17     rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
18     http://www.example.com/owl-s/freight/Concepts.owl#Dest_City
19   </process:parameterType>
20 </process:Input>
21 <process:Input rdf:ID="Freight_size">
22   <process:parameterType
23     rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
24     http://www.example.com/owl/Hi-Onto/Quantities.owl#Volume
25   </process:parameterType>
26 </process:Input>
27 <!--Outputs-->
28 <process:Output rdf:ID="Freight_costReturn">
29   <process:parameterType
30     rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
31     http://www.example.com/owl/Hi-Onto/Quantities.owl#Charge
32   </process:parameterType>
33 </process:Output>
34 <!--Locals-->
35 <process:Local rdf:ID="Freight_origin">
36   <process:parameterType
37     rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
38     http://www.example.com/owl/Hi-Onto/geospatial.owl#City
39   </process:parameterType>
40 </process:Local>
41 <!--Process-->
42 <process:AtomicProcess rdf:ID="Freight_cost">
43   <process:hasInput
44     rdf:resource="#Freight_destination"/>
45   <process:hasInput
46     rdf:resource="#Freight_size"/>
47   <process:hasOutput
48     rdf:resource="#Freight_costReturn"/>
49 <!--Preconditions-->
50 <hasPrecondition>
51   <expr:SWRL-Condition rdf:ID="OriginIsLondon">
52     <rdfs:label>
53       SameAs(Freight_Origin, London)
54     </rdfs:label>
55     <expr:expressionLanguage rdf:resource=
56       "http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#SWRL" />
57     <expr:expressionBody rdf:parseType="Literal">
58       <swrl:AtomList>
59         <rdf:first>
60           <swrl:IndividualPropertyAtom>
61             <swrl:propertyPredicate
62               rdf:resource="#SameIndividualAtom" />
63             <swrl:argument1
64               rdf:resource="#Freight_origin" />
65             <swrl:argument2 rdf:resource=

```

---

---

```

66   "http://www.example.com/owl/Hi-Onto/geospacial.owl#London"/>
67   </swrl:IndividualPropertyAtom>
68   </rdf:first>
69   <rdf:rest rdf:resource=
70     "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
71   </swrl:AtomList>
72   </expr:expressionBody>
73   </expr:SWRL-Condition>
74 </hasPrecondition>
75 <hasPrecondition>
76   <expr:SWRL-Condition rdf:ID="DestinationLimitation">>
77     <rdfs:label>
78       One_of(Freight_Destination, (Rome, Berlin, Madrid))
79     </rdfs:label>
80     [Expression body in SWRL.]
81   </expr:SWRL-Condition>
82 </hasPrecondition>
83 </process:AtomicProcess>
84 </rdf:RDF>

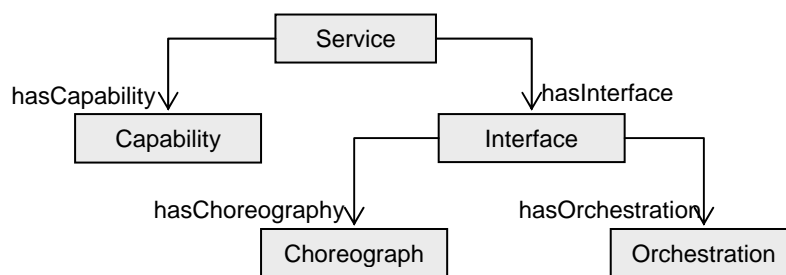
```

---

WSMO takes a different approach to describe functional properties. It describes Web Services by two components: Interface and Capability (Figure 4). The Web Service Interface determines the information needed for executing and interacting with the Web Service. We discuss the Interface later on in sections 5.3 and 7. Here we continue with Capability; which describes the functional properties of a service by means of a set of conditions to hold before its invocation and a set of conditions that would hold after its execution.

WSMO Capability describes service functionality in terms of the following parameters:

- Preconditions: conditions that should hold for the information space before the Web Service is performed;
- Post-conditions: conditions that will hold for the information space after a successful completion of the Web Service;



**Figure 4.** Main classes used in the description of WSMO Web Services.

- Assumptions: conditions that should hold for the state of the world before the Web Service is performed; and
- Effects: set of conditions that will hold in the state of the world after a successful completion of the Web Service.

A distinction between the information space and real world state is made by WSMO. An example of a WSMO Precondition for a service is validity of the credit

card of a customer. This condition can be evaluated in the information space available to the service, for example, by checking the credit card number with another service.

An Assumption is a condition on the state of the world that would not be evaluated in the information space available to the service. An example of an Assumption is the condition on the `origin` of the `Cost` service, in our earlier example. This Assumption is not to be evaluated in the available information space. Note that, if `origin` was part of the service interface and could be evaluated in the information space, then it should appear as a Precondition. An Assumption can be evaluated against a request for a service. Furthermore, an intelligent invoker of a service can be informed that the service provider is assuming a condition before invoking the service. In our example, the invoker would be informed that the `origin` of the service is London. One can see the difference with the Precondition here. In the second OWL-S Precondition, a condition on the destination is defined. In this case the condition is defined on a value that is provided in the information space and provided by the invoker. OWL-S treats both these types of conditions as precondition. As we can see in Listing 5, both above conditions appear as OWL-S Precondition.

WSMO treats input and output type description implicitly as part of its Preconditions and Post-conditions. One can specify input or output types of a service as a constraint in WSMO Preconditions or Post-conditions, respectively. WSMO treats Local parameters also implicitly in its Capability descriptions, just as input and output type descriptions.

An example of the WSMO service description for the service description of Listing 2 is shown in Listing 6. The underlying language for WSMO is Web Service Modeling Language (WSML, 2005). At the first glance, one can see that WSML is not an XML-based language.

The Capability description for the service starts from line 17. The input and output type definitions appear in the Precondition (lines 23 and 24) and Post-condition (line 35), respectively. One can also observe how the two OWL-S Preconditions appear in WSMO Precondition (line 25 to 28) and Assumption (line 42).

### Listing 6. WSMO capability description for the same service described in Listing 5

---

```
1 Namespace
2 geo: <<http://www.example.com/owl/Hi-Onto/geospatial.wsml#>>
3 quantity: <<http://www.example.com/owl/Hi-Onto/Quantities.wsml#>>
4 concept: <<http://www.example.com/owl-s/freight/Concepts.wsml#>>
5 dc: <<http://purl.org/dc/elements/1.1#>>
6 targetnamespace:
7     <<http://www.example.com/wsmo/freight/FreightWS#>>
8 Webservice
9     <<http://www.example.com/wsmo/freight/FreightWS.wsml#>>
10 nonFunctionalProperties
11     ... [for non-functional properties see section 5.2]
12 endNonFunctionalProperties
13 importedOntologies {
14     <<http://www.example.com/owl/Hi-Onto/geospatial.wsml#>>,
```

---

---

