



## D1.1-Annex A Comparison between OWL-S and WSMO

Nicholas Gibbins, Terry R. Payne, Ahmed Saleh &  
Hai H.Wang (University of Southampton)

### Abstract

EU-IST Specific targeted research project (STREP) IST-2004-026460 TAO  
Deliverable D1.1 -- Annex (WP 1)

Semantic Web Service Research has been recognized as one of the most promising technologies to emerge, exhibiting huge commercial potential, and attracted significant attention from both industry and the research community. Currently there exist several different frameworks and languages for describing a Web service semantically. OWL-S and WSMO are two of very most important approaches.

Not only users, especially newcomers, often have difficulty recognising the differences and choosing the appropriate paradigm for their application, but also a clear separation between the research communities is witnessed. In this paper, we systematically compare WSMO and OWL-S in the context of different views: the service requester, provider and broker based view. It will help users to better understand the strengths and limitations of OWL-S and WSMO. Furthermore, SA-WSDL is another recently emerged new approach for Semantic Web approach and this annex also introduces the main features SA-WSDL.

---

**Keyword list:** Web Services, Semantic Web, Semantic Web Services, OWL-S, WSMO, SAWSDL

---

WP1 **Semantic Web Services Bootstrapping Methodology**

Document ID: TAO/2008/D1.1-annex/v1.0

Nature: **Report**

Dissemination: **PU**

Contractual date of delivery: **31/03/2008**      Actual date of delivery: **11/04/2008**

Reviewed By: **Jesus Martin**

## TAO Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2004-026460.

### University of Sheffield

Department of Computer Science  
Regent Court, 211 Portobello St.  
Sheffield S1 4DP  
UK  
Tel: +44 114 222 1930  
Fax: +44 114 222 1810  
Contact person: Kalina Bontcheva  
E-mail: K.Bontcheva@dcs.shef.ac.uk

### Mondeca

3, cité Nollez  
75018 Paris  
France  
Tel: +33 (0) 1 44 92 35 03  
Fax: +33 (0) 1 44 92 02 59  
Contact person: Jean Delahousse  
E-mail: jean.delahousse@mondeca.com

### University of Southampton

Southampton SO17 1BJ  
UK  
Tel: +44 23 8059 8343  
Fax: +44 23 8059 2865  
Contact person: Terry Payne  
E-mail: trp@ecs.soton.ac.uk

### Sirma Group Corp., Ontotext Lab

Office Express IT Centre, 5th Floor  
135 Tsarigradsko Shosse Blvd.  
Sofia 1784  
Bulgaria  
Tel: +359 2 9768 303  
Fax: +359 2 9768 311  
Contact person: Atanas Kiryakov  
E-mail: naso@sirma.bg

### Atos Origin Sociedad Anonima Espanola

Dept Research and Innovation  
Atos Origin Spain, C/Albarracin, 25, 28037  
Madrid  
Spain  
Tel: +34 91 214 8835  
Fax: +34 91 754 3252  
Contact person: Alberto Capellini  
E-mail: alberto.capellini@atosresearch.eu

### Dassault Aviation SA

DGT/DPR  
78, quai Marcel Dassault  
92552 Saint-Cloud  
Cedex 300  
France  
Tel: +33 1 47 11 53 00  
Fax: +33 1 47 11 53 65  
Contact person: Farid Cerbah  
E-mail: Farid.Cerbah@dassault-aviation.com

### Jozef Stefan Institute

Department of Knowledge Technologies  
Jamova 39  
1000 Ljubljana  
Slovenia  
Tel: +386 1 477 3778  
Fax: +386 1 477 3131  
Contact person: Marko Grobelnik  
E-mail: Marko.Grobelnik@ijs.si

## Executive Summary

Semantic Web Service Research has been recognized as one of the most promising technologies to emerge, exhibiting huge commercial potential, and attracted significant attention from both industry and the research community. Currently there exist several different frameworks and languages for describing a Web service semantically. OWL-S and WSMO are two of very most important approaches.

Not only users, especially newcomers, often have difficulty recognising the differences and choosing the appropriate paradigm for their application, but also a clear separation between the research communities is witnessed. In this paper, we systematically compare WSMO and OWL-S in the context of different views: the service requester, provider and broker based view. It will help users to better understand the strengths and limitations of OWL-S and WSMO. Furthermore, SA-WSDL is another recently emerged new approach for Semantic Web approach and this annex also introduces the main features SA-WSDL.

## Terminology

<b>API</b>	Application Programming Interface
<b>AI</b>	Artificial Intelligence
<b>ASM</b>	Abstract State Machine
<b>DAML</b>	DARPA Agent Markup Language
<b>DL</b>	Description Logic
<b>F-logic</b>	Frame logic
<b>IT</b>	Information Technology
<b>OWL</b>	Ontology Web Language
<b>PDDL</b>	Plan Domain Description Language
<b>PSM</b>	Problem-Solving Methods
<b>RDBMS</b>	Relational Database Management System
<b>RDF</b>	Resource Description Framework
<b>SAWSDL</b>	Semantic Annotations for WSDL and XML Schema
<b>SOA</b>	Service-Oriented Architecture
<b>SOAP</b>	Simple Object Access Protocol
<b>SPA</b>	Service Provider Agent
<b>SSOA</b>	Semantic Service Oriented Architecture
<b>SW</b>	Semantic Web
<b>SWRL</b>	Semantic Web Rule Language
<b>SWS</b>	Semantic Web Services
<b>TAO</b>	Transitioning Applications to Ontologies
<b>UML</b>	Unified Modeling Language
<b>UPML</b>	Unified Problem Solving Method description Language
<b>W3C</b>	World Wide Web Consortium
<b>WS</b>	Web Services
<b>WSDL</b>	Web Services Description Language
<b>WSDL-S</b>	Web Service Semantics
<b>WSML</b>	Web Service Modelling Language
<b>WSMO</b>	Web Services Modelling Ontology
<b>WSMX</b>	Web Service Modelling eXecution environment
<b>XML</b>	eXtensible Mark-up Language

**Contents**

<b>TAO Consortium .....</b>	<b>2</b>
<b>Executive Summary .....</b>	<b>3</b>
<b>Terminology.....</b>	<b>3</b>
<b>Contents .....</b>	<b>4</b>
<b>1 Introduction.....</b>	<b>5</b>
<b>2 OWL-S and WSMO recall .....</b>	<b>5</b>
2.1 OWL-S.....	5
2.2 WSMO .....	8
<b>3 View-based comparison.....</b>	<b>9</b>
3.2 The requester's viewpoint .....	10
3.2.1 Separation between requesters and providers view .....	11
3.2.2 Non-functional property.....	11
3.2.3 Request capacity .....	12
3.2.4 Reusability of Requests.....	12
3.3 Providers' viewpoint .....	13
3.3.1 Capability .....	13
3.3.2 Grounding .....	15
3.4 Brokers' viewpoint .....	16
3.4.1 Choreography.....	16
3.4.2 Orchestration.....	18
3.5 Other differences.....	19
3.5.1 Ontology languages .....	19
3.5.2 Mediator.....	21
3.5.3 Relations with other Semantic Web standards.....	21
3.6 Discussion .....	22
<b>4 SAWSDL.....</b>	<b>23</b>
4.2 SAWSDL Model Reference .....	23
4.3 Annotating WSDL Documents .....	24
4.3.1 Annotating WSDL 2.0 .....	24
4.3.2 Annotating WSDL 1.1 .....	25
<b>5 Discussion.....</b>	<b>25</b>
<b>6 Conclusion .....</b>	<b>26</b>
<b>Bibliography and references .....</b>	<b>27</b>

## 1 Introduction

OWL-S (Ankolenkar et al. 2002) and WSMO (Roman, Keller et al. 2005) are both Web Service modelling languages and many of their constructs are superficially similar; both specify service requests and capabilities. Users, especially newcomers, often have difficulty recognising the differences.

