WSMO Deliverable

# D4.1 v0.1
# A Conceptual Comparison between WSMO and OWL-S

WSMO Working Draft – January 6, 2005

**Authors:**
Rubén Lara, Axel Polleres, Holger Lausen, Dumitru Roman, Jos de Bruijn, and Dieter Fensel

**Editors:**
Axel Polleres and Rubén Lara

**This version:**
http://www.wsmo.org/2004/d4/d4.1/v0.1/20050106/
**Latest version:**
http://www.wsmo.org/2004/d4/d4.1/v0.1/
**Previous version:**

# Abstract

Web Services have added a new level of functionality on top of the current Web, enabling the use and combination of distributed functional components within and across company boundaries. The addition of semantic information to describe Web Services, in order to enable the automatic location, combination and use of distributed components, is nowadays one of the most relevant research topics due to its potential to achieve dynamic, scalable and cost-effective Enterprise Application Integration and eCommerce. In this context, two major initiatives aim to realise Semantic Web Services by providing appropriate description means that enable the effective exploitation of semantic annotations with respect to discovery, composition, execution and interoperability of Web Services, namely: WSMO and OWL-S. In this deliverable, we conduct a comparison that identifies the overlap and differences between both initiatives in order to evaluate their applicability in a real world setting and their potential to become widely accepted standards.

A preliminary version of this deliverable has been presented at the European Conference on Web Services (ECOWS 2004), Erfurt, Germany, September 27-30, 2004

# Contents

# 1 Introduction

Web Services have added a new level of functionality to the current Web, making the first step to achieve seamless integration of distributed components. Nevertheless, current Web Service technologies only address the syntactical aspects of a Web Service and, therefore, only provide a set of rigid services that cannot adapt to a changing environment without human intervention. The human programmer has to be kept in the loop and scalability as well as economy of Web Services are limited [FB02]. The vision of Semantic Web Services is to describe the various aspects of a Web Service using explicit, machine-understandable semantics, enabling the automatic location, combination and use of Web Services. The work in the area of Semantic Web is being applied to Web Services in order to keep the intervention of the human user to the minimum. Semantic markup can be exploited to automate the tasks of discovering services, executing them, composing them and enabling seamless interoperation between them [The04b], thus enabling intelligent Web Services.

The description of Web Services in a machine-understandable fashion is expected to have a great impact in areas of e-Commerce and Enterprise Application Integration, as it can enable dynamic and scalable cooperation between different systems and organisations. These great potential benefits have led to the establishment of an important research activity, both in industry and academia, which aims at realizing Semantic Web Services. Two major initiatives have to be considered in this context. The first one is OWL-S [The04b], an effort by BBN Technologies, Carnegie Mellon University, Nokia, Stanford University, SRI International and Yale University to define an ontology for semantic markup of Web Services. OWL-S is intended to enable automation of Web Service discovery, invocation, composition, interoperation and execution monitoring by providing appropriate semantic descriptions of services. The second one is the Web Service Modelling Ontology [RLK04], an initiative to create an ontology for describing various aspects related to Semantic Web Services, aiming at solving the integration problem. WSMO is led by the Semantic Web Services working group[1] of the SDK cluster[2], which includes more than 50 academic and industrial partners. Both initiatives have the goal of providing a world-wide standard for the semantic description of Web Services, grounding its application in a real world setting. Due to the wide intended audience of these initiatives, it is essential to identify their overlap and differences in order to evaluate their potential to realize the vision of Semantic Web Services and to be adopted in real applications.

In this context, this paper presents a comparison between the latest versions of the Web Service Modelling Ontology (WSMO) i.e. WSMO v1.0 [RLK04] and the Semantic Web Services ontology OWL-S [The04b] i.e. OWL-S v1.1 Beta. The purpose of the comparison is to identify the commonalities and differences between the two specifications[3]. The paper is organized as follows: Section 2 gives an overview of the core modelling elements introduced in WSMO. In Section 3 the OWL-S modelling elements are introduced and compared to WSMO. The comparison is illustrated with examples taken from the WSMO Use Case Modelling and Testing Working Draft [SLPL04] and the OWL-S specification[4]. The languages used by both initiatives are presented and compared in Section 4. A summary of the main differences between WSMO and OWL-S is presented

[1] http://www.wsmo.org/

[2] http://www.sdk-cluster.org/

[3] A comparison between DAML-S, the predecessor of OWL-S, and WSMF [FB02], the framework which served as a starting point for WSMO, is presented in [Fle02]

[4] http://www.daml.org/services/owl-s/1.1B/examples.html

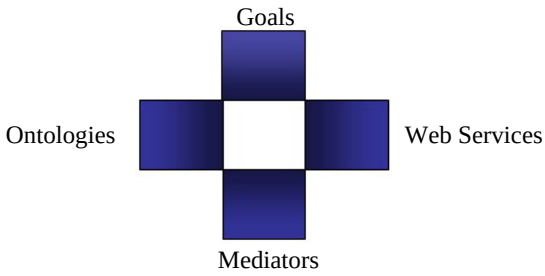


Figure 1: WSMO core elements

in Section 5. Finally, Section 6 gives our conclusions and future work.

# 2 WSMO Elements

Before conducting the detailed comparison between OWL-S and WSMO, we introduce the WSMO modelling elements that will be used in the discussion of Section 3.

WSMO relies on four major components inspired by the conceptual work done in the definition of WSMF [FB02] (see Figure 1), namely:

**Ontologies.** They provide the terminology and formal semantics for describing the other elements in WSMO.

**Goals.** These elements provide the means to specify the requester-side objectives when consulting a Web Service, describing at a high-level a concrete task to be achieved.

**Web Services.** They provide a semantic description of Web Services, including their functional and non-functional properties, as well as other aspects relevant for interoperating with them.

**Mediators.** These modelling elements are connectors that resolve heterogeneity problems in order to enable interoperation between heterogeneous parties.

In addition to these core elements, WSMO introduces a set of **core non-functional properties** that are defined globally and that can be used by all its modelling elements. These properties include the Dublin Core Metadata Element Set [WKLW98] plus a version element, and some of these properties e.g. creator or publisher are recommended to range over instances of the FOAF [BM04] *agent* concept.

An extension of the core non functional properties is defined for Web Services, including aspects such as reliability, security or trust. Although this is the only predefined extension of the core properties, WSMO allows any other extension when necessary for the problem at hand.

## 2.1 Ontologies

Ontologies are described in WSMO at a meta-level. A meta-ontology allows describing all the aspects of the ontologies that provide the terminology for the other WSMO elements. An ontology consists of non-functional properties (from

the set of predefined core properties), imported ontologies, and the definition of concepts, relations, axioms, functions, and instances of the ontology. In addition, it declares the *ooMediators* that are used for importing ontologies for which alignment, merging or transformation problems must be solved during the importation (see Section 2.4). For a detailed description of the meta-ontology we refer to [RLK04].

## 2.2 Goals

Goals are defined in WSMO as the objectives that a client may have when consulting a Web Service. They consist of non-functional properties (from the set of pre-defined core properties), imported ontologies, used mediators, post-conditions and effects.

Postconditions and effects describe the state of the information space and the world, desired by the requester, respectively. Ontologies can be directly imported as the terminology to define the goal when no conflicts need to be resolved. However, if any aligning, merging, or conflict resolution is required, they are imported through *ooMediators*.

## 2.3 Web Services

Several aspects of Web Services are described in WSMO. As mentioned above, specific non-functional properties for Web Services can be used to characterize the non-functional aspects of a given service. The required terminology, as for goals, can be imported directly or via *ooMediators* when conflicts need to be resolved. In addition, the capability and interfaces of the service are described:

### 2.3.1 Capability

The capability defines the functional aspects of the offered service, modelled in terms of preconditions, assumptions, postconditions and effects. It is defined separately from the requester goals, thus distinguishing between the requester and provider points of view.

The *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.

### 2.3.2 Interfaces

The interfaces of a service give details about its operation in terms of its choreography and its orchestration.

The *choreography* provides the necessary information for the user to communicate with the Web Service (i.e. it describes how the service works and how to access the service from the user's perspective), while the *orchestration* describes how the service works from the provider's perspective (i.e. how a service makes use of other Web Service or goals in order to achieve its capability) [RLK04].