```

15 <<http://www.example.com/owl/Hi-Onto/Quantities.wsml>>,
16 <<http://www.example.com/owl-s/freight/Concepts.wsml>>}
17 capability freightCapability
18 precondition
19   axiom freight_precondition
20   definedBy
21     forAll ?Freight_size, ?Freight_Destination
22     (
23       ?Freight_Destination memberOf concept:Dest_city and
24       ?Freight_size memberOf quantity:volume and
25       (?Freight_Destination = berlin or
26        ?Freight_Destination = rome or
27        ?Freight_Destination = madrid
28       )
29     ).
30 postcondition
31   axiom freight_postcondition
32   definedBy
33     forAll ?Freight_costReturn
34     (
35       ?Freight_costReturn memberOf quantity:charge and
36     ).
37 assumption
38   axiom freight_assumption
39   definedBy
40     forAll ?Freight_Origin
41     (
42       ?Freight_Origin = london
43     ).
44 interface freight_Interface
45 choreography ...
46 orchestration ...

```

---

Since OWL-S version 1.1, the collection of Outputs and Effects along with a condition is called Result. The Result binds service Outputs and/or Effects to a condition. Conditions assigned to these elements add further dynamism to the OWL-S descriptions. These conditions are introduced to describe a service that may have alternative outcomes (i.e. Outputs and/or Effects) depending on the circumstances. That is, if a service produces different output message under special conditions, a Result in form of combination of a condition and an output is used to describe it.

In a more complex example here, we show the difference between Effects and Post-conditions in WSMO and further clarify the conditional Results in OWL-S. In the following we describe a book-selling service that can have two possible alternative outcomes.

- First, a successful purchase in case the book is in stock that results in:
  - charging the credit card, i.e. WSMO Post-condition and OWL-S Effect
  - shipping the book to the customer's address, i.e. WSMO Effect and OWL-S Effect
  - sending a purchase acknowledgement to the client agent, i.e. WSMO Post-condition and OWL-S Output.
- Second, a successful reservation for the book when book is not in stock that results in:
  - reserving a book on the next delivery, i.e. WSMO Post-condition and OWL-S Effect

- sending a reservation acknowledgment, i.e. WSMO Post-condition and OWL-S Output.

The above OWL-S Results will take place depending on the condition: “if the book title is in stock”. OWL-S allows us to define two different sets of results under different conditions. Such conditions appear explicitly in OWL-S as a discrete part of description of Results –see (Martin et al, 2004a) for example code of conditional Result. We refer to the description of the conditional Results in section 7.2 on choreography, because in fact, it is a way of describing the message exchange pattern in OWL-S.

## 5.2. Non-Functional Properties

Apart from the properties discussed in section 5.1 that directly influence the functionality of services, there are other properties that are important to describe a service. A good example is the information about the service provider, while they may be important for the service consumer, they do not affect the service functionality. Both OWL-S and WSMO provide specifications to define non-functional properties.

Non-functional properties appear in the service Profile for OWL-S (Figure 3). There are three properties presenting the non-functional properties in Profile, namely:

- `contactInformation` provides the information for contacting individuals in charge of the service. Early versions of OWL-S provided a specific OWL class called `Actor` to define this information. Using `Actor` specification is now optional and any other specification can be used to provide the contact information –e.g. `VCard`. We use `Actor` in our example in Listing 7 (lines 19 to 23).
- `serviceCategory` offers the possibility of defining a service category. Here, we can define the service type in different national or international categorization scheme. Note that there is another way of defining ontological service type by building a taxonomy tree for service Profiles –see (Martin et al., 2004a) for `ProfileHierarchy`.
- `serviceParameter` allows defining a name-value pair for extending non-functional parameters, that is to say adding optional parameters. We defined an imaginary quality rating parameter in our example (lines 25 to 34).

### Listing 7. OWL-S Profile

---

```

1 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
2   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
3   xmlns:owl="http://www.w3.org/2002/07/owl#"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5   xmlns:process="http://www.daml.org/services/owl-s/1.1/Process.owl#"
6   xmlns:profile="http://www.daml.org/services/owl-s/1.1/Profile.owl#"
7   xmlns:actor=
8     "http://www.daml.org/services/owl-s/1.1/ActorDefault.owl#"
9   xml:base="http://www.example.com/owl-s/freight/Profile.owl">
10 <profile:Profile rdf:ID="Freight_Cost">
11   <profile:serviceName>
12     Freight_Cost_Calculator
13   </profile:serviceName>
14   <profile:textDescription>

```

---

---