In this annex, we first compare WSMO with OWL-S systematically. In particular, we address the different perspectives on Semantic Web Services and the associated context dependent requirements. We focus on several of the key differences from various viewpoints, i.e. the service requester, provider and broker-based view. A view structures requirements of SWS in arbitrary dimensions and allows us to achieve a more comprehensive comparison of WSMO and OWL-S.

It is not a trivial task to compare OWL-S and WSMO. The major difficulties include:

1. Although both OWL-S and WSMO aim to enhance existing Web services by using Semantic Web technologies, they have had different positioning of roles in realizing this vision. OWL-S aims to only provide a high-level SWS specification. Developers have the freedom to choose various implementation alternatives. However, providing an execution environment for different SWS tasks are one of the important design aims for WSMO. Therefore, WSMO predefines many implementation details.
2. OWL-S is defined as a service specification language, but WSMO is a framework and the service specification is only one part of this framework.
3. Both OWL-S and WSMO are still under development and some aspects of these languages have only been promised for future development. Therefore, some of the comparison can only be based on this forecast.
4. Unlike WSMO, which has a clear scope, OWL-S is more like an academic effort. Many research groups have developed various theories and tools for OWL-S. It is difficult to judge the quality and status of these works.

Our comparison will help users to better understand the strengths and limitations of OWL-S and WSMO.

Furthermore, we also introduce a newly emerged Semantic Web service approach -- Semantic Annotations for WSDL and XML Schema (SAWSDL) (Kopecky, Vitvar et al. 2007). The tools developed in TAO project focus on producing SAWSDL definitions.

## 2 OWL-S and WSMO recall

### 2.1 OWL-S

OWL-S (Ankolenkar 2002), originally known as DAML-S, originates from the need to define Web Services or agent capabilities in such a way that was semantically meaningful (within an open environment), in order to facilitate meaningful message exchange between peers. DARPA's Coordination of Agent Based Systems (CoABS) program had explored the challenge of dynamically discovering and coordinating with agents that provides the necessary capabilities to achieve complex tasks. Such tasks were decomposed into individual agent capabilities through various AI planning

mechanisms, and the CoABS Grid infrastructure was developed to facilitate communication and cooperation between a priori unknown agents within an open environment. However, a major obstacle to achieve the goal of automated interoperability between agents and service providers was that about semantic representation of distributed, heterogeneous components within an open environment. Part of the remit of a follow-on program, the DARPA Agent Markup Language programme, was to investigate how services could be represented within a semantically enabled framework.

The OWL-S (OWL for services) ontologies emerged from this research and are motivated by the need to provide three essential types of knowledge about a service using OWL-DL ontologies, each characterized by the question it answers:

- What does the service provide for prospective clients?
- How is it used?
- How does one interact with it?

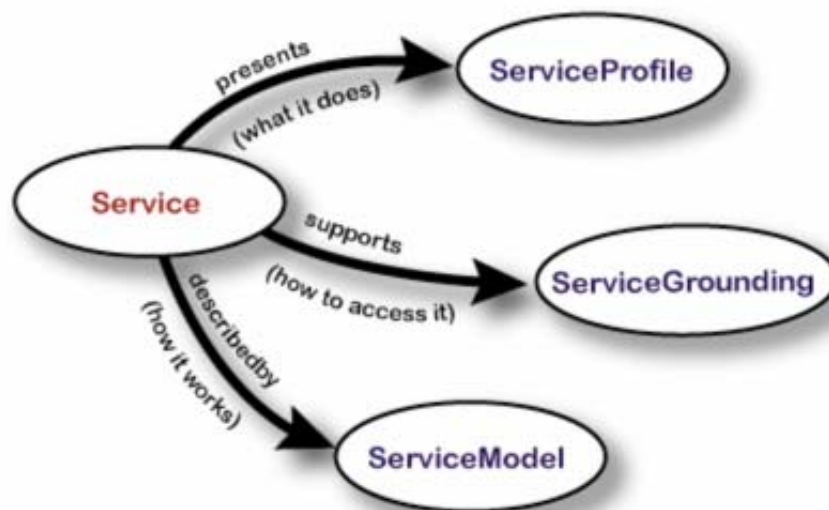


Figure 1: OWL-S top elements

To provide answers to these questions, three essential types of knowledge about a service are needed. Figure 1 shows the structure of OWL-S, and illustrates the high-level relationships between the major ontologies, i.e.:

- How the service is described so that it can be discovered for potential use;
- What are the different processes that constitute a service workflow, and how is the process and data flow defined;
- Where are the interaction steps within the workflow, and how are messages sent to facilitate such communication.

The classes *ServiceProfile*, *ServiceModel*, and *ServiceGrounding* are used in OWL-S to represent the respected aspect of a service.

*Service Profile* The service profile provides a high-level description of the service and is only used for the purposes of service discovery. Service descriptions (and queries) are constructed from a description of functional properties (i.e. inputs, outputs, preconditions, and effects - *IOPEs*), and non-functional properties

(human oriented properties such as service name, etc, and parameters for defining additional meta data about the service itself, such as concept type or quality of service). In addition, the profile class can be subclassed and specialized, thus supporting the creation of profile classes of services.

*Service Model* The service model, known as the OWL-S process model, gives a detailed perspective on how to interact with a service. There are three types of processes in OWL-S model: the atomic, simple and composite process. An atomic process is a single, black box process description with exposed IOPEs (inputs, outputs, preconditions, and effects) and it corresponds to the actions a service can perform by engaging it in a single interaction. Inputs and outputs relate to data channels, where data flows between processes. Preconditions specify facts of the world that must be asserted in order for an agent to execute a service. Effects characterize facts that become asserted given a successful execution of the service, such as the physical side effects that the execution of the service has on the physical world. Simple processes provide a means of describing service or process abstractions; such elements have no specific binding to a physical service, and thus have to be realized by an atomic process (e.g. through service discovery and dynamic binding at run-time), or expanded into a composite process. Composite processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. These process workflows are constructed using a number of different composition and control constructs, including: *Sequence*, *Unordered*, *Choice*, *If-then-else*, *Iterate*, *Repeat-until*, *Repeat-while*, *Split*, and *Split+join*.

*Service Grounding* The grounding of a service specifies the details of how to access the service - details having mainly to do with protocol and message formats, serialization, transport, and addressing. It maps the abstract specification of a service to a concrete specification of those service description elements that are required for interacting with the service. For example, the process model is mapped to a WSDL description of the service, through a thin grounding. Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of XML data types. Additional properties pertaining to the binding of the service are also provided (i.e. the IP address of the machine hosting the service, and the ports used to expose the service).

