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 document, we provide a mapping between OWL-S and WSMO descriptions in order to get a better understanding of what is the relation between them and to enable the interoperability between OWL-S and WSMO services.

Table of contents

1. Introduction
2. The WSMO and OWL-S Web Service Description Languages
   2.1 Web Service Modeling Ontology (WSMO)
   2.2 Web Ontology Language for Services (OWL-S)
3. Mapping from OWL to WSML
4. Mapping of Logical Expressions
   4.1 Notations for expressing rules in OWL-S and WSML
   4.2 SWRL FOL to WSML
   4.3 WSML to SWRL FOL
5. Conceptual Model Mapping
   5.1 OWL-S to WSMO
      5.1.1 Upper Service Mapping
      5.1.2 Service Profile Mapping
      5.1.3 Process Model Mapping
      5.1.4 Grounding Mapping
   5.2 WSMO to OWL-S
6. APIs
   6.1 WSMO4J
   6.2 OWL API
   6.3 OWL-S API
7. Mapping Tool
8. Conclusions and Further Directions
References
Acknowledgement

1. Introduction
WSMO and OWL-S are the main specifications for describing Semantic Web Services. Both of these descriptions aim at providing automatic discovery, composition, invocation and interoperation of Web Services. The mapping which we intend to provide is based on WSMO 1.0 [Roman et al., 2004] and OWL-S 1.1 [Martin et al., 2004]. Furthermore, the specification of some aspects of WSMO such as choreography and orchestration are still in progress and, hence, our mapping is not complete.

We will address the mapping specification in a step-by-step manner. In Section 2 we start by providing an overview of both WSMO and OWL-S Web Service descriptions in order to give the reader a general idea of both. In Section 3, we map the foundational languages of both specifications. WSMO defines the WSML family of languages [De Bruijn et al., 2004], and OWL-S relies on the use of OWL [Dean et al., 2004]. In Section 4 we describe how logical expressions are defined in OWL-S and we provide a bidirectional mapping between them and the corresponding ones in WSMO. In Section 5 we map the conceptual models of the two specifications by taking one by one each element (and sub-elements) in OWL-S and map them to the corresponding ones in WSMO (Section 5.1) and the other way around (Section 5.2). In Section 6 we present an overview of the APIs available which may be used in the tool development. We finally present the mapping tool in Section 7 and present our conclusions and future work in Section 8.

2. The WSMO and OWL-S Web Service Description Languages

Although the aim of both WSMO and OWL-S is to enable users and organisations to specify semantic information about their services, they differ in their characteristics. In this section, we will give an overview of both specifications, for a detailed description we refer the reader to [Roman et al., 2004] and [Martin et al., 2004].

2.1 Web Service Modeling Ontology (WSMO)

WSMO is based on the Web Service Modeling Framework (WSMF) [Fensel and Bussler, 2002]. The core concepts are Ontologies, Goals, Web Services and Mediators as shown in Figure 1 [Roman et al., 2004].

![Figure 1. WSMO Elements](image)

Ontologies serve to express information in a formal manner and also to link machine and human terminologies. The terminology used by every WSMO element is provided by ontologies, which consist of the following parts: Non-Functional Properties, Imported Ontologies, Used Mediators, Axioms, Concepts, Relations, Functions and Instances. Non-Functional properties specify information that do not apply to the functionality of the particular service. The elements that make up Non-Functional Properties are based on Dublin Core metadata [Weibel et al., 1998] except for the "version" element. Ontologies can import other ones or use mediators in order to resolve conflicts between the respective ontologies. For details about the modelling of ontologies in WSMO we refer the reader to [Roman et al., 2004].

Goals express what the user wants and they are decoupled from Web Service capabilities (which describes what the service provides). This is essential since the requester (which might be either a human or an agent) can create a goal request independently from any Web Service description. A goal is described by Non-Functional Properties, Imported Ontologies, Used Mediators, Postconditions and Effects. A Postcondition defines the state of the information space which is desired. An Effect describes the state of the world desired after the service provision.

Web Services are described by Non-Functional Properties, Imported Ontologies, Used Mediators, Capability and Interfaces. In addition to the core non-functional properties, a Web Service can define its specific ones such as quality of service, performance, security and robustness, financial information, etc. A Web Service must have exactly one capability: describing what the service does in terms of preconditions, postconditions, assumptions and effects. Preconditions describe what the service expects for enabling it to provide its service. Postconditions describe the state of the information space that will be reached by executing the service. Assumptions constrain the set of states of the world to the set of valid starting states. The service is guaranteed to give the described functionality only if the specified assumptions hold. Finally, effects in a web service capability describe the state of the world after the service is executed. The Interface of the service describes its functionality in terms of the Choreography and Orchestration. The Choreography defines the behaviour of the service and this is done using Abstract State Machines as the underlying formalism [Roman et al., 2004]. The Orchestration defines how the overall functionality of the service is achieved by the cooperation of other WSMO service providers, that is, it describes how the service behaves from the provider's point of view.

Mediators in WSMO are used to solve heterogeneity problems between different entities. In WSMO 1.0, they are described by non-functional properties, imported ontologies, source and target components, mediation service and used mediators. There are four different types of mediators, namely: ooMediators, ggMediators, wgMediators and wwMediators. ooMediators are used to solve terminological problems between different ontologies. ggMediators are used to link a goal to another one, defining the refinement of the former by the latter. wgMediators link Web Service capabilities to goals that the web service (totally or partially) fulfills. Finally, wwMediators link different services and resolve protocol and process differences.