A general framework, inspired by Abstract State Machines (ASMs) [B̈99] is being considered for defining WSMO choreographies. However, the details of such ASMs inspired framework have not been published yet.

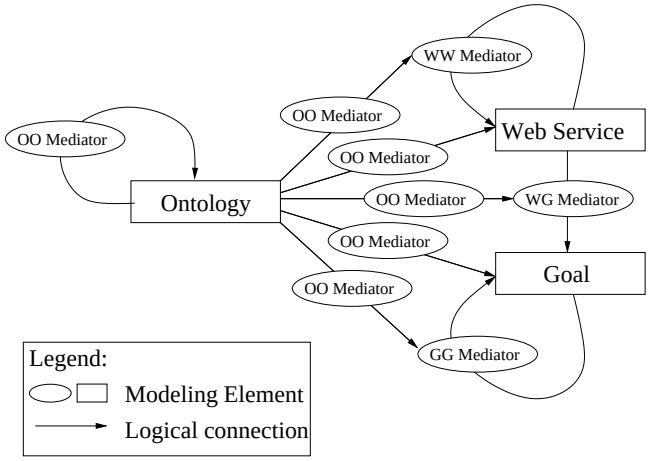


Figure 2: Mediators in WSMO

## 2.4 Mediators

As one of its pillars, WSMO introduces the concept of mediators in order to resolve heterogeneity problems[5]. 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 or reductions between the linked elements. As depicted in Figure 2.4, four different types of mediators are defined, namely:

### 2.4.1 ggMediators

They 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 goals, but also goals to *ggMediators*, thus allowing the reuse of multiple goals to define a new one.

### 2.4.2 ooMediators

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

### 2.4.3 wgMediators

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

### 2.4.4 wwMediators

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

[5]For an introduction to the concept of Mediators, cf. [Wie92]

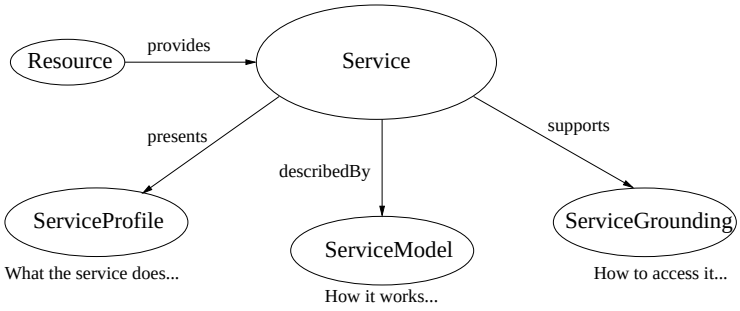


Figure 3: OWL-S upper ontology

# 3 Comparison OWL-S/WSMO

In this section, we introduce each of the description elements of OWL-S and identify the equivalent or similar concepts in WSMO (if any).

OWL-S defines an upper ontology for services with four major elements (see figure 3), namely:

**Service.** This concept serves as an organizational point of reference for declaring Web Services; every service is declared by creating an instance of the Service concept.

**Service Profile.** The profile describes what the service does at a high level, describing its functionality and other non-functional properties that are used for locating services based on their semantic description.

**Service Model.** The model of a service describes how the service achieves its functionality, including the detailed description of its constituent processes.

**Service Grounding.** The grounding describes how to use the service i.e. how a client can actually invoke the service.

## 3.1 OWL-S Service

The *Service* concept in OWL-S links the profile, service model and grounding of a given service through the properties *presents*, *describedBy*, and *supports*, respectively. As an example of the use of the *Service* concept in OWL-S, the BravoAir service[6] from a fictitious airline is modelled as follows:

```
<service:Service rdf:ID="BravoAir_ReservationAgent">
  <service:presents rdf:resource="BravoAirProfile.owl#Profile_BravoAir_ReservationAgent"/>
  <service:describedBy rdf:resource="BravoAirProcess.owl#BravoAir_Process" />
  <service:supports rdf:resource=BravoAirGrounding.owl#Grounding_BravoAir_ReservationAgent"/>
</service:Service>
```

WSMO also provides a direct link between a Web Service, its capability and its interfaces (containing the service choreographies and groundings). However, WSMO explicitly decouples the requester point of view from the provider point of view: goals are defined independently from Web Services and they are linked through *wgMediators*. In addition, the requester and the provider can use different terminologies, as the difference is resolved by the *ooMediators* used by the *wgMediator*.

[6] http://www.daml.org/services/owl-s/1.1B/BravoAirService.owl. This example is part of the OWL-S specification

The example below illustrates what the definition of WSMO Web Services and goals looks like. In the example, different ontologies are imported and used as the terminology for describing the goal and the Web Service. It can be seen that the interface of the Web Service is undefined, as the latest version of WSMO still leaves the description of interfaces open. As mentioned before, *wgMediators* can be defined to explicitly link Web Services and goals, although none is defined in [SLPL04]:

```
goal <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/goal.wsml>>
  nonFunctionalProperties
    dc:description hasValue "Buying a train ticket from Innsbruck to Frankfurt on..."
    [...]
  endNonFunctionalProperties
  importedOntologies <<http://www.wsmo.org/ontologies/dateTime>>
  [...]

webservice <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/ws.wsml>>
  nonFunctionalProperties
    [...]
    dc:description hasValue "Web Service for booking online train tickets for Austria and Germany"
    [...]
  endNonFunctionalProperties
  importOntologies <<http:://www.wsmo.org/ontologies/dateTime>>,
    <<http:://www.wsmo.org/ontologies/trainConnection>>,
    <<http:://www.wsmo.org/ontologies/purchase>>,
    <<http:://www.wsmo.org/ontologies/location>>
  capability _#
    [...]
  interface _#
    nonFunctionalProperties
      dc:description hasValue "describes the Interface of Web Service (not specified yet)"
    endNonFunctionalProperties
    choreography ***
    orchestration ***
```

## 3.2 OWL-S Service Profile

In OWL-S, the service profile describes the intended purpose of the service, both describing the service offered by the provider and the service desired by the requester. This concept is split into two different views in WSMO: the goal and the Web Service capability. While a goal specifies the objectives that a client may have when he consults a Web Service, a Web Service capability defines the service by means of what the service offers to the requester i.e. what functionality the service provides. In the following we go through the details of the OWL-S service profile and describe their counterparts in WSMO.

### 3.2.1 Profile Cardinality

OWL-S links a Web Service to its profile via the property *presents* and its inverse property *presentedBy*. The ontology does not dictate any minimum or maximum cardinality for this property, so the service can present no profile or several profiles. WSMO allows linking a Web Service to none or several goals by using different mediators, and to have an empty capability for the service. WSMO does not allow a service to present more than one capability. However, as the semantics of linking a Web Service with a goal via a *wgMediator* is that the service provides the functionality requested in the goal, providing such link to several goals means that the service can fulfill all of them. This is similar to defining several profiles for an OWL-S Web Service.

### 3.2.2 Profile hierarchy

The profile of a Web Service can be positioned in a hierarchy of profiles[7]. Positioning a given service profile in a profile hierarchy is though optional, and a concrete profile can be directly defined as an instance of the profile class.