The design of OWL-S was influenced by diverse work on agents and processes, including:

- Artificial Intelligence research and standardization efforts for planning languages (like PDDL) (Ghallab et al. 1998);
- Programming languages and distributed systems (Milner 1999, Meseguer 1992);
- Emerging standards in process modelling and workflow technology such as Process Specification Language (PSL) (Schlenoff et al. 2000) and the Workflow Management Coalition effort<sup>1</sup>.
- Modelling verb semantics and event structure (Narayanan 1999)
- Action-inspired Web Service markup (McIlraith et al. 2001)

---

<sup>1</sup> <http://www.wfmc.org>

- AI on modelling complex actions (Levesque et al. 1997)
- Agent communication languages (Martin et al. 1999, Finin et al. 1997)

### 2.2 WSMO

The Web Service Modelling Ontology (WSMO) (Roman, Keller et al. 2005) is the other major approach for modelling services semantically. WSMO provides a *framework* for semantic descriptions of Web Services and acts as a meta-model for such Services based on the Meta Object Facility (MOF)<sup>2</sup>. Semantic service descriptions, according to the WSMO meta model, can be defined using the language defined by WSML (Web Service Modelling Language)<sup>3</sup>, which consists of four core elements deemed necessary to support Semantic Web services: *Ontologies*, *Goals*, *Web Services* and *Mediators* (Figure 2). *Ontologies* are described in WSMO at a meta-level. A meta-ontology supports the description of all the aspects of the ontologies that provide the terminology for the other WSMO elements. *Goals* are defined in WSMO as the objectives that a client may have when consulting a Web service. *Web Services* provide a semantic description of services on the Web, including their functional and non-functional properties, as well as other aspects relevant to their interoperation. *Mediators* in WSMO are special elements used to link heterogeneous components involved in the modelling of a Web service. They define the necessary mappings, transformations and reductions between linked elements. In WSMO, four different types of mediators are defined:

*ggMediators* These link two goals, expressing the reduction of a source *goal* into a target *goal*. They can use *ooMediators* to bypass the differences in the terminology employed to define these goals. In addition, WSMO allows linking not only of goals, but also of goals to *ggMediators*, thus allowing the reuse of multiple goals to define a new one.

*ooMediators* These import ontologies and resolve possible representation mismatches between them such as differences in representation languages or in conceptualizations of the same domain.

*wgMediators* They link a Web Service to a goal. This link represents the (total or partial) fulfilment of the goal by the Web Service. *wgMediators* can use *ooMediators* to resolve heterogeneity problems between the Web Service and the goal.

*wwMediators* These link two Web Services, containing the *ooMediators* necessary to overcome the heterogeneity problems that may arise in cases where the services use different vocabularies.

---

<sup>2</sup> <http://www.omg.org/mof/>

<sup>3</sup> <http://www.wsmo.org/TR/d16/d16.1/v0.21/>



**Figure 2: WSMO core elements**

WSMO is mainly influenced by the earlier work on problem-solving methods (PSM). It uses PSM to represent SWS, attaching each WS to a PSM that describes it. A PSM is an abstract implementation of a domain-independent description of reasoning processes, which can be applied to solve tasks in a specific domain. WSMO is based on the UPML (Unified Problem Solving Method description Language)<sup>4</sup>, which was part of a “...framework for developing knowledge-intensive reasoning systems based on libraries of generic problem-solving components...”.

The reasons for many differences between OWL-S and WSMO can be boiled down to their various historical provenances. We will discuss these differences in later sections.

### **3 View-based comparison**

Various stakeholders, e.g. service users, service providers, tool developers etc. have different requirements and understanding of software architecture. A common conclusion is difficult to achieve, because even if the basic intention of stakeholders is similar, different stakeholders need information at different levels of abstraction and aggregation (IEEE, 2000). Therefore, to achieve a more comprehensive comparison of WSMO and OWL-S, we turn to consider the different viewpoints required in the Web Service architecture. According to IEEE Std-1471-2000 (IEEE, 2000), a system view is a representation of a whole system from the perspective of a related set of concerns.

---

<sup>4</sup> <http://www.cs.vu.nl/~upml/> - Unified Problem-solving Method description Language

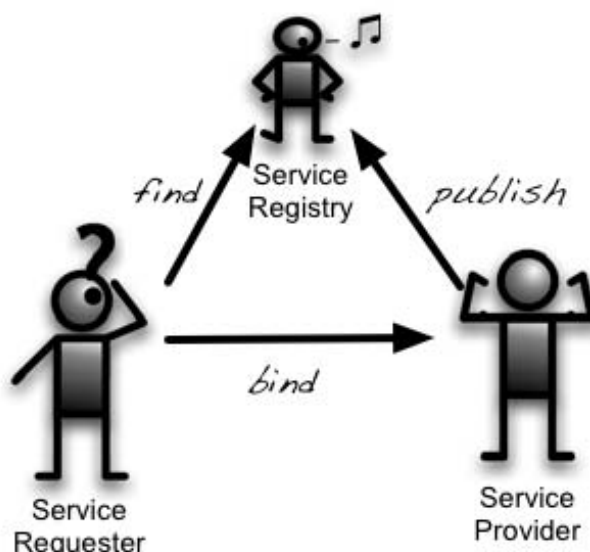


Figure 3: Web Service usage scenario

Semantic Web Services combine the Semantic Web and Web Services technologies. Figure 3 shows the common usage scenario for Web services. It can be defined by three phases: *Publish*, *Find*, and *Bind*; and three entities: the service *requester*, which invokes services; the service *provider* which responds to requests; and the *registry* where services can be published or advertised. A service provider publishes a description of a service that it provides to a service registry. This description (or advertisement) includes a profile on the provider of the service (e.g. company name and address); a profile about the service itself (e.g. name, category); and the URL of its service interface definition. When a developer realizes a need for a new service, he finds the desired service either by constructing a query, or browsing the registry. The developer then interprets the meaning of the interface description (typically through the use of meaningful labels, variable names, comments, or additional documentation) and binds to (i.e. includes a call to invoke) the discovered service within the application they are developing. This application is known as the service requester. At this point, the service requester can automatically invoke the discovered service (provided by the service provider) using some Web Service communication protocols. Our comparison is mainly conducted from the viewpoints of the service requesters, providers and registries.

### 3.2 The requester's viewpoint

After realizing the need for a service, a service client will try to find the desired service by constructing a query, which will be used by service broker/brokers to select a matched service. How the client's request is described is one of the important differences between OWL-S and WSMO.

In WSMO, a special kind of first-class element, *Goal*, is defined to describe users' desires. As a requester, user specifies the objectives for which fulfilment is sought through the execution of Web services from his own point of view. A WSMO *Goal* has a set of predefined non-functional properties and some imported ontologies as the terminology to define the other elements that are part of the goal. A *goal* may also use mediators to reuse already existing goals (for this, *ggMediators* are used) or to resolve

some conflicts between heterogeneous terminologies (for this, *ooMediators* are used). The *requested Capability*, (the WSDL element to define the Web service by means of its functionality) in the definition of the goal describes the capability of the Web services the user would like to have. A *Goal* also defines the *Interface* (the WSDO element to describe how the functionality of the Web service can be achieved) to describe the interface of the Web service the user would like to have and interact with.

A match between a WSMO *Goal* and a WSMO *Service* can be checked in three steps:

*Abstract-level match:* It operates on abstract descriptions of a *Goal* and *Service* without their data being taken into account. This match is mainly defined on set-theoretic relationships between objects satisfying the goal and the objects satisfying the service description. The match could be classified as *exact match*, *subsumption match*, *plug-in match*, *intersection match* or *disjointness*.

*Instance-level Match:* If the goal and the Web service match in previous step, it is further checked if the service can provide a concrete service by consulting the data of the goal and the service.

*Data Matching:* If all data is not available for previous step, the data needs to be obtained from the service.

OWL-S, on the other hand, does not define any specific constructs for describing users' requests. The *profile* is used to specify the object of a service for both user's requests and provider's advertisements. The subsumption between request profiles to advertised profiles plays an important role in the service discovery.