2.2 Web Ontology Language for Services (OWL-S)

OWL-S specifies a set ontologies based on OWL [Dean et al., 2004] which are used to describe the different aspects of a Semantic Web Service. We will refer to the latest release of OWL-S i.e. version 1.1 [Martin et al., 2004]. The primary aim of OWL-S is to support automatic service discovery, invocation, composition and interoperer. The set of ontologies to describe an OWL-S service include an upper Service ontology, Service Profile, Service Model and Service Grounding. The upper Service ontology links to the...
other constituent parts of the OWL-S description (Figure 2) and for the purpose of this mapping we are assuming that a full OWL-S
description is present, that is, it includes all of the Upper Service, Service Profile, Service Model and Service Grounding ontologies.

The Service Profile describes what the service does and is used to advertise the service. It includes non-functional properties
about the service. The serviceName property contains the name of the service. The textDescription provides a
human-readable description of the service. The contactInformation property does not have any range restriction, but it can be
restricted by the user to use any relevant description system. However, an extra Actor Ontology has been defined which is also used in the BravoAir example given by the OWL-S Coalition. The Profile exposes the service’s functionality in terms of its Inputs, Outputs, Preconditions and Effects (commonly referred to as IOPEs). These are referenced from the Profile ontology to the Service Model since the Profile does not define a schema to create instances of such elements. Furthermore, it is recommended that only a subset of the IOPEs is referenced from the profile (however there is no real restriction in the Profile ontology for such a scenario). This restriction on referencing IOPEs occurs when the service provider does not want to advertise all the operations of the service. Two other attributes of the Profile include serviceParameter and serviceCategory. The serviceParameter allows to point to a parameter and to specify also the value of the parameter within some OWL ontology. The serviceCategory allows to specify the categories to which the service pertains. This may be carried out on the basis of some classification that maybe outside OWL-S and even OWL requiring some specialized reasoning engine.

The Service Model specifies how the service works with respect to its preconditions, effects, inputs, outputs and processes. The key idea behind the Service Model is to model a service as a set of processes (in fact the class Process is a subclass of ServiceModel). A process in OWL-S is not a program to be executed but rather it is a specification of the way a client may interact with a service. There are three different types of processes, namely, Atomic, Composite and Simple. All of these processes are a subclass of Process. An Atomic process is a one-step process i.e. it gets some input and provides some outputs and effects. A Composite process specifies a workflow in order to perform some set of transactions using other processes. An Atomic process is an abstract view of either a composite or a simple process but it is not directly callable. The Process class defines properties to describe IOPEs, data bindings, local parameters (for Atomic processes only) and participants (for Composite processes only). Inputs, Outputs and Preconditions are specified through the hasInput, hasOutput and hasPreCondition properties respectively. An effect is specified in the Result of a process which specifies conditions (inCondition), effects (hasEffect), result variables (hasResultVar) and outputs (withOutput). The in-condition specifies a condition such that it must be true in order for the effect of the particular process to occur. An effect describes the state of the world after the particular process is performed. The ResultVar class declares variables that are bound in the inCondition. The withOutput property defines a binding mechanism between processes. The convention adopted by OWL-S for data binding is that of a consumer-pull mechanism. For example, if Process A feeds Process B, then the binding is declared in Process B. Details of how the binding is performed is outside the scope of this document and we refer the reader to [Martin et al., 2004]. Composite Processes define their workflows using Control Constructs which are Sequence, Split, Split-Join, Any-Order, Choice, If-Then-Else, Iterate, Repeat-While and Repeat-Until. It is worth noting how OWL-S defines constructs to describe logical expressions. Another OWL ontology has been defined which specifies the class Expression. Its subclasses are SWRL-Expression, DRS-Expression and KIF-Expression representing the specifications of SWRL [Patel-Schneider, 2004], DRS [McDermott, 2004] and KIF respectively. Furthermore, another subclass is Condition whose other subclasses are SWRL-Condition, DRS-Condition and KIF-Condition. These expressions and conditions are used to specify different aspects in the OWL-S Service Model, namely, preconditions, effects and in-conditions.

The Service Grounding describes how the service is accessed by specifying a mapping from the abstract to the concrete levels of a Web Service. Both the Service Profile and the Service Model entities in OWL-S are regarded as abstract which differ from the Service Grounding being a concrete entity. The Service Grounding deals with message formats, serialization, transport and addressing. So far, OWL-S presents WSDL [Christensen et al., 2001] as its basis for the grounding mechanism. In simple terms, the Atomic Processes, Inputs and Outputs are mapped to Operations, Input Messages and Output Messages respectively in a WSDL. However, when it comes to atomic processes, there is no restriction for a one-to-one mapping to a WSDL operation but rather a one-to-many since there may be defined multiple groundings for a service.

3. Mapping from OWL to WSML

Since the underlying ontology languages for OWL and WSML defer, we first need a mapping between them in order to reuse more complex expressions and the domain ontologies referenced. The mapping is partially provided in [De Brujin et al., 2004] (Part III - Mapping to OWL). For the purpose of this document we will only give an outline of how the both languages correspond, the concrete and complete definitions are out of the scope of this document and will be provided in future versions of [De Brujin et al., 2004]. Both languages provide subsets with different expressivity. In the case of OWL, they are OWL Lite, OWL DL and OWL Full. However the different translations are based on the computational tractability within the WSML Family of ontologies and in conclusion do not directly correspond to the OWL layering.