```

15   This service calculates the cost of transporting goods
16   from London to some cities in Europe.
17 </profile:textDescription>
18 <profile:contactInformation>
19   <actor:Actor rdf:ID="customer-relation">
20     <actor:name>John Doe</actor:name>
21     <actor:title>Sales Manager</actor:title>
22     <actor:email>john.d@freight.com</actor:email>
23   </actor:Actor>
24 </profile:contactInformation>
25 <profile:serviceParameter>
26   <profile:ServiceParameter>
27     <profile:serviceParameterName
28       rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
29       SomeQualityRating
30     </profile:serviceParameterName>
31     <profile:sParameter rdf:resource=
32 "http://www.example.com/owl/Hi-Onto/ServiceQuality.owl#qualityRating_Good" />
33   </profile:ServiceParameter>
34 </profile:serviceParameter>
35 <profile:hasInput>
36   <process:Input rdf:ID="Freight_destination">
37     <process:parameterType rdf:datatype=
38       "http://www.w3.org/2001/XMLSchema#anyURI">
39 http://www.example.com/owl-s/freight/Process.owl#Freight_destination
40     </process:parameterType>
41   </process:Input>
42 </profile:hasInput>
43 <profile:hasInput>
44   <process:Input rdf:ID="Freight_size">
45     <process:parameterType rdf:datatype=
46       "http://www.w3.org/2001/XMLSchema#anyURI">
47       http://www.example.com/owl-s/freight/Concept.owl#Volume
48     </process:parameterType>
49   </process:Input>
50 </profile:hasInput>
51 <profile:hasOutput>
52   <process:Output rdf:ID="Freight_costReturn">
53     <process:parameterType rdf:datatype=
54       "http://www.w3.org/2001/XMLSchema#anyURI">
55 http://www.example.com/owl-s/freight/Process.owl#Freight_costReturn
56     </process:parameterType>
57   </process:Output>
58 </profile:hasOutput>
59 </profile:Profile>
60 </rdf:RDF>

```

---

The main purpose of OWL-S Profile is to present information needed for registries such as UDDI and formulating service requests. As a result, Profile contains more than, merely, non-functional properties. OWL-S functional properties appear in the Profile as well as in the Process description, but only as a replication of IOPEs in the Process. OWL-S does not impose any constraint on the consistency of IOPEs in the Profile with those in the Process (Martin et al., 2004a). The IOPEs presented in the Process description is a more reliable source of knowledge, although, either one could be taken into account. In our example of a Profile, the first input parameter appears as a pointer to the input definition in the Process description (see lines 35 to 41); while the second input parameter definition points directly to the ontology file (see lines 42 to 49). We believe the former approach is good practice as it prevents inconsistencies between Profile and Process descriptions. The

later practice may cause inconsistencies in case the PROCESS description is modified without affecting the Profile.

The notion of non-functional properties has a much wider interpretation in WSMO. Non-functional properties are bound to most concepts in WSMO. They may be assigned not only to services but also to other component definitions such as Ontologies, Goals, etc. WSMO defines a very detailed set of non-functional properties that are suitable to be used by service registries (WSMO, 2004a). We show an example of such description in Listing 8 including a few of the properties.

**Listing 8.** An example definition of a non-functional properties for the WSMO Web Service in Listing 6.

---

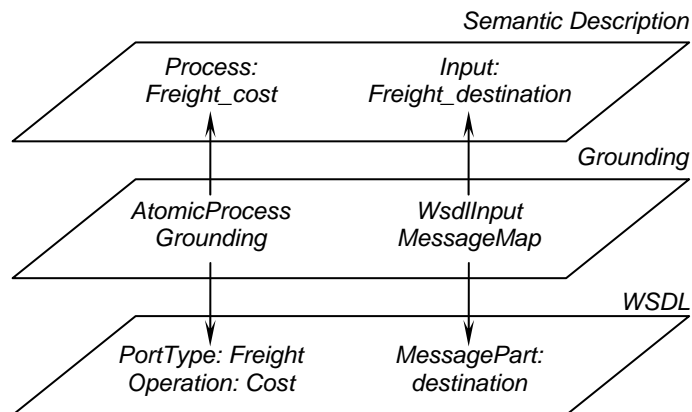
```
1 Webservice
2 <<http://www.example.com/wsmo/freight/FreightWS.wsml>>
3 nonFunctionalProperties
4   dc:title hasValue "Freight cost calculator"
5   dc:creator hasValue "Our Imaginary Freight Ltd."
6   dc:description hasValue
7     "A Web Service for calculating the freight cost from
8     London to some European cities."
9   dc:publisher hasValue
10    "DERI International"
11  dc:type hasValue <<http://www.wsmo.org/2004/d2/#webservice>>
12  dc:format hasValue "text/html"
13  dc:language hasValue "en-us"
14  dc:relation hasValues
15    {<<http://www.example.com/wsml/Hi-Onto/Quantities.wsml>>,
16     <<http://www.example.com/wsml/Hi-Onto/geospatial.wsml>>,
17     <<http://www.example.com/wsmo/freight/Concepts.wsml>>}
18  dc:coverage hasValues {tc:austria, tc:germany}
19 endNonFunctionalProperties
```

---

### 5.3. Binding Semantics to Services

Grounding is a mechanism to assign the semantic descriptions of a service to its schematic description. We discuss only the OWL-S grounding in the following as WSMO grounding is still under development at the time of writing.

OWL-S presents two approaches for binding the Process description to the WSDL description. In the first approach, grounding defines the bindings between an AtomicProcess to an operation in a WSDL description (Figure 5), without the need to modify the WSDL description. In the second approach, OWL-S offers an extension to the WSDL description. This extension provides the possibility of defining the bindings to the OWL-S semantic description inside the WSDL description. One may use any of the two approaches depending on the possibility of modifying the WSDL description and suitability of hard coding the bindings to the WSDL description.



**Figure 5.** OWL-S Grounding relates Atomic processes to WSDL descriptions.

Listing 9 shows the grounding for binding the semantic description of the OWL-S Process in Listing 5 to the WSDL description in Listing 2, using the first approach. Our WSDL Grounding (line 12) binds the OWL-S process “Freight\_cost” (lines 14 and 15 in Listing 5) to an operation in the WSDL description (lines 16 to 25 in Listing 2). The WSDL operation is uniquely defined by its PortType (Freight) and its Operation name (cost).

**Listing 9.** Grounding descriptions for binding the semantics in Listing 5 to WSDL description in Listing 2

---

```

1 <rdf:RDF
2   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:service="http://www.daml.org/services/owl-s/1.1/Service.owl#"
4   xmlns:grounding=
5     "http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
6   xml:base="http://www.example.com/owl-s/freight/Grounding.owl">
7   <grounding:WsdGrounding rdf:ID="WsdGrounding">
8     <service:supportedBy rdf:resource="FreightService"/>
9     <grounding:hasAtomicProcessGrounding
10      rdf:resource="#Freight_cost_Grounding"/>
11   </grounding:WsdGrounding>
12   <grounding:WsdAtomicProcessGrounding
13     rdf:ID="Freight_cost_Grounding">
14     <grounding:owlsProcess rdf:resource=
15 "http://www.example.com/owl-s/freight/Process.owl#Freight_cost"/>
16     <grounding:wsdOperation>
17       <grounding:WsdOperationRef>
18         <grounding:portType rdf:datatype=
19           "http://www.w3.org/2001/XMLSchema#anyURI">
20           http://www.example.com/axis/WSDL/freight.wsdl#Freight
21         </grounding:portType>
22         <grounding:operation rdf:datatype=
23           "http://www.w3.org/2001/XMLSchema#anyURI">
24           cost
25         </grounding:operation>
26       </grounding:WsdOperationRef>
27     </grounding:wsdOperation>
28     <grounding:wsdInputMessage rdf:datatype=
29       "http://www.w3.org/2001/XMLSchema#anyURI">

```

---

---

```

30     http://www.example.com/axis/WSDL/freight.wsdl#costRequest
31 </grounding:wSDLInputMessage>
32 <grounding:wSDLInput>
33   <grounding:WSDLInputMessageMap>
34     <grounding:owlsParameter rdf:resource=
35       "http://www.example.com/owl-s/freight/Process.owl#Freight_destination"/>
36     <grounding:wSDLMessagePart rdf:datatype=
37       "http://www.w3.org/2001/XMLSchema#anyURI">
38       http://www.example.com/axis/WSDL/freight.wsdl#destination
39     </grounding:wSDLMessagePart>
40   </grounding:WSDLInputMessageMap>
41 </grounding:wSDLInput>
42 . . .
43 </grounding:WSDLAtomicProcessGrounding>
44 </rdf:RDF>

```

---

The rest of the example (lines 28 to 41) shows how the `destination` parameter from the WSDL description is bound to the Input parameter `Freight_destination` in the OWL-S description.

To follow the second approach for grounding we should alter the WSDL description. For example, the operation definition in Listing 2 (lines 32 to 32) should be modified as in Listing 10 –see (Martin et al., 2004b) for details. Using this approach the binding to the semantic descriptions is hard-coded into the WSDL description.

#### Listing 10. Grounding by extension to WSDL description.

---