In this section a few important differences between OWL-S profile and WSMO goal are discussed.

### 3.2.1 Separation between requesters and providers view

As can be seen from above, the first main difference between OWL-S and WSMO from the requesters' point of view is that WSMO separates the requester's and provider's view of a service, while OWL-S adopts a unified way, i.e. *service profiles*, for both users to describe the services they desire and for providers to advertise their services. OWL-S's approach seems to be simpler and because that requests and services are described in the same way, some existing OWL reasoners could be directly reused to support service discovering and matching. By contrast, WSMO's approach seems more intuitive to the requester; it is natural that service providers and requesters have different ways of thinking about a service, ways to describe it and terms to annotate it. There exist certain use case such as a service requester (may not be an end user) who knows exactly what his requirements are and wants to plan some complex services. The OWL-S's unified approach is handier for this case.

### 3.2.2 Non-functional property

Both OWL-S and WSMO use non-functional properties to describe the non-functional aspects of a service request. OWL-S only pre-defines three non-functional properties and they are *serviceName*, *textDescription* and *contactInformation*. Users have the

freedom to add new non-functional properties for their needs. Note that the earlier version of OWL-S had defined additional non-functional properties, like *geographicRadius* and *degreeOfQuality* etc., which have been deprecated in the current version of OWL-S. These properties are used to provide extra information about the service, such as its provenance (Paolucci. et al. 2002).

On the other hand, WSMO pre-defines a whole set of core non-functional properties for both *service* and *goal* description. The non-functional properties that can be attached to a goal are similar to the one attached to WSMO services. An extra non-functional property, the *Type of Match*, can be attached to a WSMO *goal*, which represents the type of match desired for a particular goal against a service (under the assumption of a set based modelling this can be an exact match, a match where the goal description is a subset of the Web service description, or a match where the Web service description is a subset of the goal description).

In OWL-S, the non-functional properties are mainly used to provide 'human-readable' information about a request/service; and OWL-S does not constrain the range of non-functional property values. WSMO recommends using widely accepted vocabularies such as the Dublin Core Metadata Element Set<sup>5</sup> or FOAF<sup>6</sup>, encouraging the reuse of existing terminologies and allows users to use logical expressions as the non-functional property values.

The approaches of OWL-S and WSMO to handle non-functional properties could both be beneficial from various prospects. OWL-S is more flexible; in general, a service requester and provider care about different non-functional aspects of the service. Even for the same aspect, the requester and provider are likely to use various terms to express their interests. For example, both service requesters and providers may pay attention to the security of a service. However, users may value their security requirement as *high*, *medium* or *low* and providers may use more technical terms, such as *XML-C14N* etc. At the same time, the flexibility of OWL-S also brings the extra difficulties in service matching for some cases. Since different terminologies may be used by requesters and providers, the service brokers may have to understand the mappings between those terms.

### 3.2.3 Request capacity

WSMO and OWL-S have different ways to describe the functionality of the service being requested. In OWL-S, the service's functionality is described using the so-called *IOPE* model (Inputs, Outputs, Preconditions, Effects), while WSMO defines a special constructor *capability* to describe it. The difference between OWL-S's IOPE service model and WSMO's capability will be discussed further in later section.

### 3.2.4 Reusability of Requests

It would be very costly if every service requester needed to write all his requests from scratch. Both OWL-S and WSMO have defined some mechanisms to reuse already existing requests. OWL-S uses the subsumption relations among *profile* classes to

---

<sup>5</sup> <http://dublincore.org>

<sup>6</sup> <http://xmlns.com/foaf/0.1>

derive new requests from existing requests. By contrast, in WSMO a *goal* may use *ggMediator* to reuse already existing goals. The advantage of OWL-S' approach is that an OWL reasoner can be used for efficient management of service requests by maintaining OWL class hierarchy. However, due to the monotonicity of OWL semantics, the reusability is relatively limited. New requests have to be more specific or narrow than the old requests. *ggMediator* allows a more wide range of goal reusing. Users can only refine a goal specification, but also adjust existing goals by strengthening and weakening.

### 3.3 Providers' viewpoint

The most important aspect of SWS is to describe the Web Services semantically. In OWL-S, a Service Model is used to represent the control and data flow of a service, while WSMO defines a special constructor *webService* to describe the computational entity providing access to services that provide some value in a domain. In order to achieve various designed tasks of SWS, service providers have to provide at least two kinds of information:

- What does the service provide for prospective clients?
- How is it used?

OWL-S and WSMO use different ways to describe these aspects of a service. More detailed comparison is given as follows.

#### 3.3.1 Capability

In OWL-S, a service's functionality is described using the *IOPE* model (Inputs, Outputs, Preconditions, Effects). A service can have any number of *inputs* (including zero), representing the information that is, under some conditions, required for the performance of the process. It can have any number of *outputs*, the information that the process provides to the requester. There can be any number of *preconditions*, which must all hold in order for the process to be successfully invoked. Finally, the process can have any number of *effects*. *Effects* can depend on conditions that hold true of the world state at the time the process is performed.

That OWL-S adopts this *IOPE* model to represent a service capability is inspired by the AI-based action. In OWL-S, each Web service is conceived as an action -- either a *PrimitiveAction* or a *ComplexAction*. Primitive actions are in turn conceived as world altering actions that change the state of the world; as information-gathering actions that change the agent's state of knowledge, so that after executing the action, the agent knows a piece of information; or as some combination of the two.

An advantage of exploiting this action metaphor to describe Web services is that it lets us bring to bear the vast AI research on reasoning about action, to support automated reasoning tasks such as Web service composition. To represent the action metaphor of Web services for reasoning about and planning sequences, a common used specification language for describing planning domains—Plan Domain Description Language (PDDL) has been adopted for specifying each of the Web services in terms of PDDL-inspired *Parameters*, *Preconditions*, and *Effects*. The *Input* and *Output* necessary for automatic Web service execution also play the role of

*Knowledge Preconditions* and *Knowledge Effects* for the purposes of Web service composition and interoperation.

WSMO defines a special constructor *capability* to describe the functionality offered by a given Web service. Capability defines the functional aspects of the offered service, modelled in terms of preconditions, assumptions, postconditions and effects. *Preconditions* of the capability describe the valid states of the information space prior to the service execution. *Postconditions* describe the state of the information space that is guaranteed to be reached after the service execution. *Assumptions* are similar to preconditions, but they define valid states of the world for a correct service execution. *Effects* describe the state of the world that is guaranteed to be reached after executing the service.

There are several interesting differences between OWL-S and WSMO's description of a service capability.

### 3.3.1.1 Precondition and assumption

As we can see that unlike OWL-S, apart from *preconditions*, WSMO defines a set of *assumptions*, which have to be satisfied for the successful provision of the Web service, and it goes beyond the precondition specified by OWL-S. This separation is inherited from UPML (Fensel et al. 2003). Unlike preconditions, assumptions are not necessarily checked by the Web service. *Assumption* is defined to allow an explicit notion of conditions which exist in the real world, but which exist outside the information space. Similarly, WSMO also defines both *postcondition* and *effect*, whereas OWL-S makes no such distinction.

