An Approach for Semantic Query Processing with UDDI

Jim Luo, Bruce Montrose and Myong Kang

Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC 20375
{luo, montrose, mkang}@itd.nrl.navy.mil

Abstract. UDDI is not suitable for handling semantic markups for Web services due to its flat data model and limited search capabilities. In this paper, we introduce an approach to allow for support of semantic service descriptions and queries using registries that conforms to UDDI V3 specification. Specifically, we discuss how to store complex semantic markups in the UDDI data model and use that information to perform semantic query processing. Our approach does not require any modification to the existing UDDI registries. The add-on modules reside only on clients who wish to take advantage of semantic capabilities. This approach is completely backward compatible and can integrate seamlessly into existing infrastructure.

1. Introduction

Automatic discovery of Web services is an important capability for the Service-Oriented Architecture (SOA). The first step in providing this capability is to mark up Web services with metadata in a well-understood and consistent manner. The W3C community developed the Web ontology language (OWL) to address this problem [1]. It is a machine understandable description language that is capable of describing resources in a manner much richer than the traditional flat taxonomies and classification systems. OWL-S is a set of ontology developed specifically to describe web services [2]. After the semantic service descriptions are created, the next step is to advertise them in a registry capable of fine-grained semantic matchmaking. Universal Description, Discovery and Integration (UDDI) is a Web-based distributed registry for the SOA [3]. It is one of the central elements of the interoperable framework and an OASIS standard with major backers including IBM and Microsoft. However, UDDI is limited to using flat syntax-based identification and classification system. It is not capable of storing and processing semantic service descriptions written in OWL.

It is clear that semantic annotation and matchmaking for Web services will produce much more refined search results than UDDI-style syntactic matching [4, 5]. It is also clear that UDDI is fast becoming widely accepted as a Web infrastructure standard already with widespread deployment by companies, government agencies, and the military. The goal of this ongoing work is to add OWL based semantic markups and query capabilities to existing registry implementations that conforms to

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2005		**00-00-2005 to 00-00-2005**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
An Approach for Semantic Query Processing with UDDI	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Center for High Assurance Computer Systems,4555 Overlook Avenue, SW,Washington,DC,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**10**	

the UDDI V3 specification. Service descriptions can be expressed using the OWL-S ontology, however, our approach provides support for the OWL language as a whole and any ontology can be used. This approach does not require any modification to the existing UDDI infrastructure. Users that wish to take advantage of semantic annotation and query capabilities can simply install modules in their own client machines and use UDDI registries as semantic registries. This will not be the ideal solution in the long term. Registries specifically developed for semantic service description and query processing will be much more effective and efficient. However, this approach will provide a short-term solution that will allow organizations to start using OWL service descriptions without having to make significant additional investments in SOA infrastructure.

2. Semantic Annotation and Queries

This section describes the kinds of semantic annotations and queries we plan to support. We will use the following three example ontologies in figure 1 and the service description concept in figure 2 throughout the rest of the paper.

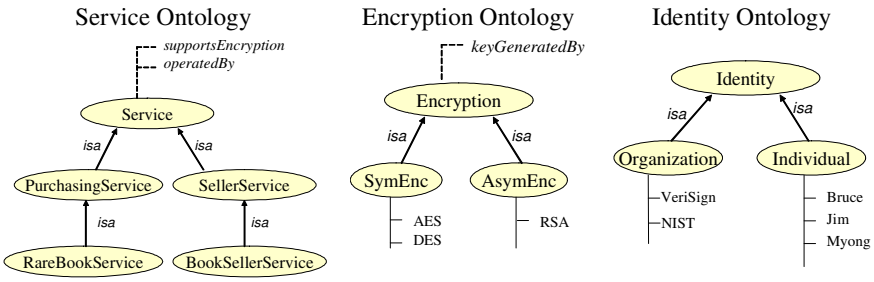

Fig. 1. Ontology examples. Ovals represent classes, solid lines represent instances, dotted lines represent properties, and arrows represent subclass relationships

Semantic annotations describe Web services using concepts from ontologies. For example, a Web service may advertise itself as a BookSellerService from the Service Ontology. The service description can further annotate the ontology concept by defining additional properties on BookSellerService such as in figure 2.

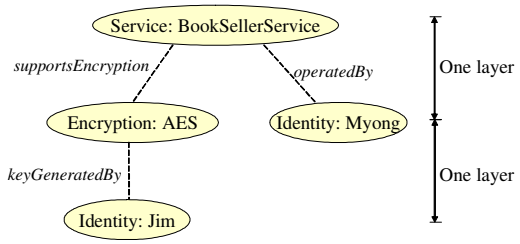

Fig. 2. An example of Web service semantic annotation

Semantic annotations can have multiple layers of annotations. However, the UDDI data model is only capable of storing one layer of annotations because it was designed to deal with flat identification and categorization systems. Thus the first challenge is to correctly express complex semantic service descriptions in the UDDI data model without losing any details.

The second challenge is to provide support for semantic query. Two types of queries must be supported by the system.

Exact Match Queries: the first type of queries uses the same concepts as those specified in the service description. Query processing does not require ontology awareness.

- Find a BookSeller service
- Find a BookSeller service operated by Myong
- Find a BookSeller service that supports AES encryption
- Find a BookSeller service that supports AES encryption with the key generated by Jim

Semantic Match Queries: The second type of queries uses semantically related concepts as those specified in the service description. Query processing requires ontology awareness.

- Find a BookSeller service that supports SymEnc
- Find a BookSeller service that supports Encryption
- Find a BookSeller that supports SymEnc with the key generated by Jim

The classes and instances specified in these queries are not the same as those in the service description. However, they should all match to the concept in figure 2 because they are related due to the class hierarchy and inferences established by the ontologies. This semantic information must be captured and processed by the UDDI registry in order to support semantic matchmaking.

3. Mapping Strategy

UDDI is intended to serve as a repository for Web services. The UDDI specification defines a set of core data models for storing Web service descriptions and an API for interacting with the registry. The core data model consists of objects describing the Web service (*businessService*), the service provider (*businessEntity*), and the service binding (*bindingTemplate*). In addition to the static text-based data fields, these data objects can incorporate metadata into the description by making references to *tModels*. TModels, or technical models, provide extensibility to the overall UDDI framework because they can be used to represent *any* concept or construct. The versatility of the tModel comes from the fact that it only serves as a place holder. Information is not actually imported into the registry. UDDI core data model objects reference tModels with *keyedReferences* and *keyedReferenceGroups*. KeyedReferenceGroups can be used to group logically related concepts.

The tModel framework is very powerful and provides the UDDI system with a great deal of extensibility and flexibility. They can be used for a variety of different purposes including for the storage of ontology information.

In this mapping scheme, each individual concept within the ontology including all instances, classes and properties will be incorporated into the UDDI registry as tModel objects that can be referenced individually. This use of tModels is much more complex than what is envisioned by the UDDI specification. Each distinct ontology concept will be represented by a separate tModel. Additional tModels will be created to represent anonymous composite concepts defined in service descriptions such as the one in figure 2. Figure 3 shows the UDDI representation of that concept using tModels and keyedReferences.

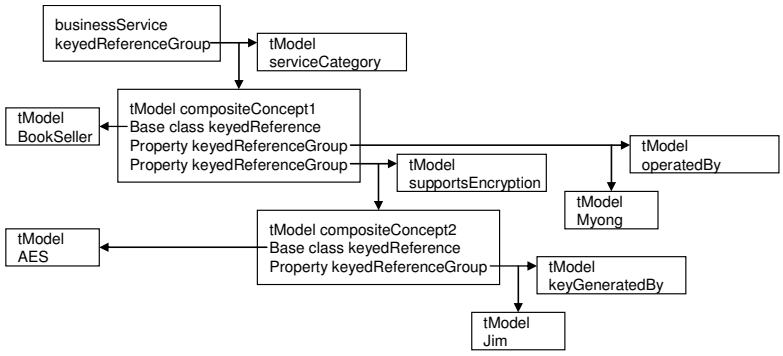

Fig. 3. UDDI tModel representation of the complex service description in figure 2

The overall composite concept is represented by the tModel "compositeConcept1." In addition, the component composite concept, AES with key generated by Jim, is represented by its own tModel "compositeConcept2." Composite concept tModels will hold a reference to designate its base class. Annotations are captured as keyedReferenceGroups what will reference the property type and property value tModels as children keyedReference elements. This way, each tModel can hold one layer of annotations and chaining multiple tModels together will allow for representation of composite concepts with multiple layers of annotation. The entire composite concept can be referred to by referencing the top layer concept tModel.

4. Semantic Support

The UDDI search engine is only capable of performing syntax matching. However, OWL requires a semantic matchmaker capable of taking into account relationships between concepts established by the ontology. Our approach fully resolves and indexes ontology relationships at publishing time of the ontology. This way, queries can be processed syntactically by the UDDI search engine and yield results equivalent to those of a semantic matchmaker.

The first type of ontology relationships that must be resolved involves the property and class hierarchy defined using the *subClassOf*, *equivalentClass*, *subPropertyOf*, and *equivalentProperty* constructs. *Identity concepts* are defined as the set of related concepts for which queries should yield the original concept based on the class and property hierarchy established in the ontology. If query for Encryption should return class RSA, then Encryption would be an identity class of RSA. An ontology reasoner can resolve the list of identity concepts for all the base concepts at the publishing time of the ontology. When service descriptions are published, any references made to ontology concepts need to also include the identity concepts. For example, if a service is capable of RSA, it also needs to indicate that it is capable of Encryption. This way, queries for both RSA and Encryption will yield the service.

The second type of relationships that must be resolved involves property characteristics. Object properties can be defined with the characteristics of *TransitiveProperty* and *SymmetricProperty*. *Inferred properties* are properties not explicitly defined in the ontology or service description but could be inferred based on property characteristics and other property definitions. Inferred properties will be resolved for all the base ontology concepts by the reasoner at the publishing time of the ontology. For symmetric properties, the reverse of the explicit property definition must also be defined as an inferred property. For transitive properties, the entire chain of transitivity must be resolved and referenced as inferred properties. When service descriptions are published, it must make appropriate references to inferred properties in addition to the explicit property definitions.

True ontology awareness is only necessary during publication of the ontology. Publication and query of service description can be done syntactically using identity concepts and inferred properties captured during ontology publication. The UDDI search engine, however, is not capable of all the syntax matching operations necessary due to its lack of supporting Boolean queries. Therefore, our system will use an additional matchmaker component on the client side. The part of the query processed by the UDDI search engine will only deal with base ontology class of composite concepts and return a coarse list of possible matching services. The matchmaker on the client side will refine the list by matching composite concepts in their entirety. For example, if the query is for the composite concept presented in figure 2, the query passed on to UDDI will simply be for BookSeller. The matchmaker on the client side will match the service descriptions returned by UDDI against the full composite concept.

Our prototype implementation will not fully support all aspects of the OWL language [1]. This is governed by the functional limitations of UDDI as well as the desire to keep the prototype relatively simple. The system will not enforce validation of ontologies and service descriptions. It will be up to the user to validate their own OWL documents. Complex class expressions involving *intersectionOf* and *complementOf* will not be supported. This is the lack of Boolean query support in the current version of UDDI specification. Class expressions involving only *unionOf* will be supported because the identity concept approach treats sets of classes and properties as unions by default. More advanced inferences based on cardinality, complex class expressions and other property characteristics and will not be supported. These types of inferences are generally intended for reasoning about the

ontology and are not directly relevant to matchmaking. It is important to note that the actual mapping is lossless and all information will be captured inside UDDI data structures. The limitations are on the query side in that some constructs will not be taken into consideration during query processing.

5. System Architecture and Query Processing

Details of our prototype implementation can be found in [6]. Figure 4 shows the overall system architecture. The shaded boxes are the four add-on modules that will reside in client sides. Only clients wishing to use UDDI registries as semantic registries will need to add these modules to their machines.

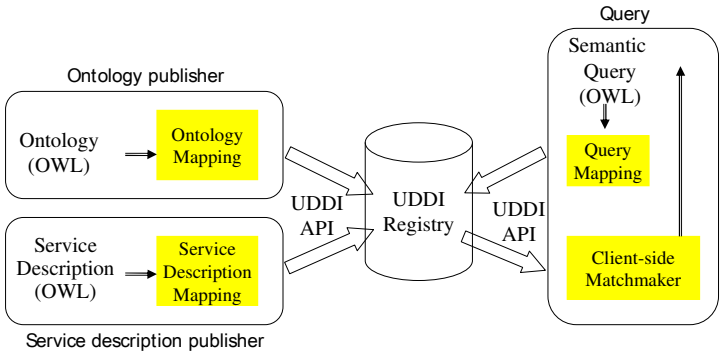

Fig. 4. A system architecture for semantic processing with UDDI

Ontology mapping and service description mapping modules are implemented in XSLT and Java. XSLT translates OWL documents into the UDDI data model and Java code publishes them into the registry using the UDDI client-side API. Identity concepts and inferred properties are resolved in the ontology mapping module which includes a simple ontology reasoner implemented in XSLT. The service description mapping module will syntactically propagate semantic information captured in the ontology tModels to the service descriptions.

The query mapping module is also implemented in XSLT and Java. The XSLT component will strip annotations from composite concepts and translate the OWL queries into the UDDI queries. The Java component will then query the registry through the UDDI client-side API. The results returned by the registry will be the businessServices objects with references that match only the base ontology concepts.

The client-side matchmaker is a Java module that will refine the query results by performing matchmaking that takes into account all the annotations of composite concepts. It will query the registry for concept tModels referenced by the businessService object to fully reconstruct the service description. Then it will match the service description with the query by examining the concepts at each layer of annotation. This component is only necessary because the UDDI specification lacks support for Boolean queries. Its task can be folded into the registry if future versions of UDDI provide that support.

6. Mapping Specification for Publication

This section summarizes the mapping specification from OWL ontology and service descriptions to the UDDI data model.

6. 1. Ontology Mapping

Ontology tModel: the ontology tModel that will serve as a place holder and namespace for the ontology as a whole. It will hold overview information including the ontology name, description, and URL of external descriptions. This tModel will be referenced by all instance, class and property tModels associated with the ontology using the KeyValue of "ontologyReference."

Property Type tModel: the property type tModel will store ObjectProperty information defined in the ontology. The name of the tModel will be set to the name of the class defined in the ontology. Information not used for query processing such as domain, range, and property characteristics will also be captured.

```
<owl:ObjectProperty rdf:ID="operatedBy">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="&identity;Identity"/>
  <rdfs:type rdf:resource="&owl:TransitiveProperty">
</owl:ObjectProperty>
```

```
<tModel tModelKey="uuid:operatedBy">
  <name>operatedBy</name>
  <categoryBag>
    <KeyedReference keyValue="ontologyReference" tModelKey="uuid:service"/>
    <KeyedReference keyValue="true" tModelKey="uuid:IsOntologyCore"/>
    <keyedReference keyValue="domain"  tModelKey="uddi:service:Service" />
    <keyedReference keyValue="range"  tModelKey="uddi:idnetity:Identity" />
    <keyedReference keyValue="TransitiveProperty"  tModelKey="uddi:owl:type" />
  </categoryBag>
</tModel>
```

Fig. 5. tModel that maps an object property

Class and Instance tModel: the name of the tModel will be set to the name of the concept defined in the ontology. It will hold keyedReferences to the tModels representing its identity concepts. The list of identity relationships will be derived by the ontology reasoner based on the class hierarchy. The types of the identity concepts are stored in the keyValue field of keyedReferences. All classes will have an identity relationship to itself with the relationship type of "exactRelationship." The other possible relationship types are "equivalentRelationship" for equivalent concepts, "generalizationRelationship" for parent concepts, and "specializationRelationship." for children concepts. Classes defined as union of other classes will hold identity concepts to those other classes as well as their identity concepts.

```
<owl:Class rdf:ID="SellerService">
  <rdfs:subClassOf rdf:resource="#Service"/>
</owl:Class>
```

```
<tModel tModelKey="uuid:SellerService ">
  <name>SellerService</name>
  <categoryBag>
    <KeyedReference keyValue="ontologyReference" tModelKey="uuid:Service"/>
    <KeyedReference keyValue="exactRelationship" tModelKey="uuid:SellerService"/>
    <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Service"/>
    <KeyedReference keyValue="specializationRelationship" tModelKey="uuid:BookSeller"/>
  </categoryBag>
</tModel>
```

Identity concepts

Fig 6. tModel that maps the SellerService class from figure 1

Composite Concept tModel: composite concepts are classes and instances with property definitions. They can be defined in the ontology as restriction classes or in service descriptions as anonymous instances. For restriction classes, the tModel name will be set to the name of the class defined in the ontology. Anonymous instances are not named and the name of the tModel will be left blank. Composite concepts can have multiple layers of annotations. If the annotations are composite concepts themselves, new tModels need to be created for them as well. Property definitions are captured as keyedReferenceGroups. The property type and property value tModels as well as the tModels of their identity concepts are captured as keyedReferences under the keyedReferenceGroup. Both explicit and inferred properties are captured the same way and no distinction is made between them.

```
<owl:Class rdf:ID="SecureSellerService">
  <rdfs:subClassOf rdf:resource="#SellerService"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="supportsEncryption"/>
      <owl:hasValue>encryption:AES</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```
<tModel tModelKey="uuid:SecureSellerService ">
  <name>SecureSellerService</name>
  <categoryBag>
    <KeyedReference keyValue="ontologyReference" tModelKey="uuid:service"/>
    <KeyedReference keyValue="true" tModelKey="uuid:IsOntologyCore"/>
    <KeyedReference keyValue="exactRelationship" tModelKey="uuid:SellerService"/>
    <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Service"/>
    <KeyedReference keyValue="specializationRelationship" tModelKey="uuid:BookSeller"/>
    <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
      <KeyedReference keyValue="exactProperty" tModelKey= uuid:supportsEncryption">
      <KeyedReference keyValue="exactRelationship" tModelKey="uuid:AES"/>
      <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:SymEnc"/>
      <KeyedReference keyValue="specializationRelationship" tModelKey="uuid:Encryption"/>
    </KeyedReferenceGroup>
  </categoryBag>
</tModel>
```

base class

Identity concepts

property type

property value

Identity concepts

Fig. 7. tModel that maps a restriction class

6.2. Service Description Mapping

Translation of service description involves two steps. First, tModels for any anonymous composite concept defined in the service description must be published into the registry unless they already exist. Second, the service description must be translated into a corresponding UDDI businessEntity and businessService objects.

Service Description: Information that maps to the UDDI businessEntity and businessService data model objects can be translated directly. Ontology references in the service description are stored as keyedReferenceGroup the same way as properties in the tModels for composite concepts. In addition, the base class of composite property value concepts is also referenced.

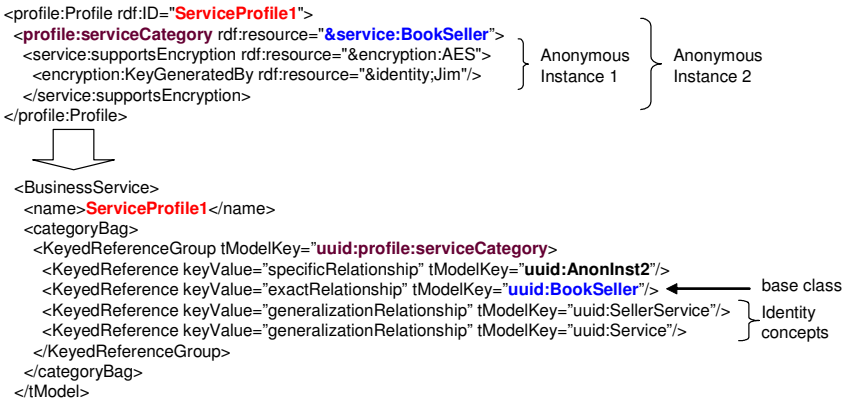

Fig. 8. BusinessService referencing the concept in figure 2

7. Related Work

Srinivasan [7] added semantic support to UDDI by placing add-on modules on the registry side. This means the existing registry infrastructure needs to be modified extensively to provide semantic support. Furthermore, the add-on modules create special interfaces for processing semantic publications and queries separate from the UDDI interface. In effect, these modules act as separate semantic registries that happen to be on the same server as opposed to integrated with the UDDI registry.

Sivashanmugam [4] developed a scheme to store ontology-based semantic markups using the native UDDI data model. However, their solutions do not support composite concepts with multiple layers of annotations. Ontology concepts can be referenced as is, but they cannot be further annotated by service descriptions. Furthermore, class hierarchy between concepts in the ontology is not captured by the translation. It is not clear if semantic queries can be supported using this approach.

The Web Service Modeling Ontology (WSMO) based Web Service Modeling Language (WSML) is an alternative to OWL-S and OWL for semantically describing

web services [8]. Since a direct mapping exists between WSML and OWL [9], WSMO will be supported indirectly in our system.

8. Conclusion

We presented an approach for supporting semantic markups of Web services and semantic queries using existing registries conforming to the UDDI V3 specification. Support is provided for the OWL language as a whole and the system will operated with any OWL ontology including OWL-S. A special lossless translation scheme that fully supports composite concepts was developed to store ontologies and semantic service descriptions inside UDDI registries. Once all the semantic information is captured, semantic query processing can be performed using a combination of the UDDI search engine and syntax based client-side matchmaker.

This approach does not require any modification to the existing registry or infrastructure. The advantage is that it is completely backward compatible. The add-on modules only need to be installed on the clients of users who wish to take advantage of semantic markups. They can be integrated seamlessly into existing systems and operations without any modification of the infrastructure.

References

1. W3C, "OWL Web Ontology Language Overview." 2004 <http://www.w3.org/TR/owl-features/ >.
2. Web Ontology Working Group, "OWL-S: Semantic Markup for Web Services," W3C. <http://www.daml.org/services/owl-s/1.1/overview/>.
3. UDDI Spec Technical Committee, "UDDI Version 3.0.2," OASIS. 2004 <http://uddi.org/pubs/uddi_v3.htm>.
4. K. V. K. Sivashanmugam, A. Sheth, and J. Miller, "Adding Semantics to Web Services Standards," presented at International Conference on Web Services, 2003.
5. A. Dogac, G. Laleci, Y. Kabak, and I. Cingil, "Expoliting Web Service Semantics: Taxonomies vs. Ontologies," IEEE Data Engineering Bulletin, vol. 25, 2002.
6. J. Luo, B. Montrose, and M. Kang, "Adding Semantic Support to Existing UDDI Infrastructure," Naval Research Lab, Washington, D.C., NRL Memorandum Report NRL/MR/5540-05-650, 2005.
7. M. P. N. Srinivasan, and K. Sycara, "Adding OWL-S to UDDI, implementation and throughput," presented at First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, California, USA, 2004.
8. W3C, "Web Service Modeling Ontology (WSMO)." 2005 <http://www.w3.org/Submission/WSMO/>.
9. W3C, "Web Service Modeling Language (WSML)." 2005 <http://www.w3.org/Submission/WSML/>.