```

1 <wSDL:operation name="cost"
2   owl-s-process=
3   "http://www.example.com/owl-s/freight/Process.owl#Freight_cost"
4   parameterOrder="destination size">

```

---

## 6. Intelligent Service Discovery

In this section we describe how semantics adds value to the Web Services discovery. One of the motivations of research on Semantic Web Services is to improve the functionality of service registries. The goal of discovery is to find an appropriate Web Service that meets certain functional and non-functional criteria stated by a service consumer. The most crucial issue in service discovery is to match service requests with service descriptions to find the most appropriate service or a list of services.

The discovery process can be performed either at design time or run time by a requester human using software tools (discovery service) or by a client agent (a program). The key to facilitate smarter and more flexible automation of service provision (from the perspective of providers) and use (from the perspective of requesters) is the ability to discover, select and invoke the appropriate services dynamically according to the requirements of requesters and the constraints of providers at run time.

Discovery approaches based on WSDL/UDDI are based on keyword matching of WSDL descriptions, and businesses, services, and tModels in UDDI repository. Current UDDI specification allows publishing and discovery of service descriptions.

One way of improving the functionality is to augment the service descriptions in the registry. Adding information about the semantics to the UDDI can provide richer result. For example, the UDDI registry can provide information about the input and output data types defined in an ontology. A way of augmenting service descriptions in UDDI is presented in (Paolucci et al. 2002a), where the OWL-S profile information is stored in UDDI registries using tModels (see section 2.3). Following our OWL-S example, we show how our Profile (in Listing 7) can appear in a UDDI information model in Listing 11. Using this approach a service consumer can access the OWL ontological description of the input and output types through the information stored in the UDDI. Consequently, an intelligent invoker can match the desired input and output with those of services.

**Listing 11.** Example of how an OWL-S Profile may appear in UDDI data model (Paolucci et al. 2002a).

---

```

1 <businessEntity businessKey="" >
2 <name>V-Transport</name>
3 <contacts>
4 <contact useType="Sales Manager">
5 <personName>John Doe</personName>
6 <email>john.d@freight.com</email>
7 </contact>
8 </contacts>
9 <businessServices>
10 <businessService serviceKey="" >
11 <name>Freight_Cost_Calculator</name>
12 <description>
13 This service claculates the cost of transporting goods
14 from London to some cities in Europe.
15 </description>
16 <categoryBag>
17 <keyedReference
18 keyValue="http://www.example.com/owl-s/freight"
19 tModelKey="[the key for OWL-S service description]"/>
20 <keyedReference keyName="Freight_size"
21 keyValue=
22 "http://www.example.com/owl-s/freight/Concept.owl#Volume"
23 tModelKey="[the key for OWL-S Profile input]"/>
24 <keyedReference
25 keyName="Freight_destination"
26 keyValue="http://www.example.com/owl-
27 s/freight/Process.owl#Freight_destination"
28 tModelKey="[the key for OWL-S Profile input]"/>
29 <keyedReference
30 keyName="Freight_costReturn"
31 keyValue=
32 "http://www.example.com/owl/Hi-Onto/Quantities.owl#Charge"
33 tModelKey="[the key for OWL-S Profile output]"/>
34 </categoryBag>
35 </businessService>
36 </businessServices>
37 </businessEntity>

```

---

Another way of improving the functionality of the registry is to enhance its discovery, rather than only augmenting the stored information and the outcome of a service. In other words, UDDI service discovery is mainly keyword-based and bringing semantics to Web Services is to push the service discovery one step ahead.

To that end, we need inference engines capable of interpreting semantic description of services. An intelligent matching is based on matching the input and output types as well as the preconditions and effects. Therefore, inference engines are required for interpreting logical expressions describing the service capabilities and matching them with the service requests. This issue is indeed the most important step at present and attracts much research.

There are two main approaches to make use of inference engines; first by adding intelligent matching functionalities to the existing registries. The OWL-S community made an effort to bring such intelligent matching functionalities to UDDI (Paolucci et al., 2002a and 2002b). The second approach is to implement a registry on top of an inference engine. A good example of this approach is the Internet Reasoning System (IRS-III; Domingue, 2004). IRS-III is built on top of an inference engine and implemented intelligent service matching capabilities for WSMO specifications.

As the service matching becomes more complex the service request itself becomes a discrete topic. A specification for service request is to enable a client agent to express its need without the need to determine the implementation details. That is, it should allow us to define a set of requirements free from details present in description of services. Again, OWL-S and WSMO have different approaches to perceive a service request. In OWL-S, Profile (section 5.2) is seen not only as a collection of information for the registries but also as a template for expressing a service request. A request for a transport consulting service that produces charges by receiving a city origin, a city of destination and the size is presented in Listing 12.

**Listing 12.** Simplified example of a service request Profile.

---

```

1 <profile:Profile rdf:ID="Transport_Consulting_Service_Request">
2   <profile:hasInput>
3     <process:Input rdf:ID="Origin">
4       <process:parameterType>
5         http://www.example.com/owl/Hi-Onto/geospatial.owl#City
6       </process:parameterType>
7     </process:Input>
8   </profile:hasInput>
9   <profile:hasInput>
10    <process:Input rdf:ID="Destination">
11      <process:parameterType>
12        http://www.example.com/owl/Hi-Onto/geospatial.owl#City
13      </process:parameterType>
14    </process:Input>
15  </profile:hasInput>
16  <profile:hasInput>
17    <process:Input rdf:ID="size">
18      <process:parameterType>
19        http://www.example.com/owl/Hi-Onto/Quantities.owl#Volume
20      </process:parameterType>
21    </process:Input>
22  </profile:hasInput>
23  <profile:hasInput>
24    <process:Input rdf:ID="Cost">
25      <process:parameterType>
26        http://www.example.com/owl/Hi-Onto/Quantities.owl#Charge
27      </process:parameterType>
28    </process:Input>

```

---

---

```

29 </profile:hasInput>
30 <profile:serviceCategory>
31   <addParam:NAICS rdf:ID="NAICS-category">
32     <profile:value>Freight Consulting Services</profile:value>
33     <profile:code>541614</profile:code>
34   </addParam:NAICS>
35 </profile:serviceCategory>
36 </profile:Profile>

```

---

Paolucci et al. (2002a and 2002b) show how matching for services are performed for OWL-S service descriptions. The service request in Listing 12 can be performed by existing OWL-S tools. In Listing 13, we show how the request in Listing 12 can be coded in Java using the OWL-S Matchmaker tool (Paolucci et al., 2002a; details of the API at <http://www.daml.ri.cmu.edu/matchmaker/>). Note that the existing tools offer only partial support for matching the OWL-S service descriptions.

**Listing 13.** Simplified example of an intelligent discovery based on OWL-S Matchmaker.

---