### 3.3.1.2 Logic formulas

Both WSMO and OWL-S use some kinds of logical formulas to represent the different expressions (assumptions, preconditions etc.). Since OWL-S is defined as an OWL ontology itself and it is difficult to represent such logical formulas within RDF and OWL, in OWL-S the concept expression was defined for facilitating the declaration of an “expression language” and “expression body”; it is necessary to annotate expressions with the language they are expressed in. With this approach is possible to treat expressions as literals, either string literals or XML literals. The latter case is used for languages whose standard encoding is in XML, such as SWRL or RDF. The former case is for other languages such as KIF33 and PDDL. On the other hand, WSMO explicitly defines a family of formal languages (WSML), which are used for specifying the logical expressions in WSMO. As before, this difference is due to the differing aims of OWL-S and WSMO; WSMO aims to provide a comprehensive execution environment.

These differences, and among some others, are all due to the fact that WSMO and OWL-S have different positioning of roles in realizing the vision of SWS. OWL-S aims to only provide a high-level SWS specification. Developers have the freedom to choose and develop various ad-hoc implementation alternatives. However, providing an execution environment is one of important design for WSMO. Therefore, WSMO predefines many implementation details.

### 3.3.1.3 Explicitly defined inputs and outputs in OWL-S

The *IOPE* model adopted by OWL-S allows service providers (also requesters) to describe the functionalities of a service using its inputs and outputs, which is more intuitive for most people. The most common used pattern when people are asked to describe the services they provide or need is that “given me ... .. (some resources), I can provide .... ....” or “I want to get ... .. and I can provide ... ..”. For WSMO, the input and output information can only be assumed from a service's *Choreography* definition.

### 3.3.1.4 Dual descriptions of OWL-S service

In OWL-S, the service providers may need to offer more than one functionality description for a service, i.e., in the *profile* and in the *service* model, while in WSMO, the providers only need to provide one description. The OWL-S *profile* of a service provides a concise description of the service to a registry, but once the service has been selected the client will use the service model to control the interaction with the service. Although the profile and the service model play different roles during the transaction between Web services, they are two different representations of the same service and may have different *IOPE* models. For example, it is possible that the Profile would only describe some of the inputs, outputs, preconditions and results that the complete process model does, or describe them more generally than the process model does, especially when the service has the potential to produce a variety of results on request. OWL-S does not dictate any constraint between Profiles and Process Models, so the two descriptions may be inconsistent without affecting the validity of the OWL expression. Still, if the Profile describes a service that is not consistent with the service represented in the Process Model, the interaction will break at some points. By contrast, in WSMO, providers only need to provide a single specification of the service (*WSMO Capacity*), which defines the overall functionality of a Web Service for both internal and external usages. This unification can avoid the extra consistency checking needed between OWL-S profile and OWL-S service model and reduce the providers' workload. However, some internal information may be published unnecessarily.

### 3.3.1.5 Cardinality

Similarly to the WSMO *goal*, a WSMO *service* can only have exactly one *capability*. On the other hand, in OWL-S, a service can have at most one service model defined. An OWL-S service with zero models does not provide automated online access; it exists only for discovery purposes; that is, it exists so as to provide a *Profile*.

### 3.3.2 Grounding

The grounding of a service specifies the details of how to access the service. It is necessary when there is a separation between the conceptual representation of data and its XML realization. For example, there is a traditional Web service, which has its data model, represented by XML and XML schema. Developers can transition this Web service to SWS by wrapping it with some ontological annotations on the data. The mapping between the service's XML data realisation and its ontological representation has to be consistently defined. Irrespective of this, some description of

Web communication points need to be explicitly stated if services are not all situated in a common framework. Grounding is a mapping from an abstract to a concrete specification of those service description elements that are required for interacting with the service.

OWL-S does not include an abstract construct for explicitly describing messages. Rather, the abstract content of a message is specified implicitly by the input or output properties of some atomic process. Thus, atomic processes, in addition to specifying the basic actions from which larger processes are composed, can also be thought of as the communication primitives of a (abstract) process specification. In particular, OWL-S also shows how to use the WSDL and SOAP in crafting a grounding mechanism<sup>7</sup>.

WSMO also provides some guidelines for grounding WSMO service descriptions to WSDL<sup>8</sup>. This has been defined from two aspects -- *data grounding* which defines the mapping between ontological data and its representation as XML messages and *choreography grounding* which describes what messages are supposed to be sent and received between clients and service providers, and the necessary serialization and networking details, i.e. what underlying protocol should be used for passing the messages.

Both OWL-S and WSMO also provide the mapping to SA-WSDL.

### 3.4 Brokers' viewpoint

The service brokers control the usage of the services. *Orchestration* and *choreography* are the most two important aspects for a service execution and we define as follows:

*Orchestration* describes a workflow to support the execution of a specific business process, operation or service; i.e., it describes how Web Services can interact with each other at the message level, including the business logic and execution order of the interactions. These interactions may span applications and/or organizations, and result in a long-lived, transactional, multi-step process model.

*Choreography* defines the sequence (and conditions) under which multiple, cooperating, independent services exchange messages in order to perform a task to achieve a goal state.

#### 3.4.1 Choreography

Choreography is typically associated with the public message exchanges that occur between multiple Web Services, rather than a specific business process that is executed by a single party. Languages such as WS-Choreography (which is based on Pi-Calculus (Milner et al. 1992)) describe the interactions with an invoking party (which might be other Web Services, applications or human beings), and may consist of multiple separate interactions whose composition constitutes a complete

---

<sup>7</sup> <http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/Grounding.owl>

<sup>8</sup> <http://www.wsmo.org/TR/d24/d24.2/v0.1/>

transaction. This composition, along with its message protocols, interfaces, sequencing, and associated logic, is considered to be choreography.

The conceptual model for WSMO choreography is based on Abstract State Machine (Ronchi et al. 2004). ASMs provide high flexibility in modelling systems and are scientifically well-founded. Thus, choreography in WSMO inherits the principles of Abstract State Machines, defining a set of states where each one is described by algebra, and guarded transitions are used to express state changes by means of updates. The primary elements of WSMO Choreography are as follows:

- *State* is described by a set of instances, either explicitly or through a link to an instance store.
- *Guarded Transitions* express the changes of states using rules. As an extension to a “normal” WSMO ontology, an ontology that is used to describe states in WSMO choreography introduces a new non-functional property. When a concept, relation or function in choreography is defined, the attribute mode can be defined as a new non-functional property. It can take one of the following values:
  - *Static* specifies that the extension of the concept, relation or function cannot be changed. The attribute *mode* takes this as the default value unless otherwise stated.
  - *Controlled* specifies that the extension of the concept, relation or function can only be changed by the Web service.
  - *In* specifies that the extension of the concept, relation or function can only be changed by the environment and a grounding mechanism should be specified in order to provide write access for the environment.
  - *Shared* specifies that the extension of the concept, relation or function can be changed by both the environment and the Web service. The grounding mechanism should in this case provide both read and write access to the environment.
  - *Out* specifies that the extension of the concept, relation or function can be changed only by the Web service and a grounding mechanism should be specified in order to provide read access for the environment.

The *signature* of the states of WSMO choreography is defined by the set of valid WSMO identifiers, concepts, relations, functions, and axioms. This signature is the same for all states. The elements that can change and that are used to express different states of choreography are the instances (and their attribute values) of concepts, functions, and relations that are not defined as being static. In conclusion, a specific state is described by a set of explicitly defined instances and values of their attributes or through a link to an instance store.

The guarded transitions express the changes of states by means of rules expressible in the following form:

*if* condition *then* updates

A condition is an arbitrary WSMO axiom formulated in the given signature of the state. The updates consist of arbitrary WSMO Ontology instance statements expressing the changes of the latter's attribute values or class membership.

Since WSMO allows more than one interface to be defined, multiple choreographies can be defined for a Web service and hence multiple modes of interactions may exist for the same Web service.