OWL* [De Brujin et al., 2004b] defines a subset of OWL Lite that is also known as DLP fragment. This subset can be mapped to WSML Core, which semantically corresponds to the intersection of Description Logics and Horn Logic.

OWL DL If an ontology uses features that can not be mapped to WSML Core, the ontology will be mapped to WSML DL. Currently WSML DL is not specified, but it will be a syntactic variant of OWL DL using the WSML syntax. The mapping will be similar to the known mapping between Description Logics and First Order Logic. Also an alternative syntax allowing more concise descriptions of DL axioms is currently under discussion.
OWL Full inherits the semantics from RDFS and is a full first order logic language. It can be partly mapped to WSML Full, however since WSML Full is currently not specified such a mapping does not yet exist.

The mapping from the different WSML dialects to OWL will be considered in future versions of this deliverable.

4. Mapping of Logical Expressions

4.1 Notations for Expressing Rules in OWL-S and WSML

In WSML, currently only the species WSML-Core and WSML-Flight are specified. Both these dialects allow to specify logical axioms and rules to some extent. These logical expressions are allowed among others in axioms in Ontology definitions as well as in Preconditions, Assumptions, Postconditions and Effects of Web Service Capabilities and Goals. As long as OWL-S service descriptions purely rely on OWL DL axioms to express conditions and the like we suffice with and adhere to the same restrictions for the mappings from and to OWL as in [De Brujin et al., 2004]. OWL-S itself has in its last version been enriched by the possibility to express logical conditions not only by pure OWL statements but also accept more expressive language extensions, such as DRS [McDermott, 2004], KIF and SWRL [Patel-Schneider, 2004]. We will, for its well-defined semantics and smooth syntactical integration with OWL only take SWRL into account here. SWRL is a submission to the W3C for an extension of OWL by rules of the form

\[ \text{Implies} \left( \text{Antecedent}\left(\left\{\text{Atom}\right\}\right), \text{Consequent}\left(\left\{\text{Atom}\right\}\right) \right) \]

where the Antecedent and the Consequent correspond to a list of atoms. These atoms are of the form C(i-object) or D(d-object) where C is an OWL description or a D datatype, respectively, or R(i-object, i-object) or Q(i-object, d-object) where R and Q are ObjectProperties or Datatype properties, respectively. Furthermore, SWRL allows several unary built-in predicates for datatype predicates. Here, i-objects and d-objects refer to individualIDs or data literals, respectively, or to (object or datatype) variables. For allowing existential OWL restrictions in the heads of rules and building up on OWL DL, SWRL cannot be translated to neither WSML Core or WSML Flight. Since however the semantics of WSML Full is not defined yet and there is no separate as WSML layer foreseen for capturing FOL at the moment, the best we can do is to translate to WSML Full. However, the syntax and semantics of WSML Full is not yet specified. Therefore we restrict ourselves to an informal mapping which we will refine in future versions of this document. Furthermore we will include some discussion on under which syntactic restrictions on an OWL-S specification the result of the mapping falls into WSML Flight, or (likely, although also yet underspecified) into WSML Rule.

4.2 SWRL FOL to WSML

Note that the current Version we cover only SWRL. Will be extended to SWRL FOL in the next version. Original SWRL has been suggested as a simple extension of OWL DL by rules. In principle, each SWRL rule is translated by translating the contained OWL descriptions as outlined in [De Brujin et al., 2004], property atoms of the form R(X,Y) are translated to Molecules of the form X[R hasValue Y]. We will give a detailed overview of which built-ins are translatable in furtle versions of this document upon future versions of [De Brujin et al., 2004]. SameAs(X,Y) and DifferentFrom(X,Y) do not have a correspondence in the currently specified WSML variants. SWRL variables are translated one to one to variables in WSML. A complete rule

\[ \text{Implies} \left( \text{Antecedent}(E_1), \text{Consequent}(E_2) \right) \]

in SWRL is then translated to a complex logical expression of the form

\[ \text{tr}(E_1) \text{ implies } \text{tr}(E_2) \]

in WSML where tr(\{\}) is the translation of the list of atoms.

This translation will be more detailed in subsequent versions.

4.3 WSML to SWRL FOL

Rules of the form in WSML

This translation will be more detailed in subsequent versions.

5. Conceptual Model Mapping

In this section, we will provide a mapping (where possible) between the WSMO and OWL-S conceptual models. We will start by mapping the OWL-S components to the equivalent WSMO elements. For each component, we will further analyse its properties and attempt to specify these properties in WSMO.

5.1 OWL-S to WSMO

To provide a mapping from OWL-S to WSMO, we will consider each OWL-S component separately and expand on each. First we will get over the general Upper Service Ontology which links to the other components. We will then continue considering the Service Profile, Service Model and Service Grounding elements of OWL-S. We foresee that in certain cases a mapping will not be possible. In those cases, the reasons that make the mapping impossible will be explained. We will use as an example the BravoAir service provided with the OWL-S 1.1 Release and define the equivalent elements in WSMO.

5.1.1 Upper Service Mapping

At the conceptual level, the Service concept of OWL-S corresponds to the Web Service element of WSMO. It can be argued that an instance of Service in OWL-S can also be interpreted as a service request and mapped to a goal. Although the OWL-S specification is not detailed on this regard, our interpretation is that a service request is intended to be described by only an OWL-S service profile, without any need for the definition of a service instance. Table 1 shows the translation of an OWL-S service instance into a WSMO Web Service.

<table>
<thead>
<tr>
<th>OWL-S</th>
<th>WSMO</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BravoAir</td>
<td>Service</td>
<td></td>
</tr>
</tbody>
</table>
namespaces | namespaces | From these, the default namespace is translated into the default namespaces of the web service description
--- | --- | ---
Instance of service | web service |  
Identifier of instance of service | Identifier of web service | Value of `dc:identifier` non functional property of web service
Value of `owl:versionInfo` | Value of `version` non functional property of web service |  
--- | --- | ---
Value of `owl:imports` | Generation of an `ooMediators` to import each ontology, and include such mediators as `usedMediators` in the web service | Some of the ontologies imported in the OWL-S description will not be necessary for the WSMO description, but at that point we cannot distinguish them so we will generate a mediator and use it for every of them.
[Remark HL:] Why did your mapping change, I found it more intuitive to not import them since we are translating all of them with this document into one webService description in WSML anyway.
Value of presents property | Web service capability identifier |  
Value of `describedBy` property | Web service choreography identifier | It can be argued that the OWL-S service model could also be interpreted as the WSMO orchestration. However, the common interpretation of the service model in the work related to OWL-S (e.g. [Ankolekar et al., 2004]) is that it defines the external, observable behaviour of the Web service.
Value of supports property | Choreography grounding identifier | Translation not possible, as the WSMO grounding is not yet defined.

Table 1: Mapping of OWL-S service to WSMO Web service.

In addition, we can generate additional WSMO non functional properties, such as `dc:type` with value `web service`, `dc:format` having as format the one chosen for the generation of the WSMO description, and the `dc:relation`, having as value the original OWL-S description. The `dc` namespace is also automatically added to the resulting WSMO translation. Listing 1 shows the definition of an example OWL-S service given by the OWL-S coalition, and Listing 2 shows the corresponding translation into WSMO. Notice that in Listing 2, an arbitrarily chosen target namespace is used. A strategy for generating such namespaces and to translate identifiers is required, and will be discussed in future versions of this document.

[Remark HL:] When you map the OWL-ID to `dc:identifier` in WSML anyway, then you also should use that targetNamespace...
Listing 1. The BravoAir service in OWL-S.

```xml
<?xml version='1.0' encoding='ISO-8859-1'?><
<!DOCTYPE uridef [  
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">  
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">  
<!ENTITY owl "http://www.w3.org/2002/07/owl">  
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema">  
<!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">  
<!ENTITY ba_profile "http://www.daml.org/services/owl-s/1.1/BravoAirProfile.owl">  
<!ENTITY ba_process "http://www.daml.org/services/owl-s/1.1/BravoAirProcess.owl">  
<!ENTITY ba_grounding "http://www.daml.org/services/owl-s/1.1/BravoAirGrounding.owl">  
<!ENTITY DEFAULT "http://www.daml.org/services/owl-s/1.1/BravoAirService.owl"> ]>

<rdf:RDF xmlns:rdf="&rdf;#" xmlns:rdfs="&rdfs;#" xmlns:owl="&owl;#" xmlns:xsd="&xsd;#" xmlns:service="&service;#" xmlns:ba_profile="&ba_profile;#" xmlns:ba_process="&ba_process;#" xmlns:ba_grounding="&ba_grounding;#" xmlns="&DEFAULT;#" xml:base="&DEFAULT;">
  <owl:Ontology rdf:about="">
    <owl:versionInfo>
      $$Id: d42.html,v 1.20 2004/12/17 21:22:33 axel Exp $$
    </owl:versionInfo>
    <rdfs:comment>
      This ontology represents the OWL-S service description for the
      BravoAir web service example.
    </rdfs:comment>
    <owl:imports rdf:resource="&service;" />
    <owl:imports rdf:resource="&ba_profile;" />
    <owl:imports rdf:resource="&ba_process;" />
    <owl:imports rdf:resource="&ba_grounding;" />
  </owl:Ontology>

  <service:Service rdf:ID="BravoAir_ReservationAgent">
    <!-- Reference to the BravoAir Profile -->
    <service:presents rdf:resource="&ba_profile;#Profile_BravoAir_ReservationAgent"/>
    <!-- Reference to the BravoAir Process Model -->
    <service:describedBy rdf:resource="&ba_process;#BravoAir_Process"/>
    <!-- Reference to the BravoAir Grounding -->
    <service:supports rdf:resource="&ba_grounding;#Grounding_BravoAir_ReservationAgent"/>
  </service:Service>
</rdf:RDF>
```
Listing 2. The BravoAir service in WSMO.

```xml
namespace <<http://www.daml.org/services/owl-s/1.1/BravoAirService#>>
rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>
rdfs: <<http://www.w3.org/2000/01/rdf-schema#>>
owl: <<http://www.w3.org/2002/07/owl#>>
xsd: <<http://www.w3.org/2001/XMLSchema#>>
service: <<http://www.wsmo.org/owl-smapping/v0.1/ServiceMediator.wsml>>
ba_profile: <<http://www.wsmo.org/owl-smapping/v0.1/BravoAirProfileMediator.wsml>>
ba_process: <<http://www.wsmo.org/owl-smapping/v0.1/BravoAirProcessMediator.wsml>>
ba_grounding: <<http://www.wsmo.org/owl-smapping/v0.1/BravoAirGroundingMediator.wsml>>
dc: <<http://purl.org/dc/elements/1.1#>>
targetnamespace: <<http://www.wsmo.org/owl-smapping/v0.1/ws#>>

webService BravoAir_ReservationAgent

nonFunctionalProperties
  dc:identifier hasValue <<http://www.wsmo.org/owl-smapping/v0.1/BravoAirService#BravoAir_ReservationAgent>>
  dc:type hasValue <<http://www.wsmo.org/2004/d2/#webService>>
  dc:format hasValue "text/html"
  dcrelation hasValues {<http://www.daml.org/services/owl-s/1.1/BravoAirService.owl>}
version hasValue "$Id: d42.html,v 1.20 2004/12/17 21:22:33 axel Exp $"
endNonFunctionalProperties

usedMediators {
  <<http://www.daml.org/services/owl-s/1.1/ServiceMediator.wsml>>,
  <<http://www.wsmo.org/owl-smapping/v0.1/BravoAirProfileMediator.wsml>>,
  <<http://www.wsmo.org/owl-smapping/v0.1/BravoAirProcessMediator.wsml>>,
  <<http://www.wsmo.org/owl-smapping/v0.1/BravoAirGroundingMediator.wsml>>
}
capability Profile_BravoAir_ReservationAgent
  [...]
  interface GeneratedInterface
  choreography BravoAir_Process
```