```

1 //Defining the inputs and outputs types.
2 CapabilitySearch srch = new CapabilitySearch();
3 srch.addInput(
4   "http://www.example.com/owl/Hi-Onto/geospatial.owl#City");
5 srch.addInput(
6   "http://www.example.com/owl/Hi-Onto/geospatial.owl#City");
7 srch.addInput(
8   "http://www.example.com/owl/Hi-Onto/Quantities.owl#Volume");
9 srch.addOutput(
10  "http://www.example.com/owl/Hi-Onto/Quantities.owl#Charge");
11 //Seaching for the service.
12 OWLSMatchmakerClient mc;
13 mc = new OWLSMatchmakerClient();
14 MatchmakerResultList mrl = mc.query(srch);
15 If (mrl.size > 1) {
16   MatchmakerResult mr = mrl.get(int i);
17   String uddiKey = mr.getUddiKey();
18 //uddiKey can be used for invocation by the Web Services technology.
19 } //else: no suitable service found;

```

---

Requests for services in WSMO appear in form of Goals. Notion of Goal has a root in knowledge representation domain –see *Task* in (Omelayenko, 2000). Capability descriptions appear in both WSMO Goal and WSMO Web Service (WSMO, 2004a). In the WSMO service description, Capability determines the provided service and in a Goal it specifies the desired service. WSMO perceives a Goal as a generic problem definition that many different Web Services can offer solutions for. A notable difference between the Goals and Profiles is that Goals are by definition invocable unlike Profiles that can only be used to formulate requests. When a WSMO Goal is invoked, the discovery process finds suitable Web Service descriptions and may execute the corresponding service. To see the difference we show an example Java code from IRS-III (Domingue, et al. 2004 ) in Listing 14.

**Listing 14.** Simplified example of an intelligent discovery and invocation based on WSMO by IRS-III.

---

```

1 // Defining Inputs and the output types.

```

---

---

```

2 goalInputTypes.add(new GoalRole("origin", "City"));
3 goalInputTypes.add(new GoalRole("destination", "City"));
4 goalInputTypes.add(new GoalRole("size", "Volume"));
5 GoalRole goalOutputType = new GoalRole("cost", "Charge");
6 // Creating the Goal
7 Goal goal = new Goal(
8     nonFunctionalProperties,
9     goalInputTypes,
10    goalOutputType,
11    goalUsedMediators,
12    goalPostcondition,
13    goalEffect);
14 . . .
15 Volume vol = new Volume (3, "QubicMeter")
16 // Defining the actual input values for invocation.
17 achieveGoalInputValues.add( new
18     AchieveGoalInput("origin", "London"));
19 achieveGoalInputValues.add( new
20     AchieveGoalInput("destination", "Paris"));
21 achieveGoalInputValues.add( new
22     AchieveGoalInput("size", vol);
23 // invoking the service.
24 String response6 = irsServer.achieveGoal(goal, achieveGoalInputs);

```

---

An example of a service request can illustrate the dynamism required in a complex case of an intelligent discovery and show where the Semantic Web Services technology is heading. In the following we present two Goals:

- Goal1: buying a computer printer.
- Goal2: buying a HP Color LaserJet printer model: 2550 for a price less than £450.

In the description of the Goal1, the information: “buying a printer” is typically used by discovery mechanism to find a service. Such constraint is evaluated to find a service and satisfied before the service is invoked. However, constraints such as price and printer model, in Goal2, are typically inputs to perform an instance of a service. That is, in response to Goal2 the discovery mechanism finds services selling computer peripherals as well as those selling HP products; and execution mechanism finds the desired printer by executing the service. WSMO Goal descriptions allow a user to describe her desire freely with no consideration of what information is used by discovery and what is used by the execution mechanism. It is the discovery and execution mechanisms that extract the relevant information from a Goal.

Furthermore, WSMO offers the specification to describe the reduction of a generic goal (e.g. Goal1) to a more specific Goal or a sub-Goal (e.g. Goal2) by means of ggMediators (see section 6.1). WSMO (2004b) presents a set of detailed thorough examples.

## 6.1. Mediators

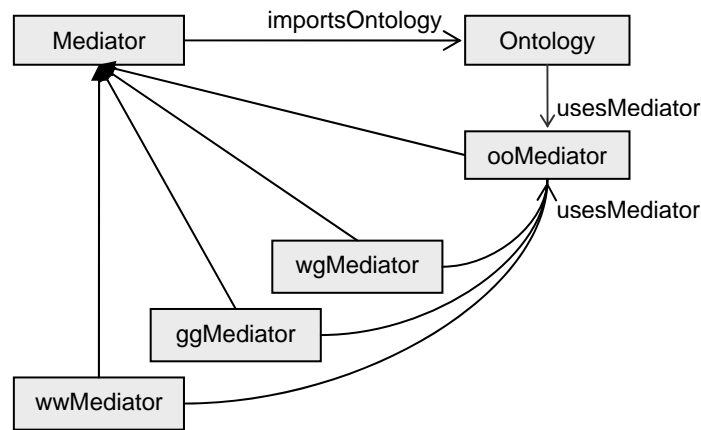
The notion of Mediator is specific to WSMO and it is one of its fundamental components. Mediator descriptions may be seen as a wrapper for a Web Service or a Goal (see section 6) that its objective is mediation. This wrapper keeps particular descriptions specific to a Mediator. All Mediators in WSMO are in fact Web

Services; however, WSMO allows further semantics assigned to such mediation services. OWL-S services may describe mediators but no specific semantics is assigned to such services. That is, one can define a Web Service to perform mediation without explicitly describing it as a mediator and vice versa; we cannot simply distinguish mediators from services by their OWL-S descriptions.

WSMO Mediators keep the following information:

- Mediation Service: a Goal or a Web Service to perform the mediation,
- Source: the entity providing inputs to the mediator,
- Target: where the result of the mediation will be provided to.

Different types of mediators are seen to mediate data or ontology in interaction between different components. Four types of Mediators (WSMO, 2004a) are introduced by WSMO as following (see Figure 6):



**Figure 6.** Different types of mediators in WSMO.

- wgMediator mediates Web Services to Goals. This mediator represents mediation between a Web Service and a Goal type that it fulfills. For example, different Web Services fulfilling Goal1 in the last section can have different mediators to mediate between the Web Service and the Goal1. As a result all Goals specializing Goal1 can use the same mediators.
- ggMediator mediates between two Goals. This mediator represents the reduction of the source Goal description into the target Goal.
- wwMediator mediates the information between two Web Services.
- ooMediator imports ontologies, resolve possible terminology mismatches and find mappings between ontologies – ooMediators are not meant to resolve syntactic mismatches (WSMO, 2004b). This mediator type can be used by the other three types of mediators to resolve ontological mismatches in the description of their source and target entities.

Another important use of WSMO mediators are in service compositions where we can use mediators to mediate between the component services of a composition. OWL-S only allows direct data bindings between component services and any

mediator would appear as a distinct service component in the composition. As OWL-S does not distinguish between Web Services and mediators, it is not a straight forward process to correspond an OWL-S service description to a WSMO Mediator.

## 7. Service Composition

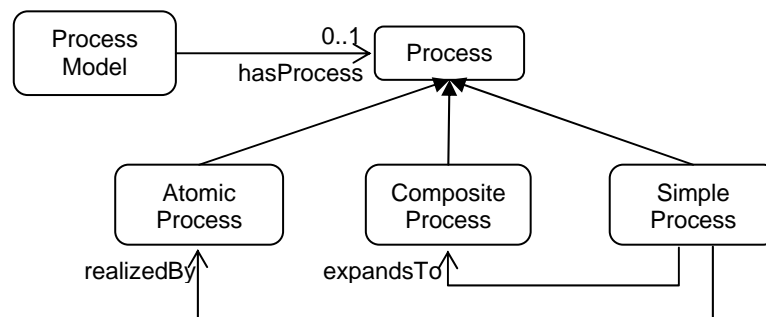
Reusability is a characteristic of Web Services that makes them suitable for building composite Web Services. A composed service is a more complex service that is built by combining existing services. Web Service composition attracted much attention in domain of Semantic Web Services and Artificial Intelligence (Sirin et al., 2003 and 2004; Ponnekanti & Fox 2002), particularly research on automatic service composition. Service composition has also been addressed in domain of Web Services –e.g. BPEL4WS (IBM, 2004). Nevertheless, this section focuses mainly on issues related to OWL-S composition models as well as existing WSMO Orchestration model (WSMO, 2005b).

Automatic service composition can be considered from two different points of view. The first perspective concentrates on developing the design time tools. The objective is to build intelligent tools to help composition designer. Sirin et al. (2004) and Sell et al. (2004) show examples of such tools. These tools help the designer by matching input or output types of services in the composition with those in a repository and suggest suitable services to the composition designer. The second point of view is to build a composition automatically on the fly. In an extremely ideal perspective a request for a service can result in building a new composition of the existing services that satisfies the request. The current technology allows us to build template composition that their components can be decided and invoked at the execution time. Both OWL-S and WSMO present the necessary features in their specification to define such templates. These templates consist of components that can result in a discovery process. As we show in our example in the next section, OWL-S SimpleProcess (as well as WSMO Goals) can play the role of such components.

### 7.1. Composition Modeling

To build a service composition, we need (1) a mechanism to describe the order of execution, which is known as *control flow*; (2) a mechanism to describe exchange of data, which is known as *data flow*. The control flow mechanism offers the possibility of performing alternative services based on conditions, repetition and so on. The data flow mechanism should be expressive enough to describe exchange of data among the component services in the composite service, as well as between the component services and the invoker. In the following we describe the composition model in OWL-S with a brief look into the WSMO composition model.

OWL-S Process is classified in three subclasses: AtomicProcess, CompositeProcess and SimpleProcess (Figure 7). An AtomicProcess is, in fact, a stateless service that receives a set of inputs to start the process and produces a set of outputs after it is executed. A CompositeProcess is composed of component Processes. As a result, a



**Figure 7.** Types of OWL-S process.

CompositeProcess can be stateful (Foster et al., 2004). In other words, it may accept inputs and produce output messages at different stages during its execution. The SimpleProcess is not directly executable. It is an abstract description of a process that can be realized and performed by any of the other two types of processes.

OWL-S description of the control flow is based on a set of ControlConstructs, such as a Sequence, IfThenElse, Concurrent, etc. to describe the control flow. It follows the structured design paradigm –similar to that of BPEL4WS (IBM, 2004). Every ControlConstruct is composed of a set of other constructs or processes. A simplified example of OWL-S service composition is presented in Listing 15. The Process is composed of four components, as follows. First service estimates the cost of transport. Then another service receives the credit card information. Third service validates the credit information. Finally, *if* the credit information is valid according to the third service the order is placed. The third process in the example is a SimpleProcess that would be assigned to a concrete (executable) processes at run time. Finding a concrete process by the execution mechanism is, in fact, a services discovery process. This discovery process can be guided by exploring the realizedBy and expandsTo links (in Figure 7) to find the concrete processes.

OWL-S defines the SimpleProcess as a process with a higher level of abstraction (Martin et al., 2004a). This is, indeed, a similar characteristic of Goals and SimpleProcesses. In practice, using a Goal or a SimpleProcess can trigger a discovery mechanism. As shown in Listing 15, this is the main role of the `validate_card` in our example. The use of a SimpleProcess as a component in the composition provides a useful level of dynamism. In such cases, a desired service is to be discovered at the execution time.

**Listing 15.** A simplified example of a composed service for ordering a freight service.

---