By contrast OWL-S does not provide an explicit definition of choreography but instead focuses on how the atomic processes are grounded. Grounding in OWL-S is a mapping from an abstract to a concrete specification of those service description elements that are required for interacting with the service. In general, grounding specifies:

- A communication protocol
- Message formats
- Other service-specific details (for example, port numbers used in contacting the service)
- The serialization techniques employed for each semantic type of input or output specified in the *ServiceModel*.

For the point of view of processes, service grounding enables the transformation from inputs and outputs of an atomic process into a concrete atomic process grounding constructs.

### 3.4.2 Orchestration

From a Web Service perspective, an orchestration is a declarative specification that describes a workflow to support the execution of a specific business processes, operation or service; i.e., it describes how Web Services can interact with each other at the message level, including the business logic and execution order of the interactions. These interactions may span applications and/or organizations, and result in a long-lived, transactional, multi-step process model. Languages such as BPEL4W<sup>9</sup> describe both the data and process flow through the workflow (i.e. the pattern of interactions that a Web service agent must follow) as well as notions such as transactions and role-back (in case of service failure).

Although WSMO well recognizes the importance of orchestration for Web services and explicitly defines a special construct named '*orchestration*' for describing which other services have to be used or which other goals have to be fulfilled to provide its functionality, the details of how WSMO orchestrations will be described are not fully defined yet. There are only brief definitions. Similarly to WSMO Choreography, WSMO Orchestration is state-based and consists of the same elements as in WSMO Choreography. However the guarded transitions are extended to take also the following form:

*If condition then mediator*

The difference with respect to choreography is that the rules are extended to support the use of mediators to link the orchestration to other goals or Web services. The mediators used are of the type *wwMediator* or *wgMediator*. If the required Web service is already known, a *wwMediator* is used to link the orchestration to the choreography of the required Web service. If the required Web service is still unknown, then a *wgMediator* is used to link to the goal which expresses what is needed by the orchestration at the given state.

---

<sup>9</sup> <http://www.ibm.com/developerworks/library/specification/ws-bpel/>

By contrast OWL-S does not provide an explicit definition of orchestration but instead focuses on a process based description of how complex Web services invoke atomic Web services (although the OWL-S process model implicitly defines some aspects of orchestration). In the OWL-S service model, there are three types of processes, namely *Atomic*, *Composite* and *Simple*:

- *Atomic* processes correspond to the actions a service can perform by engaging it in a single interaction. They are directly invocable and they take an input message, do something, and then return their output message. For each atomic process, there must be provided grounding, corresponding to a single WSDL operation, that enables a service requester to construct messages to the process from its inputs and deconstruct replies.
- *Composite* processes correspond to actions that require multi-step protocols and/or multiple server actions. They are decomposable into other (non-composite or composite) processes; their decomposition can be specified by using control constructs such as *Sequence* and *If-Then-Else*. Composite processes do not relate to the internal control structure of a service but rather to a behaviour (or set of behaviours) the client can perform by sending and receiving a series of messages. If the composite process has an overall effect, then the client must perform the entire process in order to achieve that effect and to perform a process means that a client must send a series of messages that invoke every step of the process. One crucial feature of a composite process is its specification of how its inputs are accepted by particular subprocesses, and how its various outputs are produced by particular subprocesses. A composite process is invocable only if it bottoms out in atomic processes.
- *Simple* processes provide an abstraction mechanism to provide multiple views of the same process. They are not invocable and are not associated with grounding, but, like atomic processes, they are conceived of as having single-step executions. Simple processes are used as elements of abstraction; a simple process may be used either to provide a view of (a specialized way of using) some atomic process, or a simplified representation of some composite process (for purposes of planning and reasoning).

### 3.5 Other differences

#### 3.5.1 *Ontology languages*

Both OWL-S and WSMO use ontology concepts to annotate knowledge and make it machine understandable, but the ontology defining languages are different. OWL-S is tightly coupled with OWL, while WSMO defined as a conceptual framework aims to support a family of Ontology specification languages, which include Description Logics, First-Order Logic and Logic Programming based ontology language variants. Despite of this, the syntax of WSMO ontology language (WSML) is based on F-logic. A translation is needed in order to use OWL together with WSMO to describe a web service. Furthermore, as far as we know, all most all the WSMO use cases and execution environments have adopted F-Logic based WSML variant as the ontology languages. Therefore, many people regarded Frame logic is the default ontology

language for WSMO. Description Logic (DL) and Frame logic have imposed many different development methodologies and metaphors:

*Permissions vs prohibitions:* In F-logic everything is forbidden until it is permitted by an entry in the template; in OWL everything is permitted until it is prohibited by a constraint (“*restriction*”).

*Composition vs enumeration:* In OWL the definitions are explicit and can be used to compose new terms as needed and classify them automatically (“*post coordination*”); in frames the hierarchy must be enumerated manually in advance (“*pre-coordination*”); if entities are constructed along multiple dimensions (e.g., structure, function, and use), the number of entities to be enumerated can explode exponentially.

*Meta-modelling:* In F-logic, it is possible to refer to a class directly. This is only possible in OWL-Full, which sacrifices the use of the reasoner.

*Difficulty of logic:* Most users find logic hard. OWL is more expressive in many ways than F-logic. However, the logical consequences of many OWL constructs are not obvious to most new users, and more expressivity brings with it a greater possibility of confusion. This is compounded if the results of actions are not immediately apparent.

*Frames are analogous to familiar paradigms:* Frames form templates analogously to the way in which classes in object oriented programs or UML form templates for data structures and so are more familiar to many users than OWL’s logic.

In addition there is a fundamental difference around two assumptions in OWL and F-logic.

*Open vs closed world assumptions:* OWL semantics are explicitly open-world, i.e., negation means provably false. By implication and usage, although not always in the specification, frame systems use closed world semantics, i.e., anything not provably true is taken to be false. In particular, anything not explicitly included in a template, is assumed to be illegal in that template.

*Unique name assumption:* In most frame systems, if two things have different names they are assumed to be different. Often this is extended, at least implicitly, to an assumption that all classes are disjoint unless explicitly stated otherwise. Because OWL is designed for use in an open Web environment, where many different users give names for the same thing, it cannot make either assumption. It is assumed that any two names may represent the same entity unless explicitly stated that they are different and that any two classes may overlap unless they are stated to be disjoint.

For these reasons, experience indicates that most users find F-logic systems more intuitive and easier to use, at least initially and for simpler applications. On the other hand, for large ontologies with multiple inheritances, composition of entities allows ontologies implemented in OWL to be regular and parsimonious whereas for ontologies written in frames and related formalisms, the number of enumerated

entities tends to explode exponentially. In OWL ontologies many operations can be performed automatically by a reasoner; whereas ontologies implemented in frames must be maintained largely manually. If reasoning is required, OWL provides other built in inference (e.g. consistency checking, instantiation reasoning, etc.) through the reasoner, whereas frames use a variety of external query and constraint languages. Where required for use in software, OWL's more rigid semantics and global consistency checking provide more support than is possible using the local reasoning typical of frames and their associated query and constraint languages. Finally, OWL is a W3C standard which may well promote its acceptance in the future. We also note that many researchers have noticed the impropriety of OWL and its open world semantics for specific use cases. Other flavours of OWL like OWL-flight had been proposed, but they are not accepted as standards yet. WSMO also defines some mappings from the F-logic to OWL (DL, Lite and Full), but most of these mapping are only at the syntax level.