As an example, the BravoAir service is categorized as an airline ticketing

[7] http://www.daml.org/services/owl-s/1.1B/ProfileHierarchy.html

service in a hierarchy defining, among others, E-commerce service profiles[8]:

```
<owl:Class rdf:ID="AirlineTicketing">
 <rdfs:subClassOf rdf:resource="#E_Commerce"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#merchandise"/>
   <owl:allValuesFrom rdf:resource="#CommercialAirlineTravel"/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

In the example, airline ticketing profiles are defined as a subclass of E-commerce profiles where the commercialized products are commercial airline travels.

WSMO *ggMediators* allow the definition of goals by refining existing ones. Therefore, it is also possible to describe refinement relations between goals, building a hierarchy of goals through the use of *ggMediators*. Similarly, OWL-S service profiles can be positioned in a previously defined hierarchy, WSMO services can be linked to a goal using a *wgMediator*. In addition, as *ooMediators* can be used both by *ggMediators* and *wgMediators*, the goals in the hierarchy can use different terminologies and a Web Service can use a different terminology from the one of the goal to which it is linked. This is not possible in OWL-S, as the hierarchy of profiles is defined in terms of a common terminology and the Web Services positioned in such a hierarchy have to understand this terminology.

The example of WSMO goals listed above defines two goals separately and links them through a *ggMediator*. This link establishes the relation between the goal of buying a train ticket and the specialized goal of buying a train ticket from Innsbruck to Frankfurt, on a given date, etc. Notice that, as both goals use the same terminology, no *ooMediator* is needed. Although not shown in [SLPL04], a *wgMediator* can be defined to link a given Web Service to any of these goals. We have defined an example *wgMediator* linking the Web Service from [SLPL04] to the goal of buying a specific ticket:

```
goal <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/goal1.wsml>>
  nonFunctionalProperties
    dc:title hasValue "Buying a train ticket online"
    [...]
  endNonFunctionalProperties
  importedOntologies <<http:://www.wsmo.org/ontologies/trainConnection>>,
    <<http:://www.wsmo.org/ontologies/purchase>>,
    <<http:://www.wsmo.org/ontologies/location>>
  [...]

goal <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/goal.wsml>>
  nonFunctionalProperties
    dc:title hasValue "Buying a train ticket from Innsbruck to Frankfurt on..."
    [...]
  endNonFunctionalProperties
  usedMediators
    <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/ggm1.wsml>>
    //all other mediators and ontologies are inherited from the GG Mediator
  importedOntologies
    <<http://www.wsmo.org/ontologies/dateTime>>
[...]

ggMediator <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/ggm1.wsml>>
  nonFunctionalProperties
    dc:title hasValue "GG Mediator that links the general Goal for buying tickets with the concrete Goal
                       for buying a train ticket from Innsbruck to Frankfurt"
    [...]
  endNonFunctionalProperties
  source <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/goal1.wsml>>
  target <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/goal.wsml>>

wgMediator example
  nonFunctionalProperties
    dc:title hasValue "Example of wwMediator, not included in the WSMO use case"
    [...]
  endNonFunctionalProperties
  source <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/ws.wsml>>
  target <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/goal.wsml>>
```

### 3.2.3 Service name, contact, description and category

The OWL-S service profile includes human-readable information, contained in the properties serviceName (of type string; maximum one service name),

[8] http://www.daml.org/services/owl-s/1.1B/ProfileHierarchy.owl

textDescription (of type string; maximum one description) and contactInformation (of class *Actor*[9], including information such as name, phone, fax or e-mail, without any cardinality restriction). A service categorization is also given, although the classification schemas are not fixed and, therefore, the range of this property is not specified[10]. There are no cardinality restrictions for the categorization i.e. a service can be assigned to none or multiple categories in different categorization schemes. The BravoAir example defines:

```
<profile:serviceName>BravoAir_ReservationAgent</profile:serviceName>
<profile:textDescription>This service...</profile:textDescription>
<profile:contactInformation>
  <actor:Actor rdf:ID="BravoAir-reservation">
   <actor:name>BravoAir Reservation department</actor:name>
   <actor:phone>412 268 8780</actor:phone>
   <actor:email>Bravo@Bravoair.com</actor:email>
   [...]
  </actor:Actor>
</profile:contactInformation>
<profile:contactInformation>
  <actor:Actor rdf:ID="BravoAir-information">
   <actor:name>John Doe</actor:name>
   <actor:phone>412 268 8789</actor:phone>
   <actor:email>John_Doe@Bravoair.com</actor:email>
   [...]
  </actor:Actor>
</profile:contactInformation>
<profile:serviceCategory>
 <addParam:NAICS rdf:ID="NAICS-category">
  <profile:value>Airline reservation services</profile:value>
  <profile:code>561599</profile:code>
 </addParam:NAICS>
</profile:serviceCategory>
<profile:serviceCategory>
 <addParam:UNSPSC rdf:ID="UNSPSC-category">
  <profile:value>Travel Agent</profile:value>
  <profile:code>90121500</profile:code>
 </addParam:UNSPSC>
</profile:serviceCategory>
```

This information is expressed in WSMO by using non-functional properties (core properties), such as title, description, identifier, creator, publisher or type, which can be defined for both the goal and the Web Service capability. In the following example, the *dc:title* property plays the role of the OWL-S service name, the OWL-S contact information is captured by the *dc:creator*, *dc:publisher*, and *dc:contributor* properties, the description by *dc:description*, and the category can be described by the *dc:type* property. In addition, WSMO uses a commonly accepted terminology for these properties (the Dublin Core Metadata Element Set [WKLW98]), explicitly encourages the use of the widely accepted FOAF [BM04] terminology as the range for some of these properties, and the contact information is described more accurately, since WSMO distinguishes between creators, publishers and contributors:

```
goal <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/goal.wsml>>
  nonFunctionalProperties
    dc:title hasValue "Buying a train ticket from Innsbruck to Frankfurt on..."
    dc:creator hasValue <<http://www.deri.org/foaf#deri>>
    dc:subject hasValues "Train Tickets", "Online Ticket Booking", "Train trip"
    dc:description hasValue "Express the goal of buying a concrete ticket for a train trip"
    dc:publisher hasValue <<http://www.deri.org/foaf#deri>>
    dc:contributor hasValues<<http://www.deri.org/foaf#stollberg>>,
      <<http://www.deri.org/foaf#lara>>,
      <<http://www.deri.org/foaf#lausen>>,
      <<http://www.deri.org/foaf#polleres>>,
      <<http://www.deri.org/foaf#haller>>
    dc:date hasValue "2004-10-04"
    dc:type hasValue <<http://www.wsmo.org/2004/d2#goals>>
    dc:format hasValue "text/plain"
    dc:identifier hasValue <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20041004/resources/goal.wsml>>
    dc:language hasValue "en-US"
    dc:relation hasValues <<http://www.wsmo.org/ontologies/dateTime>>,
      <<http://www.wsmo.org/ontologies/trainConnection>>,
      <<http://www.wsmo.org/ontologies/purchase>>,
      <<http://www.wsmo.org/ontologies/location>>
    dc:coverage hasValue "ID:7029392 Name:World"
    dc:rights hasValue <<http://deri.at/privacy.html>>
    version hasValue "$Revision: 1.25 $"
  endNonFunctionalProperties
```

Furthermore, non-functional properties can be defined for any of the core WSMO elements, and for other elements such as ontology concepts or attributes,

[9] http://www.daml.org/services/owl-s/1.1B/ActorDefault.owl

[10] Some examples of possible categorization, such as NAICS or UNSPC are given in http://www.daml.org/services/owl-s/1.1B/ProfileAdditionalParameters.owl

while in OWL-S non-functional properties can only be associated with the service profile. For example, in [SLPL04] the core non-functional properties are used to characterize ontologies, Web Services, goals, mediators, concepts and attributes of the ontologies, as well as some other WSMO elements. We show an extract of the definition of non-functional properties for an ontology and one of its concepts below:

```
ontology <<http://www.wsmo.org/ontologies/trainConnection>>
  nonFunctionalProperties
    dc:title hasValue "International Train Connections Ontology"
    dc:creator hasValue <<http://www.deri.org/foaf#deri>>
    dc:subject hasValues {"Train", "Itinerary", "Train Connection", "Ticket"}
    dc:description hasValue "International Train Connections"
    [...]
    dc:type hasValue <<http://www.wsmo.org/2004/d2#ontologies>>
    dc:format hasValue "text/plain"
    [...]
  endNonFunctionalProperties

concept station subconceptOf geo:geographicLocation
  nonFunctionalProperties
    dc:description hasValue "Train station"
  endNonFunctionalProperties
  [...]
```

### 3.2.4 Profile parameters

In addition to the above non-functional properties, the OWL-S profile also includes an expandable list of non-functional properties expressed as *service parameters*. The range for the service parameters is not specified[11]. For example, the BravoAir profile describes:

```
<profile:serviceParameter>
 <addParam:GeographicRadius rdf:ID="BravoAir-geographicRadius">
  <profile:serviceParameterName>BravoAir Geographic Radius</profile:serviceParameterName>
  <profile:sParameter rdf:resource="Country.owl#UnitedStates" />
 </addParam:GeographicRadius>
</profile:serviceParameter>
```

In contrast, WSMO does not offer an explicit list of expandable non-functional properties. Rather, as explained before, it uses a list of core non-functional properties (which already includes some aspects such as geographic or temporal coverage, defined by the *dc:coverage* property, as can be seen in the previous WSMO example). However, these core non-functional properties can be freely extended to include any other properties of interest. This is done, for example, for defining Web Service specific non-functional properties.

### 3.2.5 Functionality description

The OWL-S profile specifies what functionality the service provides. The functionality description is split into the *information transformation* performed by the service and the *state change* as a consequence of the service execution. The former is captured by defining the *inputs* and *outputs* of the service, and the latter is defined in terms of *preconditions* and *effects*. Inputs, outputs, preconditions and effects are normally referred to as IOPEs. In the last version of OWL-S, effects are defined as part of a *result*. The schema for describing IOPEs is not defined in the profile, but in the OWL-S process. Instances of IOPEs are created in the process and referenced from the profile, and it is envisioned that the IOPEs of the profile are a subset of those published by the process [The04b].

*Inputs and Outputs.* OWL-S inputs and outputs describe what information is required and what information is produced by the service. Inputs and outputs are modelled as subclasses of parameter, which is in turn a subclass of SWRL variable [HPSB+04] with a property indicating the class or datatype the values

[11] Examples of such properties are geographic radius or response time (see http://www.daml.org/services/owl-s/1.1B/ProfileAdditionalParameters.owl)

of the parameter belong to. Local variables can also be used, and they are modelled in the ontology as subclasses of parameter. Inputs, outputs, and local variables have as scope the process where they appear. The inputs and outputs defined in the service model are referenced from the profile via the *hasInput* and *hasOutput* properties, respectively, and there are no cardinality restrictions for inputs and outputs. Local variables can be referenced via the *hasParameter* property. For the BravoAir example, inputs and outputs are declared in the profile as follows:

```
<profile:hasInput rdf:resource="BravoAirProcess.owl#DepartureAirport">
<profile:hasInput rdf:resource="BravoAirProcess.owl#ArrivalAirport">
<profile:hasInput rdf:resource="BravoAirProcess.owl#OutboundDate">
<profile:hasInput rdf:resource="BravoAirProcess.owl#InboundDate">
<profile:hasInput rdf:resource="BravoAirProcess.owl#RoundTrip">
<profile:hasInput rdf:resource="BravoAirProcess.owl#AcctName">
<profile:hasInput rdf:resource="BravoAirProcess.owl#Password">
<profile:hasInput rdf:resource="BravoAirProcess.owl#Confirm">
<profile:hasOutput rdf:resource="BravoAirProcess.owl#FlightsFound">
<profile:hasOutput rdf:resource="BravoAirProcess.owl#PreferredFlightItinerary">
<profile:hasOutput rdf:resource="BravoAirProcess.owl#ReservationID">
```

The inputs and outputs are defined in the process[12] as part of the different atomic processes where they appear:

```
<process:Input rdf:ID="DepartureAirport">
 <process:parameterType rdf:datatype="&xsd;anyURI">Concepts.owl#Airport</process:parameterType>
</process:Input>
<process:Input rdf:ID="ArrivalAirport">
 <process:parameterType rdf:datatype="&xsd;anyURI">Concepts.owl#Airport</process:parameterType>
</process:Input>
...
<process:Input rdf:ID="Confirm">
 <process:parameterType rdf:datatype="&xsd;anyURI">Concepts.owl#Confirmation</process:parameterType>
</process:Input>

<process:Output rdf:ID="FlightsFound">
 <process:parameterType rdf:datatype="&xsd;anyURI">Concepts.owl#FlightList</process:parameterType>
</process:Output>
...
<process:Output rdf:ID="ReservationID">
 <process:parameterType rdf:datatype="&xsd;anyURI">Concepts.owl#ReservationNumber</process:parameterType>
</process:Output>
```

The information transformation performed by a service is described in WSMO by using preconditions and postconditions of the service capability. The goal only includes postconditions, defined as the state of the information space that is desired. The service capability defines both preconditions (what the service expects for enabling it to provide its service, defining conditions over the input) and postconditions (what the service returns in response to its input, defining the relation between the input and output). There are no cardinality restrictions for preconditions and postconditions. The example below illustrates the description of the preconditions and postconditions of a capability for a service selling train tickets for Austria and Germany:

```
capability _#
   precondition
     axiom _#
      nonFunctionalProperties
          dc:description hasValue "the input has to be a buyer with a purchase intention for
            an itinerary wherefore the start- and endlocation have to be in Austria
            or in Germany, and the departure date has to be later than the current Date.
            A credit card as payment method is expected."
      endNonFunctionalProperties
      definedBy
        ?Buyer memberOf po:buyer and
        ?Trip memberOf tc:trainTrip[
          tc:start hasValue ?Start,
          tc:end hasValue ?End,
          tc:departure hasValue ?Departure
        ] and
        (?Start.locatedIn = austria or ?Start.locatedIn = germany) and
        (?End.locatedIn = austria or ?End.locatedIn = germany) and
        dt:after(?Departure,currentDate).
   [...]
   postcondition
     axiom _#
      nonFunctionalProperties
          dc:description hasValue "the output of the service is a train trip wherefore
            the start- and endlocation have to be in Austria or in Germany and
            the departure date has to be later than the current Date."
```

[12] http://www.daml.org/services/owl-s/1.1B/BravoAirProcess.owl

```
    endNonFunctionalProperties
    definedBy
      ?Trip memberOf tc:trainTrip[
        tc:start hasValue ?Start,
        tc:end hasValue ?End,
        tc:departure hasValue ?Departure
      ] and
      (?Start.locatedIn = austria or ?Start.locatedIn = germany) and
      (?End.locatedIn = austria or ?End.locatedIn = germany) and
      dt:after(?Departure,currentDate).
[...]
```

*Preconditions and effects.* Preconditions are conditions on the state of the world that have to be true for successfully executing the service. They are modelled as conditions, a subclass of expression. Expressions in OWL-S specify the language in which the expression is described[13] and the expression itself encoded as a (string or XML) literal. Effects describe conditions on the state of the world that are true after the service execution. They are modelled as part of a result. A result has an *inCondition*, a *ResultVar*, an *OutputBinding*, and an *Effect*. The *inCondition* specifies the condition for the delivery of the result. The *OutputBinding* binds the declared output to the appropriate type or value depending on the *inCondition*. The *effects* describe the state of the world resulting from the execution of the service. The *ResultVars* play the role of local variables for describing results. Conditions i.e. preconditions defined in the service model are referenced from the profile via the *hasPrecondition* property, and results via the *hasResult* property, with no cardinality restrictions. The BravoAir example does not specify any precondition, but one result:

```
<profile:hasResult rdf:resource="BravoAirProcess.owl#HaveSeatResult">
```

The result is declared in the service process, as part of the atomic process where it appears. However, the definition of this result is not complete in the example. Therefore, we illustrate the definition of results with a more detailed example taken from [The04b]. This result declares that the effect of the service is that a purchase is confirmed, the object purchased is owned by the requester, and the credit limit of the credit card is decreased by the amount of the purchase. The condition for such effect to happen is that the credit limit of the credit card is bigger or equal to the purchase amount. The output is a confirmation number for the purchase:

```
<process:hasResult>
 <process:Result>
  <process:hasResultVar>
   <process:ResultVar rdf:ID="CreditLimH">
    <process:parameterType rdf:resource="&ecom;#Dollars"/>
   </process:ResultVar>
  </process:hasResultVar>
  <process:inCondition expressionLanguage="&expr;#KIF" rdf:dataType="&xsd;#string">
      (and (current-value (credit-limit ?CreditCard)
                          ?CreditLimH)
                         (>= ?CreditLimH ?purchaseAmt))
  </process:inCondition>
  <process:withOutput>
   <process:OutputBinding>
    <process:toParam rdf:resource="#ConfirmationNum"/>
     <process:valueFunction rdf:parseType="Literal">
      <cc:ConfirmationNum xsd:datatype="&xsd;#string"/>
     </process:valueFunction>
    </process:OutputBinding>
   </process:withOutput>
   <process:hasEffect expressionLanguage="&expr;#KIF" rdf:dataType="&xsd;#string">
       (and (confirmed (purchase ?purchaseAmt) ?ConfirmationNum)
            (own ?objectPurchased)
            (decrease (credit-limit ?CreditCard)
                      ?purchaseAmt))
   </process:hasEffect>
  </process:Result>
</process:hasResult>
```

State change is described in WSMO by using assumptions and effects. The goal only defines effects, as the state of the world that is desired. The capability defines both assumptions (similar to preconditions, but referencing aspects of the state of the world beyond the actual input) and effects (the state of the world

[13] The use of SWRL [HPSB+04], KIF [dpA98] or DRS [McD04] is recommended

after the execution of the service). As for preconditions and postconditions, no cardinality restrictions are placed for assumptions and effects. Below we show the assumptions and effects for the ticket selling service introduced above:

```
capability _#
   [...]
   assumption
     axiom _#
      nonFunctionalProperties
          dc:description hasValue "the credit card has to be valid, i.e. not expired.
              The current date is provided as a built-in functionality
            (currently defined explicitly as built-in function is not available)."
      endNonFunctionalProperties
      definedBy
        ?CreditCard memberOf po:creditCard[
          po:cardholdername hasValue ?someHolder,
          po:creditcardidentifier hasValue ?someIdentifier,
          po:expirydate hasValue ?someExpiration,
          po:globalcreditcardclassificationcode hasValue "Master Card"^^xsd:string
        ]
        and ?someHolder memberOf po:cardHolderName[
            po:freeformtext hasValue "Tim Berners-Lee"^^xsd:string
        ]
        and ?someIdentifier memberOf po:creditCardIdentifier[
          po:proprietaryreferenceidentifier hasValue "5535 4464 6686 7747"^^xsd:string
        ]
        and ?someExpiration memberOf po:expiryDate[
          po:expmonth hasValue "09"^^xsd:integer,
          po:expyear hasValue "2007"^^xsd:integer
        ]
        and (currentDate.date.year < ?CreditCard.expirydate.expyear or
         (currentDate.date.monthOfYear <= ?CreditCard.expirydate.expmonth and
         currentDate.date.year = ?CreditCard.expirydate.expyear)).
   [...]
   effect
     axiom _#
      nonFunctionalProperties
          dc:description hasValue "there shall be a trade for the train trip of the postcondition"
      endNonFunctionalProperties
      definedBy
        ?someTrade memberOf po:trade[
          po:items hasValues ?Trip,
          po:payment hasValue ?acceptedPayment
        ]
        and ?acceptedPayment memberOf po:creditCard .
   [...]
```

The relation between the input and the output, and between the preconditions and the effects of a Web Service have to be captured in order to accurately describe the service functionality. If such relation is not provided, then the service only characterizes the input and the output, but does not model the function that transforms one into the other. This relation can be described in OWL-S when defining the *result* of the execution of the service, where the (logical) relation between the input, output and effects is described. In the example above, the *inCondition* and the *effect* states that the credit limit of the credit card given as input will be decreased if the limit was bigger or equal than the purchase amount before invoking the service. In WSMO, this relation is described in the definition of postconditions and effects, as can be seen in the examples.

Summarizing, the OWL-S service profile can be expressed by the combination of the WSMO goal, the WSMO Web Service capability, and the Web Service non-functional properties. In contrast to OWL-S, WSMO distinguishes the requester and the provider points of view and has a different perspective on how requests are formulated. WSMO goals only describe what results are expected from the service execution, but they do not define what information the requester is willing to provide or what state of the world he can guarantee to hold previous to the service execution. OWL-S follows a different approach, and a request is formulated as the description of a profile which characterizes the service being sought. The approach followed by WSMO seems to be more natural for the requester, who is normally interested in getting some results from a service and not in characterizing what a service looks like. However, real applications will determine which approach is actually more useful in practice.

Regarding the logical language used, how the OWL concepts are used in the OWL-S logical expressions to describe conditions and results is not clearly defined. For example, it is not clear how a KIF expression can use an OWL

concept and what the underlaying semantics is. As will be discussed in Section 4, WSML, the family of WSMO languages, does address this issue.

## 3.3 OWL-S Service Model

In OWL-S, a Service Model represents how the service works i.e. how to interoperate with the service. The service is viewed as a process, and the class *ProcessModel* is the root class for its definition. The process model describes the functional properties of the service, together with details of its constituent processes (if the service is a composite service), describing how to interact with the service. The functionality description contained in OWL-S service models corresponds with the capability of a WSMO Web Service, while the descriptions of how to interact with the Web Service correspond to WSMO choreographies.

In the following, we go through the details of the OWL-S service model and present their counterparts in WSMO.

### 3.3.1 Model cardinality

OWL-S links a Web Service to its model by using the property *describedBy*, and the model to the service by using its inverse property *describes*. A Web Service can be described by at most one service model and it does not impose any minimum cardinality.

In WSMO, a Web Service can have 0 or 1 capability, and 0, 1 or more interfaces. As the interface in WSMO defines the choreography of the service, the same service can operate in different ways, while OWL-S allows only one way to interoperate with the service.

### 3.3.2 Functionality description

The functionality description, as for the service profile, is split into information transformation and state change and is expressed in terms of IOPEs (see Section 3.2). IOPEs are linked to any process via the properties *hasInput*, *hasOutput*, *hasPrecondition*, and *hasResult*, with no cardinality restrictions.

Composite process i.e. processes which contain other processes, can define the functionality of each individual or (partially) aggregated process.

Capabilities in WSMO provide the functional description described in OWL-S using IOPEs of the process. Notice that the OWL-S functionality description in the profile (from the provider point of view) and in the model are merged into a single specification in WSMO: the Web Service capability. However, the WSMO capability only defines the overall functionality of the Web Service; for multi-step services, where each step will provide a particular part of the overall functionality, these partial functionalities are not reflected in the Web Service capability. The concrete specification of WSMO choreography, which is not (yet) publicly available at the time of writing, will clarify how the steps of complex services are described and whether the functionality of each individual step, or groups of such steps, is part of these descriptions.

### 3.3.3 Participants

The definition of a process includes the description of its participants. This is described by the property *hasParticipant* and its subproperties *hasClient* and *hasServer*. However, no information about the participants is defined at the current version of OWL-S, and the purpose of such information is not clear. WSMO does not include information about the participants at its current version, but

it will be most likely included in future versions.

### 3.3.4 Profile/Model consistency

It can be seen above that the functionality description of an OWL-S Web Service is specified both in the service profile and in the service model. Nevertheless, the consistency between the service profile and the service model is not imposed. The descriptions contained in the profile and in the model can be inconsistent without affecting the validity of the OWL expression (although it may result in a failure to provide the service functionality).

In WSMO, as the definition of the choreography of the service is not detailed in the current specification, it is not defined yet what kind of consistency will be enforced between the capability and the choreography. In addition, the OWL-S specification envisions that the set of inputs, outputs, preconditions and effects (IOPEs) of the service profile are a subset of the ones defined by the service model. What kind of expectations on the consistency of the definition of capabilities and choreographies will be introduced by WSMO is not clear at the moment.

### 3.3.5 Atomic processes

OWL-S distinguishes between atomic, simple, and composite processes. OWL-S atomic processes can be invoked, have no subprocesses, and are executed in a single step from the requester's point of view. They are a subclass of *process* and, therefore, they specify their inputs, outputs, preconditions and effects. The BravoAir service describes several atomic processes. The atomic processes for getting flight details and for selecting an available flight are listed below:

```
<process:AtomicProcess rdf:ID="GetDesiredFlightDetails">
 <process:hasInput rdf:resource="#DepartureAirport"/>
 <process:hasInput rdf:resource="#ArrivalAirport"/>
 <process:hasInput rdf:resource="#OutboundDate"/>
 <process:hasInput rdf:resource="#InboundDate"/>
 <process:hasInput rdf:resource="#RoundTrip"/>
 <process:hasOutput rdf:resource="#FlightsFound"/>
</process:AtomicProcess>

<process:AtomicProcess rdf:ID="SelectAvailableFlight">
 <process:hasInput rdf:resource="#FlightsAvailable" />
 <process:hasOutput rdf:resource="#SelectedFlight" />
</process:AtomicProcess>
```

In WSMO, the functionality of an OWL-S atomic process can be defined by the service capability when such process is the only process of the service. However, the details of how to specify an atomic process in the service choreography are not defined at the moment, and if the atomic process is part of a complex service, its individual functionality is not captured by the current specification of WSMO capabilities.

### 3.3.6 Simple processes

OWL-S simple processes are not invocable and they are viewed as executed in a single step. They are used as elements of abstraction, although this kind of processes is not illustrated in the BravoAir example.

The functionality of a simple process, when it is the only process of the service, can be described by a WSMO capability as part of a Web Service description with no grounding information i.e. not invocable. However, as the definition of WSMO choreographies is not finished at the moment, whether this elements of abstraction in the interaction with the service will be present in WSMO is undetermined.

### 3.3.7 Composite processes

OWL-S composite are decomposable into other processes. OWL-S provides a set of control constructs such as *sequence* or *split* (for a complete account of the available control constructs we refer the reader to [The04b]) which are used to define the control flow inside the composite process. In addition to the control constructs, means to declare the data flow between processes are provided in the latest version of OWL-S. Processes are annotated using the *binding* class. A binding is declared at the process consuming data from other processes, which declares what other process and which concrete process parameter the data comes from. The example below defines one of the composite processes of BravoAir for booking a flight. It is a sequence of processes, from which the first one is to perform the login, and the second one is to complete the reservation. The process for completing the reservation takes data from the parent process (the *ChosenFlight* input to the parent process) and uses it as the input for its own *ChosenFlight* input:

```
<process:CompositeProcess rdf:ID="BookFlight">
 <process:composedOf>
  <process:Sequence>
   <process:components>
    <process:ControlConstructList>
     <list:first>
      <process:Perform rdf:ID="PerformLogin">
       <process:process rdf:resource="#LogIn"/>
      </process:Perform>
     </list:first>
     <list:rest>
      <process:ControlConstructList>
       <list:first>
        <process:Perform>
         <process:process rdf:resource="#CompleteReservation"/>
         <process:hasDataFrom>
          <process:Binding>
           <process:toParam rdf:resource="#ChosenFlight"/>
           <process:valueSource>
            <process:ValueOf>
             <process:theVar rdf:resource="#ChosenFlight"/>
             <process:fromProcess rdf:resource="Process.owl#TheParentPerform"/>
            </process:ValueOf>
           </process:valueSource>
          </process:Binding>
         </process:hasDataFrom>
        </process:Perform>
       </list:first>
       <list:rest rdf:resource="ObjectList.owl#nil"/>
      </process:ControlConstructList>
     </list:rest>
    </process:ControlConstructList>
   </process:components>
  </process:Sequence>
 </process:composedOf>
 <process:hasInput rdf:resource="#ChosenFlight"/>
</process:CompositeProcess>
```

WSMO can model OWL-S composite processes by defining complex (multi-step) service choreographies. Nevertheless, in its current state WSMO does not provide any means to define the choreography of the service.

One important difference between OWL-S and WSMO is that the former only defines the externally visible behaviour of the Web Service, while WSMO also models how the service makes use of other services to provide its functionality. While WSMO choreographies describe how to interact with the service from a requester perspective, WSMO orchestrations describe how the service acts as a requester for other services in order to complete the functionality declared in its capability. The orchestration defines what Web Services will be invoked or what goals have to be fulfilled (enabling automatic location of suitable services), together with how to interact with such services. However, a detailed description of WSMO choreographies and orchestrations is not available at the moment.

In addition, although the description of WSMO choreographies is not publicly available yet, they will be based on a general framework, inspired by Abstract State Machines [B̈99] in which existing languages (e.g. BPEL4WS, OWL-S process model) can be easily plugged in. Notice that the use of such

a framework will provide formal semantics to WSMO choreographies, while a formal semantics for the OWL-S process model is still missing.

However, these positive points of WSMO when compared to the OWL-S service model are still to be realized by a concrete specification of WSMO orchestrations and choreographies.

## 3.4 OWL-S Service Grounding

The grounding in OWL-S provides the details of how to access the service, mapping from an abstract to a concrete specification of the service. How a WSMO service is mapped to a concrete specification is undefined at the current version, but it will presumably be part of the choreography definition.

### 3.4.1 Grounding cardinality

OWL-S links a Web Service to its grounding by using the property *supports* and its inverse property *supportedBy*. A Web Service can have multiple groundings and a grounding must be associated with exactly one service. These groundings are associated with the atomic processes defined in the service model, although this association is not described in the model but only in the grounding. Therefore, the groundings for the atomic processes of the model can only be located by navigating from the service model to the service (via the *describes* property), and from there to the service grounding (via the *supports* property).

As mentioned before, WSMO does not define the grounding of services in its current version, but it is planned to allow multiple groundings for a Web Service.

### 3.4.2 Grounding definition

OWL-S does not dictate the grounding mechanism to be used. Nevertheless, the current version of OWL-S provides a predefined grounding for WSDL, mapping the different elements of the Web Service to a WSDL interface. An OWL-S atomic process is mapped to a WSDL operation, and its inputs and outputs to the WSDL input and output message parts, respectively. Such mappings are also established in the WSDL description, using the WSDL 1.1 [CCMW01] extensibility elements. We show an extract of the BravoAir WSDL grounding for the *GetDesiredFlightDetails* atomic process:

```
<grounding:WsdlAtomicProcessGrounding rdf:ID="WsdlGrounding_GetDesiredFlightDetails">
 <grounding:owlsProcess rdf:resource="BravoAirProcess.owl#GetDesiredFlightDetails"/>
  <grounding:wsdlOperation rdf:resource="#GetDesiredFlightDetails_operation"/>
  <grounding:wsdlInputMessage rdf:datatype="&xsd;anyURI">
    BravoAirGrounding.wsdl#GetDesiredFlightDetails_Input
  </grounding:wsdlInputMessage>
  <grounding:wsdlInput>
   <grounding:WsdlInputMessageMap>
     <grounding:owlsParameter rdf:resource="BravoAirProcess.owl#DepartureAirport"/>
     <grounding:wsdlMessagePart rdf:datatype="&xsd;anyURI">
       BravoAirGrounding.wsdl#departureAirport
     </grounding:wsdlMessagePart>
   </grounding:WsdlInputMessageMap>
  </grounding:wsdlInput>
  ...
```

In the WSDL description, the mapping is declared as follows:

```
<operation name="GetDesiredFlightDetails_operation" owl-s-process="BravoAir:#GetDesiredFlightDetails">
 <input message="tns:GetDesiredFlightDetails_Input"/>
</operation>

<message name="GetDesiredFlightDetails_Input">
 <part name="departureAirport" owl-s-parameter="BravoAir:#departureAirport_In" />
 ...
</message>
```

As explained before, WSMO does not offer any grounding mechanism at its current version. However, it is expected that WSMO will not impose any specific grounding.

We conjecture that the current grounding provided by OWL-S is also underdefined with respect to the annotation of arbitrary legacy WSDL files, since a consistent, unambiguous and standardized way to map a WSDL message to an OWL concept and viceversa is not defined. Therefore, the use of different mappings for the same messages and OWL concepts is possible, which can cause ambiguities.

# 4 Logical Languages

A framework for describing the semantics of Web Services needs a solid basis on some logical formalism which allows to express ontological structures in the used terminology, conditions over effects, relations between inputs and outputs, etc. in order to describe the functionality of a service and to allow formal reasoning based on these descriptions. In the following we will shortly summarize and compare the formalisms used in WSMO and OWL-S.

## 4.1 WSMO Languages

As an ongoing effort, the Web Service Modelling Langauge (WSML) Working Group[14] is working on the specification of a family of languages for the specification of Ontologies and Web Services, based on the WSMO conceptual model.

The family of languages, simply called WSML [dBFL+04] considers Description Logics as well as Logic Programming and conceptual modelling as a basis for different language variants.

**WSML-Core** This language is defined by the intersection of Description Logic and Horn Logic, based on [GHVD03]. It has the least expressive power of all the languages of the WSML family and therefore the most preferable computational characteristics. The main features of the language are the support for modelling classes, attributes, binary relations and instances. Furthermore, the language supports class hierarchies, as well as relation hierarchies. WSML-Core provides extensive support for datatypes and datatypes predicates, as well as user-defined datatypes.

WSML-Core is based on a subset of OWL, called OWL$^-$ [dBPLF04a], with a datatype extension based on OWL-E [PH04], which adds richer datatype support to OWL.

**WSML-DL** This language is an extension of WSML-Core, which fully captures the Description Logic $\mathcal{SHOIN}(\mathbf{D})$, which underlies the (DL species of the) Web Ontology Language OWL [DS04]. The language can be seen as an alternate syntax for OWL DL, based on the WSMO conceptual model.

**WSML-Flight** This language is an extension of WSML-Core with several features from OWL Full, such as meta-classes, and several other features, such as

[14] http://www.wsmo.org/wsml/

constraints. WSML-Flight is based on OWL Flight [dBPLF04b], which adds features such as constraints and meta-classes to a subset of OWL DL.

**WSML-Rule** This language is an extension of WSML-Core that supports Horn Logic based on minimal model semantics. Furthermore, the language also captures several extensions developed in the area of Logic Programming, such as default negation, F-Logic [KLW95] and HiLog [CKW93].

**WSML-Full** WSML-Full unifies all WSML variants under a First-Order umbrella with extensions to support specific features of WSML-Rule, such as minimal model semantics and default negation. WSML-Full is of all WSML variants the closest to the WSMO conceptual model. The syntax for WSML-Full is in fact defined as the basic syntax for WSMO [RLK04].

WSML-Core is the basic language on top of which all other variants are layered. Because different modelling styles require different styles of languages, not all WSML variants are compatible. However, WSML-Core provides a basic level of compatibility between the variants, since WSML-Core is a proper subset of all the other variants. Furthermore, WSML-Full unifies all variants under a common umbrella, based on First-Order Logic with non-monotonic extensions.

At the time of writing, the different variants of WSML present different degrees of specification. WSML-Core has been fully specified. Currently, WSML-Full has only an intuitive semantics. The specification of OWL Flight, which is the language underlying WSML-Flight, is already available. However, WSML-Flight itself has not yet been specified. WSML-DL is currently in the WSML framework for theoretical purposes and will be specified if the need for this language arises.

## 4.2 OWL-S Languages

OWL-S is an ontology specified in OWL. Actual OWL-S Web Service specifications are created by subclassing and instantiating the classes of OWL-S. Thus, one can say that the OWL language together with the OWL-S vocabulary makes up the OWL-S Web Service specification language.

However, it has been recognized that OWL alone is not enough for the specification of the behaviour of Web Services. The major problem is that OWL does not allow chaining variables over predicates, which makes it impossible to e.g. specify the relationship between input and output, which is necessary to formally describe the behaviour of any software component [PA97]. Therefore, OWL-S allows the user a choice of different languages for the specification of preconditions and effects. However, the interface between the input and the output, which are described in OWL, and the formulae in the precondition and the effect, is not clear. It is especially important to know how these interact, because it is already possible to specify conditions on the input and the output through complex OWL descriptions.

The OWL-S coalition recommends the use of either SWRL, KIF or DRS for the specification of preconditions and effects.

**SWRL** The Semantic Web Rule Language [HPSB+04] is an extension of OWL, which adds support for Horn rules over OWL DL ontologies. Instead of arbitrary predicates, SWRL allows arbitrary OWL DL descriptions in both the head and the body of the rule, where a unary predicate corresponds with an OWL class

and a binary predicate corresponds with an OWL property. Predicates with higher arity are not allowed. However, n-ary predicates can always be encoded in a description logic knowledge base by "emulating" relation parameters through the introduction of a number of functional properties.

The authors of SWRL have demonstrated that the language is undecidable. This undecidability is mainly caused by allowing existential quantification in the head of the rule (inherited from OWL), combined with chaining variables over predicates (inherited from Horn logic).

**KIF** The Knowledge Interchange Format (KIF) is a standards-proposal from the 1990s for the interchange of knowledge between knowledge bases. The language is constructed in such a way that it can be used to capture the semantics of most knowledge bases. As such, it is an extension of the first-order logic with reification.

KIF currently only has a normal text syntax and thus each KIF expression in an OWL-S description consists of text and thus does not benefit from validation and parsing services offered by XML and RDF parsers/validators.

**DRS** DRS (Declarative RDF System) is an OWL ontology, which provides a vocabulary for writing down arbitrary formulas. DRS does not prescribe the semantics of formulas written down using its vocabulary. Thus, when using DRS to specify Web Services, the user will have to find a way outside the language to agree on the semantics of the description.

Comparing the languages suggestions for WSMO and OWL-S it turns out that while OWL-S aims at combining different notations and semantics with OWL for the description of service conditions and effects, WSMO takes a more cautious approach: The layered logical analysis of decidable semantic fragments combining conceptual modelling with rules in the WSML Core, DL, Rule and Flight species seems more focused than the current suggestions in OWL-S. Combinations with SWRL or the purely syntactic framework of DRS lead to inherent undecidability or leave semantics open from the start. For full expressiveness in WSMO, the user can use WSML-Full, which does not pose any restrictions on the kinds of formulas used for the descriptions. This is similar to the use of KIF in OWL-S. However, it is not clear how the OWL and KIF descriptions interact. Furthermore, the KIF syntax is not integrated in OWL-S, since it is merely a string.

In addition, WSMO follows a language specification similar to MOF (Meta-Object Facility) [The04a], in the sense that the language specification layers are clearly separated. The language in which WSMO is defined corresponds with the meta-meta model layer in MOF. WSMO itself and the WSMO-based specification language WSML constitute the meta-model layer. The actual goal, web service, mediator and ontology specifications constitute the model layer. Finally, the actual data described by the ontologies and exchanged between the Web Services constitute the information layer in MOF.

In OWL-S, the meta-meta model consists of OWL, which is the language in which OWL-S is specified. The meta-model layer consists of OWL-S together with OWL. Then, the model layer in OWL-S consists of subclasses and instances of the OWL-S, again together with OWL, which provides the concepts for input and output specifications. The information layer in OWL-S corresponds with the information layer in WSMO. There is not a clear separation between the different model layers in OWL-S, since OWL is part of the meta-meta model, the meta-model and the model layer. This lack of conceptual separation can lead to

confusion and impedes connection with the Software Engineering community, since MOF is an important standard in that community.

# 5 Summary

In this section, we present a summary of the main differences between WSMO and OWL-S as discussed in the previous sections.

**Separation of provider and requester point of view.** While OWL-S uses a single modelling element for both views, WSMO explicitly reflects this separation by defining goals and Web Service capabilities separately.

**Use of non-functional properties.** A set of core non-functional properties is defined in WSMO, which is accessible to all the WSMO modelling elements, while OWL-S restricts the definition of non-functional properties to the profile description.

**Range of non-functional properties.** WSMO recommends to use widely accepted vocabularies such as the Dublin Core Metadata Element Set or FOAF, encouraging the reuse of existing terminologies, while OWL-S does not explicitly consider such commonly used vocabularies.

**Description of requests.** While WSMO describes the requests in terms of goals i.e. results expected by the requester, OWL-S uses profiles to characterize the service being sought.

**Mediation.** WSMO relies on loose coupling with strong mediation. Different kinds of mediators are used to link together the core WSMO elements, dealing with the heterogeneity problems inherent to a distributed environment. OWL-S does not explicitly consider the heterogeneity problem in the language itself, treating it as an architectural issue i.e. mediators are not an element of the ontology but are part of the underlying Web Service infrastructure [PSS04].

**Description of orchestration.** While WSMO explicitly defines the orchestration of the service, describing what other services have to be used or what other goals have to be fulfilled to provide its functionality, OWL-S does not model this aspect of a Web Service. However, the details of how WSMO orchestrations will be described are not available at the moment of writing.

**Maturity of the externally visible behaviour specification.** The definition of Web Service choreographies in WSMO is still under-specified, while OWL-S provides a detailed description of the service model, allowing the definition of atomic, simple and complex processes, together with their control and data flow.

**Formal semantics for the choreography description.** Although the WSMO choreography specification is not publicly available yet, it will come with a formal semantics, while this is missing in the OWL-S process model.

**Multiple choreographies.** WSMO allows the definition of multiple interfaces and, therefore, choreographies for a Web Service, while OWL-S only allows a single service model for a Web Service i.e. a unique way to interact with it.

**Grounding.** How WSMO will ground Web Services to an invocation mechanism is not yet defined at the time of writing. On the contrary, OWL-S does provide a (non-mandatory) grounding to WSDL. Nevertheless, as explained in Section 3.4, some ambiguities can arise when using such grounding.

**Languages.** While OWL-S aims at combining different notations and semantics with OWL, WSMO provides a family of layered logical languages which combines conceptual modelling with rules. The OWL-S approach presents some problems, as combinations with SWRL or the syntactical framework of DRS lead to undecidability or leaves semantics open. Furthermore, it is not clear how OWL and KIF descriptions interact, and the KIF syntax is not integrated in OWL-S, being treated as a string.

**Language layering.** As explained in the previous section, in WSMO the language specification layers are clearly separated, following a MOF style. OWL-S, on the contrary, lacks a conceptual separation between its different language layers.

# 6 Conclusions and future work

Semantic Web Services are seen as one of the most promising research directions to improve the integration of applications within and across enterprise boundaries. Both WSMO and OWL-S have the aim of providing the conceptual and technical means to realize Semantic Web Services, improving the cost-effectiveness, scalability and robustness of current solutions.

However, the comparison presented in this paper shows that, although the aims of both WSMO and OWL-S are the same, they present some differences in the approach they take to achieve their goals. It can be seen from the summary in Section 5 that OWL-S is more mature in some aspects, such as the definition of the process model and the grounding of Web Services. However, WSMO presents some important advantages when compared to OWL-S: its conceptual model has a better separation of the requester and provider point of view, includes the orchestration of a Web Service enabling the static or dynamic reuse of Web Services to achieve the functionality of a more complex Web Service, it will provide formal semantics for the choreography of Web Services and will allow multiple ways of interacting with a given service, provides a better language layering, and the languages used (although some of them under development) overcome the problems of combining OWL with different rule languages.

In addition, we see the approach followed by WSMO to describe user requests more natural i.e. closer to the user intuition than the OWL-S description of requests in terms of the desired profile. We also conceptually support the consideration of mediators as a modelling element in WSMO instead of as an element of the underlying Web Service infrastructure in OWL-S. As heterogeneity is inherent to the domains of application for Web Services, the modelling of Web Services should include information (if available) to indicate how this heterogeneity is to be solved by the underlying infrastructure. However, these arguments are to be clarified in the context of real use cases.

While WSMO seems to be a conceptually stronger approach than OWL-S, it has to further define some open aspects to be completely usable in real applications. OWL-S, however, has been developed for a longer time, so some aspects are better specified than in WSMO. Another consideration to be taken into account is the tool support for both initiatives: WSMO is being rapidly

supported by different implementations (WSMX[15], IRS-III[16], Semantic Web Fred[17]), while the major OWL-S work in that direction is the DAML-S virtual machine [PASS03].

Our future work will concentrate on following the evolution and further development of OWL-S and WSMO, especially the modelling of real use cases following both paradigms in order to determine their applicability in a real world setting. Additionally, the comparison conducted in this paper will eventually guide a formal mapping between the two specifications when a complete version of WSMO is available and its missing elements are defined in more detail.

# 7 Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperonto, and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate programme.

The editors would like to thank to all the members of the WSMO working group for their advice and input into this document.

# References

[B$\ddot{9}$9] E. Börger. High Level System Design and Analysis using Abstract State Machines. In D. Hutter and W. Stephan and P. Traverso and M. Ullmann, editor, *Current Trends in Applied Formal Methods (FM-Trends 98)*, number 1641 in LNCS, pages 1–43. Springer-Verlag, 1999.

[BM04] D. Brickley and L. Miller. *FOAF Vocabulary Specification*, 2004. Available from http://xmlns.com/foaf/0.1/.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, March 2001.

[CKW93] Weidong Chen, Michael Kifer, and David Scott Warren. HILOG: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, 1993.

[dBFL$^+$04] Jos de Bruijn, Douglas Foxvog, Holger Lausen, Eyal Oren, and Dieter Fensel. The WSML family of languages. Deliverable D16, WSML, 2004. Available from http://www.wsmo.org/2004/d16/.

[dBPLF04a] Jos de Bruijn, Axel Polleres, Rubén Lara, and Dieter Fensel. OWL$^-$. Deliverable d20.1v0.2, WSML, 2004. Available from http://www.wsmo.org/2004/d20/d20.1/v0.2/.

[dBPLF04b] Jos de Bruijn, Axel Polleres, Rubén Lara, and Dieter Fensel. OWL flight. Deliverable d20.3v0.1, WSML, 2004. Available from http://www.wsmo.org/2004/d20/d20.3/v0.1/.

[15] http://www.wsmx.org/
[16] http://kmi.open.ac.uk/projects/irs/
[17] http://www.deri.at/research/projects/swf/

[dpA98] dpANS. KIF. knowledge interchange format: Draft proposed american national standard (dpANS). Technical Report NCITS.T2/98-004, 1998. available from http://logic.stanford.edu/kif/dpans.html.

[DS04] Mike Dean and Guus Schreiber, editors. *OWL Web Ontology Language Reference*. 2004. W3C Recommendation 10 February 2004. Available from http://www.w3.org/TR/owl-ref/.

[FB02] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.

[Fle02] A. Flett. A comparison of DAML-S and WSMF. Technical report, Vrije Universiteit Amsterdam, 2002.

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proc. Intl. Conf. on the World Wide Web (WWW-2003)*, Budapest, Hungary, 2003.

[HPSB+04] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. Available from http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/, May 2004.

[KLW95] Michael Kifer, Geord Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843, 1995.

[McD04] D. McDermott. DRS: A set of conventions for representing logical languages in RDF. Available from http://www.daml.org/services/owl-s/1.1B/DRSguide.pdf, January 2004.

[PA97] J. Penix and P. Alexander. Towards automated component adaptation. In *Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering*, June 1997.

[PASS03] Massimo Paolucci, Anupriya Ankolekar, Naveen Srinivasan, and Katia Sycara. The DAML-S virtual machine. In *International Semantic Web Conference (ISWC 2003)*, 2003.

[PH04] Jeff Z. Pan and Ian Horrocks. OWL-E: Extending OWL with expressive datatype expressions. IMG Technical Report IMG/2004/KR-SW-01/v1.0, Victoria University of Manchester, 2004. Available from http://dl-web.man.ac.uk/Doc/IMGTR-OWL-E.pdf.

[PSS04] Massimo Paolucci, Naveen Srinivasan, and Katia Sycara. Expressing WSMO mediators in OWL-S. In *Semantic Web Services Workshop at ISWC 2004*, 2004. To appear.

[RLK04] Dumitru Roman, Holger Lausen, and Uwe Keller, editors. *Web Service Modeling Ontology (WSMO)*. 2004. WSMO Working Draft D2v1.0, September 2004. Available from http://www.wsmo.org/2004/d2/v1.0/.

[SLPL04] Michael Stollberg, Holger Lausen, Axel Polleres, and Rubén Lara, editors. *WSMO Use Case Modeling and Testing*. 2004. WSMO Working Draft D3.2v0.1, July 2004. Avalilable from http://www.wsmo.org/2004/d3/d3.2/v0.1/.

[The04a] The Object Management Group. Meta-object facility. Technical report, 2004. Available from http://www.omg.org/technology/documents/formal/mof.htm.

[The04b] The OWL Services Coalition. OWL-S 1.1 beta release. Available from http://www.daml.org/services/owl-s/1.1B/, July 2004.

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[WKLW98] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. *RFC 2413 - Dublin Core Metadata for Resource Discovery*, September 1998.