Notice that at the moment we do not include the generation of the ooMediators. This will be included in future versions of the document.

5.1.2 Service Profile Mapping

OWL-S does not distinguish between the profiles offered by a service and the profiles requested by a user. Therefore, its interpretation can be ambiguous. However, as explained in the previous section, we assume that a requested profile will be defined as a stand-alone profile and, therefore, will not be linked to any instance of service. Under this assumption, we will interpret an OWL-S profile as a WSMO capability if it has some value for its presentedBy property, and as a WSMO goal otherwise. Please notice that the latter situation is possible, as the OWL-S ontology does not define any cardinality restriction for the presents and presentedBy properties and, therefore, a profile not referring to any service is a valid OWL-S description. Table 2 shows the translation of an OWL-S profile into a WSMO capability.
<table>
<thead>
<tr>
<th>Value of phone property from actor</th>
<th>Value of foaf:phone property of dc:creator, dc:publisher and dc:contributor</th>
<th>This is only in the case where the range of contactInformation is actor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of fax property from actor</td>
<td>No mapping available</td>
<td></td>
</tr>
<tr>
<td>Value of e-mail property from actor</td>
<td>Value of foaf:mbox property of dc:creator, dc:publisher and dc:contributor</td>
<td>This is only in the case where the range of contactInformation is actor.</td>
</tr>
<tr>
<td>Value of physicalAddress property from actor</td>
<td>No mapping available</td>
<td></td>
</tr>
<tr>
<td>Value of webURL property from actor</td>
<td>Value of foaf:homepage property of dc:creator, dc:publisher and dc:contributor</td>
<td>This is only in the case where the range of contactInformation is actor.</td>
</tr>
<tr>
<td>Value of serviceCategory property</td>
<td>Value of dc:subject non functional property of web service</td>
<td>The range of the serviceCategory is not restricted, so what kind of values we will deal with is not clear.</td>
</tr>
<tr>
<td>Value of serviceParameter property</td>
<td>No mapping available</td>
<td></td>
</tr>
<tr>
<td>Value of hasInput property</td>
<td>Value of preconditions property of web service</td>
<td>The actual definition of the precondition axioms will be given in the OWL-S process or at least using its scheme for the definition of inputs.</td>
</tr>
<tr>
<td>Value of hasOutput property</td>
<td>Value of postconditions property of web service</td>
<td>The actual definition of the postcondition axioms will be given in the OWL-S process or at least using its scheme for the definition of outputs.</td>
</tr>
<tr>
<td>Value of hasPrecondition property</td>
<td>Value of assumptions property of web service</td>
<td>The actual definition of the assumption axioms will be given in the OWL-S process or at least using its scheme for the definition of conditions.</td>
</tr>
<tr>
<td>Value of hasResult property</td>
<td>Value of effects property of web service</td>
<td>The actual definition of the effect axioms will be given in the OWL-S process or at least using its scheme for the definition of results.</td>
</tr>
<tr>
<td>Value of serviceClassification property</td>
<td>Value of dc:subject property of web service</td>
<td>Notice that serviceCategory is also mapped to dc:subject.</td>
</tr>
<tr>
<td>Value of serviceProduct property</td>
<td>Value of dc:coverage property of web service</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Mapping of OWL-S profile to WSMO capability.</th>
</tr>
</thead>
</table>

The Service Profile allows to reference for IOPEs which are specified in the Process Model ontology. Since at this stage we are only dealing with references, it is more appropriate to map these elements from the Process Model.

Listing 3 shows the definition of an example OWL-S profile given by the OWL-S coalition, and Listing 4 shows the corresponding translation into WSMO.

Listing 3. The BravoAir profile in OWL-S.

```xml
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE undeclared>
<!ENTITY xsql "http://www.w3.org/2001/XMLSchema">
<!ENTITY rdfs "http://www.w3.org/1999/02/22-rdf-syntax-ns">
<!ENTITY rdfrdfs "http://www.w3.org/2000/01/rdf-schema">
<!ENTITY owl "http://www.w3.org/2002/07/owl">
<!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
<!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
<!ENTITY actor "http://www.daml.org/services/owl-s/1.1/ActorDefault.owl">
<!ENTITY addParam "http://www.daml.org/services/owl-s/1.1/ProfileAdditionalParameters.owl"/>
<!ENTITY profileHierarchy "http://www.daml.org/services/owl-s/1.1/ProfileHierarchy.owl"/>
<!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
<!ENTITY ba_service "http://www.daml.org/services/owl-s/1.1/BravoAirService.owl">
<!ENTITY ba_service "http://www.daml.org/services/owl-s/1.1/BravoAirProcess.owl">
<!ENTITY country "http://www.daml.org/services/owl-s/1.1/Country.owl"/>
<!ENTITY concepts "http://www.daml.org/services/owl-s/1.1/Concepts.owl"/>
<!ENTITY DEFAULT "http://www.daml.org/services/owl-s/1.1/BravoAirProfile.owl"/>
>
<rdf:RDF
xmlns:rdf="&rdfs;#"
xmlns:owl="&owl;#"
xmlns:service="&service;#"
xmlns:process="&process;#"
xmlns:profile="&profile;#"
xmlns:actor="&actor;#"
xmlns:addParam="&addParam;#"
xmlns:profileHierarchy="&profileHierarchy;#"
>
<owl:Ontology rdf:about=""/>

<rdfs:comment>DAML-S Coalition: BravoAir Example for OWL-S Profile description</rdfs:comment>

</owl:Ontology>

<profileHierarchy:AirlineTicketing rdf:ID="Profile_BravoAir_ReservationAgent">
    <service:presentedBy rdf:resource="&ba_service;#BravoAir_ReservationAgent"/>
    <profile:has_process rdf:resource="&ba_process;#BravoAir_Process"/>
    <profile:serviceName>BravoAir_ReservationAgent</profile:serviceName>
    <profile:textDescription>
        This service provide flight reservations based on the specification of a flight request. This typically involves a departure airport, an arrival airport, a departure date, and if a return trip is required, a return date. If the desired flight is available, an itinerary and reservation number will be returned.
    </profile:textDescription>

    </profile:contactInformation>

    <profile:contactInformation>
        <actor:Actor rdf:ID="BravoAir-information">
            <actor:name>John Doe</actor:name>
            <actor:title>Sale Representative</actor:title>
            <actor:phone>412 268 8789</actor:phone>
            <actor:fax>412 268 5569</actor:fax>
            <actor:email>John_Doe@Bravoair.com</actor:email>
            <actor:physicalAddress>
                Airstrip 2,
                Teetering Cliff Heights,
                Florida 12321,
                USA
            </actor:physicalAddress>
            <actor:webURL>http://www.daml.org/services/daml-s/2001/05/BravoAir.html</actor:webURL>
        </actor:Actor>
    </profile:contactInformation>
</profileHierarchy:AirlineTicketing>
Listing 4. The BravoAir profile in WSMO.

```xml
namespace <<http://www.daml.org/services/owl-s/1.1/BravoAirService#>>
rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>
owl: <<http://www.w3.org/2002/07/owl#>>
xsd: <<http://www.w3.org/2001/XMLSchema#>>
service: <<http://www.wsomo/owl-smapping/v0.1/ServiceMediator.wsml>>
ba_profile: <<http://www.wsomo/owl-smapping/v0.1/BravoAirProfileMediator.wsml>>
ba_process: <<http://www.wsomo/owl-smapping/v0.1/BravoAirProfileMediator.wsml>>
ba_grounding: <<http://www.wsomo/owl-smapping/v0.1/BravoAirGroundingMediator.wsml>>
dc: <<http://purl.org/dc/elements/1.1#>>
profile: <<http://www.wsomo/owl-smapping/v0.1/ProfileMediator.wsml>>
actor: <<http://www.wsomo/owl-smapping/v0.1/ActorDefaultMediator.wsml>>
addParam: <<http://www.wsomo/owl-smapping/v0.1/ProfileAdditionalParametersMediator.wsml>>
process: <<http://www.wsomo/owl-smapping/v0.1/ProcessMediator.wsml>>
concepts: <<http://www.wsomo/owl-smapping/v0.1/ConceptsMediator.wsml>>
country: <<http://www.wsomo/owl-smapping/v0.1/CountryMediator.wsml>>
profileHierarchy: <<http://www.wsomo/owl-smapping/v0.1/ProfileHierarchyMediator.wsml>>
targetnamespace: <<http://www.wsomo/owl-smapping/v0.1/ws#>>

webservice BravoAir_ReservationAgent

nonFunctionalProperties
dc:identifier hasValue <<http://www.wsomo/owl-smapping/v0.1/BravoAirService#BravoAirReservationAgent>>
dc:type hasValue <<http://www.wsomo/2004/d2/#webservice>>
dc:format hasValue "text/html"
dc:relation hasValues <<http://www.daml.org/services/owl-s/1.1/BravoAirService.owl>>,
<<http://www.daml.org/services/owl-s/1.1/BravoAirProfile.owl>>
dc:title hasValue "BravoAirReservationAgent"
dc:description hasValue
"This service provide flight reservations based on the specification of a flight request. This typically involves a departure airport, an arrival airport, a departure date, and if a return trip is required, a return date. If the desired flight is available, an itinerary and reservation number will be returned."
dc:subject hasValues (profileHierarchy:AirlineTicketing)
dc:creator hasValues (BravoAir-reservation, BravoAir-information)
dc:publisher hasValues (BravoAir-reservation, BravoAir-information)
dc:contributor hasValues (BravoAir-reservation, BravoAir-information)
version hasValue "$Id: d42.html,v 1.20 2004/12/17 21:22:33 axel Exp $$"
endNonFunctionalProperties

usedMediators <<http://www.wsomo/owl-smapping/v0.1/ServiceMediator.wsml>>,
<<http://www.wsomo/owl-smapping/v0.1/BravoAirProfileMediator.wsml>>,
<<http://www.wsomo/owl-smapping/v0.1/BravoAirProcessMediator.wsml>>,
<<http://www.wsomo/owl-smapping/v0.1/ProfileMediator.wsml>>,
<<http://www.wsomo/owl-smapping/v0.1/ActorDefaultMediator.wsml>>,
<<http://www.wsomo/owl-smapping/v0.1/ProfileAdditionalParametersMediator.wsml>>,
<<http://www.wsomo/owl-smapping/v0.1/ProcessMediator.wsml>>,
<<http://www.wsomo/owl-smapping/v0.1/ConceptsMediator.wsml>>,
<<http://www.wsomo/owl-smapping/v0.1/CountryMediator.wsml>>,
<<http://www.wsomo/owl-smapping/v0.1/ProfileHierarchyMediator.wsml>>

capability Profile_BravoAir_ReservationAgent

nonFunctionalProperties

endNonFunctionalProperties

precondition axiom DepartureAirport
precondition axiom ArrivalAirport
precondition axiom DepartureAirport
precondition axiom OutboundDate
precondition axiom InboundDate
precondition axiom RoundTrip
precondition axiom AcctName
precondition axiom Password
precondition axiom Confirm
postcondition axiom FlightsFound
```
Some important remarks about the translation are in place and must be discussed:

- The *Actor* instances used in the OWL-S description would correspond to FOAF agents in WSMO. However, the contact information can correspond to a non-existing foaf:agent that, therefore, has to be created during the translation, or to an existing foaf:agent that must be located based on the information contained in the actor instance and only referenced from the WSMO description.

  **[Remark HL:]** I'd suggest to create an ontology in the same document containing that information.

- OWL-S service parameters such as *GeographicRadius* are not known a priori, and the semantics of these i.e. whether they correspond to *dc:coverage* or other WSMO non functional properties is not clear. Therefore, a straightforward mapping cannot be done. An option would be to have a list of commonly used service parameters and a mapping to the corresponding WSMO non functional properties.

- Instances of e.g. NAICS categorizations that are used as values for the OWL-S *serviceCategory* property are created inside the OWL-S description of the profile. For the translation, they should be placed in external domain ontologies and only referenced from the WSMO description. Similar to FOAF agents, this might imply locating existing categorization instances and referencing them.

  **[Remark HL:]** I suggest to do the same and define just an "inline" ontology, since we are just trying to do a mapping and not a fixing...

- As pointed out for the OWL-S service instance, some of the imported ontologies are not required in the WSMO description e.g. http://www.daml.org/services/owl-s/1.1/BravoAirProcess.owl. A possible strategy to filter out ontologies imported but not required is to check whether these are actually used in the complete WSMO translation.

  **[Remark HL:]** Filter all except external domain ontologies (since the profile, process and grounding also gets translated...)

- A consistent strategy for the generation of identifiers for the WSMO translation has to be defined

  **[Remark HL:]** What holds against doing it in the same targetNamespace? we are defining the same thing, just another representation, thus the target NS is appropriate.

### 5.1.3 Process Model Mapping

The OWL-S Service Model describes how the service works. The main idea is to model the service in terms of processes and control constructs defined in composite services. In this ontology, the Inputs, Outputs, Preconditions and Results (together with preconditions) are also specified. In essence, the Service Model corresponds to the WSMO Web Service Capability and Choreography. Notice that OWL-S doesn't distinguish between Choreography and Orchestration and hence a mapping to the orchestration cannot be obtained. Table 3 below describes the properties as specified in the Service Model and an equivalent mapping to WSMO.

<table>
<thead>
<tr>
<th>OWL-S</th>
<th>WSMO</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaces</td>
<td>namespaces</td>
<td>Same as for Service and Profile mappings</td>
</tr>
<tr>
<td>Value of owl:imports</td>
<td>Generation of a wwMediator to import each ontology and include them in the usedMediators in the Web Service</td>
<td>In this case, it is more ideal to generate wwMediators rather than ooMediators since we are connecting two ontologies that describe the actual functionality of the service. As for the profile, the generated mediators may not be needed so we will import only the ontologies which have not been imported during the translation of the service instance. <strong>[Remark HL:]</strong> Objection. I see them also as OOmediators</td>
</tr>
<tr>
<td>Value of owl:versionInfo</td>
<td>Value of version in the non-functional properties of the service choreography</td>
<td>In the profile ontology, we would have already mapped the value of this property to the version attribute in the service capability. In this case it would be more appropriate to define it in the choreography.</td>
</tr>
<tr>
<td>Subclass of ServiceModel</td>
<td>Service capability and choreography.</td>
<td></td>
</tr>
<tr>
<td>Subclass of Process</td>
<td>Service capability and choreography.</td>
<td>Since a process is a subclass of ServiceModel, it may describe itself as a service capability and choreography. However, this is only the case for a composite process since only such a process...</td>
</tr>
</tbody>
</table>
can express a choreography of components in OWL-S.

Since multiple composite processes can be specified, it is rather difficult to generate the respective elements in WSMO in this case. A possible solution would be that of adding the preconditions and effects in the capability through conjunctions and define multiple interfaces for the service in order to allow the definition of multiple choreographies.

Although the choreography in WSMO is still underspecified, we can consider an atomic process to be equivalent to a state in a particular choreography.

A simple process is an abstraction of either a composite or atomic process. It cannot be invoked and it is not yet clear whether the choreography in WSMO will specify such a similar element.

A concept in a choreography is a sub concept of another concept in WSML but it defines also the non-functional attribute mode. This attribute may have the values in, out, shared, controlled or static. Further information can be found at [Roman et al., 2004].

The range of value of these properties in OWL-S is the class Expression which may be specified using KIF, DRS or SWRL FOL. In this document we provide a mapping only SWRL FOL and hence details of such a mapping can be found in Section 4 of this document.

An inCondition specifies the condition under which a result occurs. An assumption describe the state of the world before the execution of the service and hence these properties can be mapped.

As regards to map the logical expressions in OWL-S for WSMO, we refer the reader to Section 4 of this document where a mapping between SWRL FOL and WSML is provided. Note that in this document we don't consider a mapping between WSML and KIF or DRS since only SWRL provides well defined semantics.

In future versions of this document (and as WSMO choreography will be better specified), we will provide the mappings (if any) of the other elements defined in an OWL-S process model.

5.1.4 Grounding Mapping

The Grounding in WSMO has not yet been specified. However, we will outline how this is done in OWL-S and give a brief description of the current on-going work about this aspect in WSMO. The Grounding in OWL-S is based on the Web Service Description Language (WSDL). In simple terms, an Atomic Process is mapped to an Operation and Inputs/Outputs are mapped to Messages. This technique is summarized in Figure 3.
Currently there are two approaches for WSMO grounding. The first approach is to specify an XSLT transformation from the WSML data to the XML schema provided by the given service. This is not such a good solution since during the process, the semantic information is lost. Another approach is that of specifying a set of mappings between the WSML description and the ontology provided by the service. The ontology provided by the service would then be lowered to the corresponding XML Schema. These two efforts are both in a primitive stage and further information can be obtained in future versions of [Moran et al., 2004]. Once it is clear of how the grounding will be specified in WSMO, in future versions of this document we will then provide a mapping between the groundings of OWL-S and WSMO.

5.2 WSMO to OWL-S

[To be done in future versions]

5.2.1 Goals Mapping

5.2.2 Web Services Mapping

5.2.3 Mediators Mapping

5.2.4 Ontologies Mapping

[JS]: Wouldn’t actually the mapping of ontologies be already mentioned once discussed in Section 3 of this document? Or should we list the mappings as in D16.1 Section 10?

6. APIs

Application Programming Interfaces (APIs) provide an abstraction layer from the concrete syntaxes of a language, allowing the developer to work on a conceptual level rather than being concerned about serialization or parsing details. Thus APIs are for the realization of such a mapping tool an essential building block. In the following we give a brief overview of the available tools.

6.1 WSMO4J

WSMO4J is developed by OntoText Lab. and DERI Innsbruck and provides a data model in Java for WSML and (de-)serializers for the different WSML syntaxes. It will additionally provide wrappers for different Reasoners, the first one according to the current plans will be KAON2. Others are not yet scheduled by the core development team, however also third parties can extend this open source API. Currently the the 2nd release candidate of WSMO4J is available, it is based on WSMO v1.0.

6.2 OWL API

The OWL API is developed by the University of Manchester. It provides essentially a in memory representation of an OWL ontology and (de-)serializers for RDF/XML and the OWL abstract syntax. In opposite to the Jena API it is not based on triples, but was especially designed for representing Description Logics (i.e. OWL Lite and OWL DL ontologies). We anticipate this as advantage for translations, since it provides a conceptually clearer access to the underlying language.

6.3 OWL-S API

The OWL-S API is being developed by Mindswap. The current release of the API supports OWL-S versions 1.0, 0.9 and 0.7 (the last two being formerly known as DAML-S). OWL-S 1.1 has been released very recently and hence we may expect that future versions of the API will support such a specification.

An execution engine is provided but with limited support. The grounding specifications supported so far are WSDL and UPnP and only atomic processes which are grounded to such specifications can be executed. Composite processes can be executed unless they use control constructs which are dependent on conditionals. Such constructs include IfThenElse, Repeat-While and Repeat-Until. The constructs which are supported currently are Sequence, Unordered and Split. Furthermore, preconditions and effects are not supported. These limitations arise from the fact that OWL-S 1.0 does not specify how logical expressions are to be defined in its ontologies.

Another limitation regards the manipulation of OWL within the service. There exists a class named OWLResource which is the parent of all other classes which implement the elements in OWL-S. The underlying data model is based on Jena, however, it is thought that...
in future versions of the API, this might change to the OWL API.

Since in this document we are considering a mapping to OWL-S 1.1, it is of no use to provide information about the structure of the current release of the API. This will be done once the OWL-S 1.1 is supported.

7. Mapping Tool

[Future Versions]

7.1 Overview of Architecture

8. Conclusions and Further Directions

References


[De Brujin et al., 2004] J. De Brujin (ed.): The WSML Family of Representation Languages, version 0.2 available at http://www.wsno.org/2004/d16/d16.1/v0.2/


[Roman et al., 2004] D. Roman (ed.): Choreography in WSMO, version 0.1 available at http://www.wsno.org/2004/d14/v0.1/


Acknowledgement

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; by the Vienna city government under the CoOperate programme and by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW² and TSC.

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

webmaster