### 3.5.2 *Mediator*

Some researchers consider that conceptually support the consideration of mediators as a modelling element is one of big advantages of WSMO over OWL-S. This is still a debatable claim, which depends on if the heterogeneity handling is considered as an architectural issue or service specification issue.

WSMO uses different mediators to link heterogeneous components involved in the modelling of a Web Service in distributed environments. OWL-S, as only a Web service specification ontology, does not consider the heterogeneity problem in the language itself. It assumes that some special agents within the Web Service infrastructure will provide the necessary mediating services. Mediators does not exist in the OWL-S, but it does not prevent users to use other services to achieve the same results. For example, currently there exists many research and tools on OWL ontologies mapping and alignment. These tools can be used to realize the functionality of WSMO *OOMediator*. Paolucci (Paolucci et al, 2004) shows that WSMO mediators can be expressed and discovered nevertheless in OWL-S.

### 3.5.3 *Relations with other Semantic Web standards*

W3C recommends implementing the Semantic Web principles in the layers of Web technologies and standards. Therefore, the application layer should be developed on top of the Ontology layer, which in turn is developed on top of the RDF layer. OWL-S follows this layered approach in a more narrow sense. The current version of OWL-S builds on the Ontology Web Language (OWL) directly, i.e., as one OWL ontology itself. Any OWL-S descriptions of a concrete service are valid OWL ontologies as well. This allows us to directly reuse the existing OWL reasoners to perform certain Semantic Web Service reasoning tasks, such as consistency checking, etc. Furthermore, the integration between OWL-S descriptions with other Semantic Web standards like RDF is also straightforward. On the other hand, OWL has many expressive limitations and the usage of OWL reasoners requires further constraints on the expressiveness of OWL (OWL-Lite or OWL-DL), which is only a very small traceable fragment of first order logic.

Being designed as OWL ontologies, OWL-S descriptions of concrete Web services and the OWL-S language definition itself are inevitably confined by OWL's expressive limitation. As OWL lacks the expressivity to define all the desired properties of OWL-S, such as defining the looping constructs for service processes, default values for inputs and side-effects of services, despite its aim of providing an unambiguous model for defining services, OWL-S users and tool developers have had to rely heavily on English-language documentation and comments to understand the language and interpret its models

Although being ontology-based is one of the important design principles of WSMO, ontologies are only used as the data model throughout WSMO, meaning that all resource descriptions and all data interchanged during service usage are based on ontologies. Without the expressive constraint, WSMO Web service descriptions and WSMO itself can be more completely and clearly described. WSMO has both human readable and XML syntax. Because it does not follow the W3C layer cake strictly, the mapping between WSMO and RDF has to be separately defined, which is also a challenge. Due to the nature of RDF, where every triple can be interpreted separately, it is hard to accurately capture WSMO in RDF. The main reason is that many parts of WSMO descriptions are context-dependent, for example, non-functional properties. In WSMO, URIs are interpreted depending on their context, because their context is always clear. In RDF, there is no such distinction of context and therefore it is very hard to capture WSMO completely in RDF. The RDF representation of WSMO is therefore not completely accurate with respect to standard WSMO. For example, in the RDF representation, one cannot use the same identifier both for an attribute and a non-functional property; one has to reply on users and tool developers to prevent such errors happening<sup>10</sup>.

### 3.6 Discussion

OWL-S and WSMO are two important technologies for Semantic Web Service. They all aim to complement the existing syntactic Web service standards, by providing a conceptual model and language for the semantic markup describing all relevant aspects of general services, which are accessible through a Web service interface. In summary, OWL-S mainly uses the agent planning approach and models Web services as processes, whereas WSMO is mainly based on problem solving techniques and models Web services as state machines. OWL-S is better integrated with the existing Web standards, has defined the process model and grounding mechanism. WSMO provides some relatively mature execution environments; the explicit definitions of *choreography* and *orchestration* enable the better service reuse and composition; also PSM appears to be a more nature choice of formalisms to describe Web Services. Lastly, OWL-S is just a service specification ontology, whereas WSMO is a complete Semantic Web Service framework.

Unfortunately we cannot conclude a clear winner of the two important SWS alternatives. OWL-S and WSMO currently each deliver parts of what users need for effective Web services representation and fail to deliver other parts. We also witness certain separation between SWS communities. Researchers and users are loath to switch to the other – from the point of view of each community it is a switch from a

---

<sup>10</sup> <http://www.wsmo.org/TR/d32/>

methodology with which they are familiar, is easy to use, and delivers what they require to one that they find unfamiliar, difficult to use, and lacking in support for key requirements. However if we are to get the full potential benefit of current technologies, we need a converged framework that embraces both.

Recently there emerged a new Semantic Web Services approach -- Semantic Annotations for WSDL and XML Schema (SAWSDL). We will introduce it in the next chapter.

## 4 SAWSDL

Semantic Annotations for WSDL and XML Schema (SAWSDL) (Kopecky, Vitvar et al. 2007) is the latest standard produced by W3C. Based primarily on the earlier work on WSDL-S, it provides a standard means by which WSDL and XML Schema documents can be related to semantic descriptions. The semantic annotations can be added to various parts of a WSDL document such as input and output message structures, interfaces and operations. The SAWSDL specification is compatible with WSDL 2.0, WSDL 1.1 and XML Schema extensibility frameworks. The annotations on WSDL and XML schema can be used to publish a Web service in a registry, and also to discover, compose and invoke Web services. SAWSDL introduces three new extension attributes for use in WSDL and XML Schema documents, and discusses some of their possible uses. The semantic annotations reference a concept in an ontology or a mapping document. The annotation mechanism is independent of the ontology expression language and this specification requires and enforces no particular ontology language. It is also independent of mapping languages and does not restrict the possible choices of such languages.

### 4.2 SAWSDL Model Reference

One of the major extension attribute defined by SAWSDL is *modelReference*. A model reference may be used with every element within WSDL and XML schema. However, SAWSDL defines its meaning only for *wSDL:interface*, *wSDL:operation*, *wSDL:fault*, *xs:element*, *xs:complexType*, *xs:simpleType* and *xs:attribute*.

Model references can be used to define additional semantics for the various annotated XML Schema components:

- *Annotating XML Simple types:* In the XML Schema or WSDL component model, a non-empty *modelReference* on a top-level simple type is represented as *modelReference* property of the XML Schema Simple Type Definition Schema component or the WSDL Type Definition component; the case of an empty *modelReference* or no *modelReference* at all is represented with a Type Definition that does not have a *modelReference* property. *modelReference* properties are propagated from a simple type definition schema component to all attribute and element declaration schema components that are defined with that type.
- *Annotating XML Complex types:* The complex type can be annotated using *modelReference* at either bottom level or top level. In the bottom level annotation, which is used when the members of a complex type will correspond with the concepts in a semantic model, all the member elements

and attributes in a complex type can be annotated by adding a *modelReference* attribute to the relevant schema element or attribute. The top-level annotation is used to annotate the complex types as a whole. It does not necessarily make any specific statements about the elements or attributes within the complex type. If multiple concepts describe the complex type, all of their URIs can be included in the value of the *modelReference* attribute. A complex type can be annotated at both the top and bottom level. These annotations are independent of each other.

- *Annotating XML Elements:* A non-empty *modelReference* on a top-level element declaration used in WSDL is represented as a *modelReference* property of the WSDL Element Declaration component or the XML Schema Element Declaration Component. The values of the *modelReference* property from the type definition component referenced by an element declaration will be propagated to the element declaration *modelReference* property.
- *Annotating XML Attribute:* A non-empty *modelReference* on a top-level attribute declaration is represented as *modelReference* property of the XML Schema Attribute Declaration Schema component. The values of the *modelReference* property from the simple type definition component referenced by an attribute declaration will be propagated to the attribute declaration *modelReference* property.