```

1 <process:CompositeProcess rdf:ID="Freight_Process">
2 <!--IOPE definitions-->
3 . . .
4 <process:composedOf>
5   <process:Sequence>
6     <process:Components>
7       <process:Perform>
8         <process:AtomicProcess rdf:resource="#Freight_Cost"/>
9       </process:Perform>

```

---

---

```

10 <process:Perform>
11 <process:AtomicProcess rdf:resource="#Credit_Card_Input"/>
12 </process:Perform>
13 <process:Perform>
14 <process:SimpleProcess rdf:resource="#Validate_Card"/>
15 </process:Perform>
16 <process:If-Then-Else>
17 <process:ifCondition>
18 <expr:SWRL-Condition>
19 [SWRL condition: If the credit information is valid?]
20 </expr:SWRL-Condition>
21 </process:ifCondition>
22 <process:then>
23 <process:Perform>
24 <process:AtomicProcess rdf:resource="#Accept_Order"/>
25 </process:Perform>
26 </process:then>
27 </process:If-Then-Else>
28 </process:Components>
29 </process:Sequence>
30 </process:composedOf>
31 </process:CompositeProcess>

```

---

Furthermore, OWL-S provides the means to define data bindings between the component processes. For example, would like to bind the `Freight_costReturn` output from the `Freight_cost` service (Listing 5), to an input of the `Accept_Order` service called `estimated_charge`. To define this binding, the code between lines 23 and 25 (in Listing 15) is rewritten in Listing 16.

**Listing 16.** An example of data binding in OWL-S `CompositProcess`.

---

```

1 <process:Perform>
2 <process:AtomicProcess rdf:resource="#Accept_Order"/>
3 <process:hasDataFrom>
4 <process:InputBinding>
5 <process:toParam rdf:resource="#estimated_charge" />
6 <process:valueSource>
7 <process:ValueOf>
8 <process:theVar rdf:resource="#Freight_costReturn"/>
9 <process:fromProcess rdf:resource=
10 "http://www.example.com/owl-s/freight/Process.owl#Freight_cost"/>
11 </process:ValueOf>
12 </process:valueSource>
13 </process:InputBinding>
14 </process:hasDataFrom>
15 </process:Perform>

```

---

In WSMO, description of service compositions appears in the Interface. The Web Service Interface determines the information needed for executing and interacting with the Web Service. Interface descriptions in WSMO have two major parts namely Choreography and Orchestration. Choreography describes the pattern of interaction with the Web Service. We introduce the Choreography in the next section. For a composite service, Orchestration presents a description of composition of Goals (section 6) or Web Services. It is to provide the necessary details for the execution of all the component services in a composition.

WSMO model for describing composition is still under development (WSMO, 2005b). However, one can immediately notice a difference between the two

specifications; WSMO has only one type of service descriptions as compared to three different types of processes in OWL-S. The general idea of distinction between data flow and control flow is already described in the available documents.

WSMO Mediators also play an important role in WSMO data flow. The data flow in WSMO composition can be defined by Mediators (Hakimpour et al., 2005). Unlike the static binding shown in Listing 16, WSMO allows using a Mediator in bindings –e.g. for changing the currency of the `Freight_costReturn`. Furthermore, one can use WSMO Goals similar to OWL-S SimpleProcess for building dynamic composition. Using Goals in the compositions triggers the discovery process and invocation of the result at the execution time (Hakimpour et al., 2005).

A difference between the two descriptions is that WSMO has only one type of Web Service and it does not distinguish a composite service from an atomic one, as opposed to OWL-S Process model. WSMO keeps the composition description as part of its Interface description (Figure 4). It is one of the basic ideas in WSMO descriptions to hide execution complications from the service consumer. In fact in extreme cases, the Orchestration can be a proprietary piece of knowledge for the provider, and therefore, not accessible to others including the client agent. Yet, it is important for a client to know if the service is stateful or stateless; or how to interact with a service. That piece of knowledge is provided in the WSMO Choreography. The service consumer can obtain the information needed to interact with the service by accessing the Choreography, with no need for the details of the composition.

## 7.2. Choreography in Interaction with Web Services

Interaction with stateless Web Services requires the information that can be found in an interface description such as WSDL. However, the interaction with stateful (Foster et. al., 2004) services, such as most composite services, requires more information than that can be provided by WSDL. A stateful service may receive input messages after it is started and produces output messages before it is finished. An intelligent invoker or a programmer should be ultimately able to gain the knowledge of how to interact with a stateful Web Service.

Choreography determines the pattern of messages exchanged between services (also called business protocol). Two major parts of choreography are the specification of the message types and their order. World Wide Web Consortium (W3C) recommends a model for choreography (W3C, 2004c) that defines three levels for choreography descriptions. The most abstract level contains the same two parts, message types and the message order. Other parts of the choreography descriptions defined by W3C (2004c), such as messages structures, endpoints, used technologies appear only in other levels, neither are they the concern of Semantic Web Services technologies. In fact, such information can be obtained through the grounding mechanisms in both WSMO and OWL-S.

WSMO describes the information required to interact with a Web Service in a discrete part. Choreography appears as part of the Web Service Interface definition in WSMO (2005b). That is, one would not need to access information in the Web Service composition to extract the information about exchange of the messages with the service. Since a composition may include far more information needed for the

service consumer or composition description may be a private property of the service provider.

Whilst Choreography has been addressed by WSMO, OWL-S does not represent such notion in its specification, yet. However, the information about Web Service choreography is embedded in the OWL-S descriptions. Service consumer may extract the information about the pattern of message exchange with a service by referring to the Result and the composition descriptions of an OWL-S Process.

Finally, it is important to note that in spite of the similarity in description of the Choreography in WSMO (2005b) and W3C (2004b); there is major difference in their perspective. WSMO (2005b) presents the Choreography for a type of Web Services, that is, a specification for defining the interaction for Web Services. Alternatively, W3C (2004b) presents a language to describe protocols for interaction between businesses and are not meant to present a Web Service interface. Several business partners may take part in a W3C choreography by assuming the *roles* in the description and perform the relevant activities.

## **8. Summary and Expected Development**

Semantic Web offers a promising approach to overcome some of the shortages or loopholes of today's Web Services technologies. Particularly the service description and subsequently the discovery of Web Services can be improved by using Semantic Web Services technologies. Furthermore, Semantic Web Services present specifications for semantic description of composed services as well as facilitating automatic composition of services. Semantic Web Services enhances Web Service by semantic descriptions that can be related to the existing schematic specifications offered by today's Web Service standards (i.e. WSDL, UDDI).

Semantic Web Services provide following enhancements of Web Services descriptions:

- Information that contribute to the semantic description of Web Services:
  - Input and Output Parameters: definition of the semantics of data in input and output messages using ontologies.
  - Capability Conditions: describing services by defining conditions before and after service execution.
  - Service type: definition of the type of service type either using existing standards for service categorization or using existing potentials in ontologies.
  - Non-functional properties: definition of other relevant information such as service provider and quality of services.
- Grounding of semantic descriptions relates the semantic descriptions to schematic description of services such as WSDL descriptions.
- Intelligent registries are an ultimate goal of Semantic Web Services technology. Intelligent registries provide the possibility:
  - To store the semantic information for service consumers. This information then helps the client agent to have an error free interaction with the Web Service.

- To improve their service matching functionalities from a keyword-based search to semantic based search.
- Semantic Web Services present ontological description of a composed service as well as facilitating automatic composition of services.

The last two topics, namely intelligent registries and automatic composition, although explored to an extent, still demand much further research.

These improvements result not only in an enhanced description of Web Services but also in enrichment of formulating service requests. Eventually, the improvements lead to more intelligent Web Service registries that are the ultimate goal of Semantic Web Service technologies. Intelligent registries will allow:

- The storage of the semantic information for service consumers which can help a client agent to have an error free interaction with the Web Service.
- An improvement of the Web Service matching functionalities from a keyword-based search to semantic based search.

We explore the two major Semantic Web Service specifications, OWL-S and WSMO. We show the similarities and the differences of the two standards as well as their contribution to facilitate Web Service discovery and invocation.

Whether Semantic Web Services technologies and the above specifications will be accepted in the industry or not depends on the usability and the availability of the appropriate tools. Here we identify the tools needed for applying this technology.

- Semantic Description Generator. The first step is to automatically generate semantic descriptions from available documents. There are a few tools at the moment to generate a skeleton for OWL-S descriptions. Such tools can generate a template OWL-S service description (i.e. Profile, Process and Grounding) from WSDL files or JAVA classes. The descriptions should be modified and further information added to them to present the semantics of services.
- Editing tools. Tools to generate, modify and validate semantic descriptions are needed. These tools are used to add further semantics to the template descriptions generated by tools explained above. There are several tools available for both OWL-S and WSMO.
- Registration and Browsing. After building complete and valid semantic descriptions for services we can store the semantic descriptions in existing registries and then retrieve the semantic information. OWL-S presents such tool for UDDI registries. At the moment WSMO descriptions can be stored and browsed in specific WSMO registries.
- Discovery. Both OWL-S and WSMO present repositories with enhanced match making functionality. OWL-S community developed tools integrated with UDDI and WSMO present such functionalities on top of inference engines (at the time of writing).
- Composition tools. Composition tools that can help composers in a semi-automatic way have also been available. These tools can help composers to build the composition by matching the capabilities of the services.

Information about OWL-S tools can be found on: [www.daml.ri.cmu.edu/tools/details.html](http://www.daml.ri.cmu.edu/tools/details.html), [www.daml.org/services/owl-s/tools.html](http://www.daml.org/services/owl-s/tools.html) or [www.mindswap.org/2004/owl-s](http://www.mindswap.org/2004/owl-s). Information about WSMO tools can be found at [www.wsmo.org/wsmo\\_tools.html](http://www.wsmo.org/wsmo_tools.html).

We introduce most of the existing features in the specifications, while the above tools neither exploit all the features nor offer all the corresponding functionalities, at the moment. One important example of such lack of support is in discovery tools. The existing discovery tools match inputs and outputs types, but they do not offer a complete support for matching of preconditions and effects. There is still much work required for development of suitable tools in order to make use of all the aspects of existing specifications, which may in turn result in further enrichment of the specifications.

While a large amount of research work focuses on defining, exchanging, and processing standard descriptions of Web Services at communication and semantic levels, there is still need for more research work on the specifications for the description of the business policies and regulations. It is very important for both the service requesters and providers to communicate and negotiate to set up commercial relationships in order to use services commercially and legally in business applications. For example, a provider may stipulate different payment models for different service usages.

One of the ultimate goals of Semantic Web Services technology is to improve the business negotiation that precedes the actual invocation of Web Services. Semantic Web Services can potentially facilitate (1) informed decisions during the business negotiation and (2) automatic (or semi-automatic) binding of suitable Web Services based on business negotiations. Use of Semantic Web Service descriptions in business negotiations is a major future challenge.

## References

- Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web Services: Concepts, Architectures and Applications*. Springer-Verlag.
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). *The Semantic Web*. *Scientific American*, May issue.
- Berners-Lee, T., (2002). *Web Services*. From <http://www.w3.org/DesignIssues/WebServices.html>.
- Decker, S., Melnik, S., Harmelen, F. V., Fensel, D., Klein, M., Broekstra, J. et al. (2000). *The Semantic Web: The Roles of XML and RDF*. *IEEE Internet computing*, 4(5), 63-74.
- Domingue J., Cabral L., Hakimpour F., Sell D., & Motta E. (2004). *IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services*. *Proceedings of the Workshop on WSMO Implementations*.
- Dong X., Halevy A., Madhavan J., Nemes E., & Zhang J. (2004). *Similarity Search for Web Services*. *Proceedings of 30th VLDB Conference*.

- Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., et al. (2004). Modeling Stateful Resources with Web Services. From <http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf>.
- Guarino, N., editor (1998). *Formal Ontology in Information Systems*. IOS Press.
- Hakimpour F., Sell D., Cabral L., Domingue J., & Motta E. (2005). Semantic Web Service Composition in IRS-III: The Structured Approach, 7th International IEEE Conference on E-Commerce Technology (IEEE CEC'05).
- IBM Corporation (2003). Business Process Execution Language for Web Services. From <http://www-128.ibm.com/developerworks/library/ws-bpel/>.
- Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., et al. (2004a). OWL-S: Semantic Markup for Web Services. From <http://www.daml.org/services/owl-s/1.1/overview/>.
- Martin, D., Burstein, M., Lassila, O., Paolucci, M., Payne, T. & McIlraith, S. (2004b). Describing Web Services using OWL-S and WSDL. From <http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html>.
- McGuinness D. L., & Harmelen F. V., Editors (2004). OWL Web Ontology Language Overview. From <http://www.w3.org/TR/owl-features/>.
- McIlraith, S. A., Son T. C., & Zeng H. (2001). "Semantic Web Services", *IEEE Intelligent Systems*, 46-63.
- OASIS (2004), UDDI Specifications, Version 3. From [http://uddi.org/pubs/uddi\\_v3.htm](http://uddi.org/pubs/uddi_v3.htm).
- Omelayenko, B., Crubézy, M., Fensel, D., Ding, Y., Motta, E., & Musen, M. (2000). Meta Data and UPML. UPML Version 2.0. From <http://www.cs.vu.nl/~upml/upml2.0.pdf>.
- Paolucci, M., Kawamura, T., Payne T.R., & Sycara, K. (2002a). Importing the Semantic Web in UDDI, WES 2002, Springer-Verlag, LNCS 2512, 225–236.
- Paolucci, M., Kawamura, T., Payne T.R., & Sycara, K. (2002b). Semantic matching of Web Services capabilities, Proceedings of the first International Semantic Web Conference, ISWC'02.
- Ponnekanti, S.R., & Fox A. (2002). SWORD: A Developer Toolkit for Web Service Composition. Proceedings of the Eleventh International World Wide Web Conference, WWW2002. USA.
- Sell, D., Hakimpour, F., Domingue, J., Motta E., & Pacheco, R. C. S. (2004). Interactive Composition of WSMO-based Semantic Web Services in IRS-III. AKT Workshop on Semantic Web Services.
- Sirin, E., Hendler, J., & Parsia, B. (2003). Semi-automatic Composition of Web Services using Semantic Descriptions. In Proceedings of the Workshop on Web Services: Modeling, Architecture and Infrastructure, in the 5th International Conference on Enterprise Information Systems, ICEIS-2003, France.
- Sirin, E., Parsia, B., & Hendler J. (2004). Filtering and selecting Semantic Web Services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4), 42-49.

- Sycara, K., Paolucci, M., Ankolekar A., & Srinivasan, N. (2003). Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1(1).
- W3C (2003), SOAP Version 1.2 Part 1: Messaging Framework. From <http://www.w3.org/TR/soap12/>.
- W3C (2004a). Web Services Architecture. From <http://www.w3.org/TR/ws-arch/>.
- W3C (2004b). Web Services Choreography Description Language. From <http://www.w3c.org/TR/ws-cdl-10/>.
- W3C (2004c). Web Services Choreography Model Overview. From <http://www.w3.org/TR/ws-chor-model/>.
- W3C (2004c), XML Schema Part 0: Primer Second Edition. From <http://www.w3.org/TR/xmlschema-0/>.
- W3C (2005). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. From <http://www.w3.org/TR/wsdl20/>.
- WSML (2005). Web Service Modeling Language. From <http://www.wsmo.org/TR/d16/>.
- WSMO (2004a). Web Service Modeling Ontology–Standard. Deliverable 2, version 1.0. From <http://www.wsmo.org/2004/d2/>.
- WSMO (2004b). WSMO Use Case: Virtual Travel Agency. Deliverable 3, version 0.1. From <http://www.wsmo.org/2004/d3/d3.3/>.
- WSMO (2005a), A Conceptual Comparison between WSMO and OWL-S, Deliverable 4, version 0.1. From <http://www.wsmo.org/2004/d4/d4.1/v0.1/>.
- WSMO (2005b). Ontology-based Choreography and Orchestration of WSMO Services. Deliverable 14, Version 0.2. From <http://www.wsmo.org/2004/d14/>.