SAWSDL also defines two extension attributes, i.e., *liftingSchemaMapping* (defining how an XML instance document conforming to the element or type defined in a schema is transformed to data that conforms to some semantic model) and *loweringSchemaMapping* (defining how data in a semantic model is transformed to XML instance data) that are added to XML Schema element declarations and type definitions, for specifying mappings between semantic data and XML. They are mainly used to address post-discovery issues in using a Web service.

### 4.3 Annotating WSDL Documents

WSDL 2.0 and WSDL 1.1 have a number of differences. Some of these changes include:

- WSDL 2.0 makes *targetNamespace* a required attribute of the definitions element, which allows adding further semantics to the description language.
- WSDL 2.0 removes the message constructs. These are specified using the XML schema type system in the types element.
- WSDL 2.0 does not support for operator overloading.
- WSDL 2.0 renames *PortTypes* to *interfaces*. Support for interface inheritance is achieved by using the extends attribute in the interface element.
- WSDL 2.0 renames *ports* renamed to *endpoints*.

The extension attributes defined by SAWSDL fit within both WSDL 2 and WSDL 1.1.

#### 4.3.1 Annotating WSDL 2.0

*Annotating Interfaces:* A WSDL interface element can be annotated using a *modelReference* to provide a reference to a concept or concepts in a semantic model that describe the Interface. When a new interface extends one or more existing

interfaces, the model references of the extended interfaces do not propagate to the new interface.

*Annotating Operations:* The annotation of the operation element carries a reference to a concept in a semantic model that provides a high level description of the operation, specifies its behavioural aspects, or includes other semantic definitions.

*Annotating Faults:* The annotation of the fault element carries a reference to a concept in a semantic model that provides a high level description of the fault and can include other semantic definitions. The fault annotation does not describe the fault message, which should be annotated in the XML schema.

### 4.3.2 Annotating WSDL 1.1

*Annotating portTypes:* A *portType* is annotated in the same way as WSDL 2.0 interface.

*Annotating Input and Output:* The input and output XML schemas can be annotated in the similar way in WSDL 2.0. In addition, an annotation attribute may be added to a part element (under a message element) to specify an input or output annotation that applies to the entire message part.

*Annotating Faults:* A WSDL 1.1 fault is defined identically to an input or output, i.e. as a fault subelement of the operation element. Annotation of the meaning of the fault needs to be done on the fault element in each operation where it occurs.

*Annotating Operation:* Because operation in WSDL 1.1 does not allow attribute extensibility, an operation is annotated by adding an *attrExtensions* element as a child of the operation element. The *modelReference* attribute of *attrExtensions* specifies the operation annotation.

## 5 Discussion

As we have discussed before, currently there exist three main alternatives for describing a Web service semantically, i.e. OWL-S, WSMO and now SAWSDL. Each delivers parts of what users need for effective Web services representation and fail to deliver other parts. In D1.2, we developed an abstract methodology for the (semi-) automatic construction of Semantic Web Services descriptions resulting from the transitioning of applications. To make the methodology more generic, we try to make it independent and abstracted from the concrete Semantic Web Services description languages. At the same time, the TAO project will also develop a set of tools to assist the transitioning process. For this tool suite, we adopt SAWSDL as the default SWS description language, as opposed to OWL-S or WSMO (as initially envisioned), for the following reasons:

- The task of providing a semantic representation of services becomes considerably simplified, as rather than defining a language that spans across the different levels of the WS stack (including choreography and orchestration both of which now fall out of scope of the project), SAWSDL assumes a much

simpler model (that of just augmenting WSDL), which is more inline with the aims of the TAO project.

- SAWSDL provides a very general annotation mechanism that can be used to refer to any form of semantic markup, and is logic-framework agnostic. The annotation referents could therefore be expressed in OWL, in UML, or in any other suitable language. In contrast, OWL-S would have required the use of a Description Logic model underlying the ontologies (using OWL), whereas WSMO would have required the use of Frame Logic (using WSML). SAWSDL allows us to remain agnostic with respect to the logic model used.
- SAWSDL has tried to maximize the use of available XML technology from XML schema, to XML scripts, to XPath, and attempt to lower the entrance barrier to early adopters.
- From our observation, most of Web services are atomic.
- In August 2007, W3C accepted as recommendation for semantic annotation for WSDL components, while WSMO and OWL-S are still W3C candidate recommendations.

## 6 Conclusion

In this annex, we have presented a systematic comparison between OWL-S and WSMO and also introduced SAWSDL -- a new way to annotate Web Services and XML Schema semantically. We decide that TAO suite will focus on annotating WSDL files using the domain ontology and produce SAWSDL definition.

## Bibliography and references

Ankolenkar, A. a. B. M. a. H. J. R. a. L. O. a. M. D. L. a. M. D. a. M. S. A. a. N. S. a. P. M. a. (2002). DAML-S: Web Service Description for the Semantic Web. Sardinia, Italy

Fensel, D., Motta, E., van Harmelen, F., Benjamins, V.R., Crubezy, M., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., Musen, M., Plaza, E., Schreiber, G., Studer, R., & Wielinga, B. (2003). The Unified Problem-solving Method development language UPML. *Knowl. Inf. Syst.*, 5(1), 83–131.

Finin, T., Labrou, Y., and Mayfield, J. 1997. KQML as an Agent Communication Language. In J. Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.

Ghallab, M. et al. 1998. PDDL-The Planning Domain Definition Language V. 2. Technical Report, report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

IEEE. (2000). IEEE recommended practice for architectural description of software-intensive systems. Technical Report IEEE Std 1471-2000.

Kopecky, J., Vitvar, T., Bournez, C., and Farrell, J. (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*. **11**(2-3): 60--67.

Levesque, H., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R.. 1997. GOLOG: A Logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59-84, April-June 1997.

Martin, D., Cheyer, A., and Moran, D. 1999. The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*, 13(1-2):92-128, 1999.

McIlraith, S. and Fadel, R. 2002. Planning with Complex Actions. *Proc. International Workshop on Non-Monotonic Reasoning (NMR2002)*.

Meseguer, J. 1992. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73-155.

Milner, R., Parrow J., et al. (1992). A Calculus of Mobile Processes Part I and II. *Information and Computation*, 100:1–77

Narayanan, S. 1999. Reasoning About Actions in Narrative Understanding. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI'1999)*, pages 350-357. Morgan Kaufmann Press, San Francisco.

Paolucci, M., Kawamura, T., et al. (2002). Semantic matching of web services capabilities. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, 333–347. Springer-Verlag., UK

## D1.1-Annex A Comparison between OWL-S and WSMO

Paolucci, M., Srinivasan, N, et al. (2004). Expressing WSMO Mediators in OWL-S. In Semantic Web Services Workshop at ISWC 04, 120-134, Springer-Verlag, UK

Roman, D., U. Keller, et al. (2005). "Web Services Modeling Ontology." Journal of Applied Ontology **39**(1): 77-106.

Ronchi, D. and Rocca, D. (2004) Abstract state machines-a method for high-level system design and analysis. Computer Journal, 47:270– 271(2).

Schlenoff, C., Gruninger, M., Tissot, F., Valois, J., Lubell, J., and Lee, J. 2000. The Process Specification Language (PSL): Overview and Version 1.0 Specification. NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD.