Web Service Semantics - WSDL-S

Technical Note

Version 1.0

April, 2005

Authors (alphabetically):

Rama Akkiraju, IBM Research

Joel Farrell, IBM Software Group

John Miller, LSDIS Lab, University of Georgia

Meenakshi Nagarajan, LSDIS Lab, University of Georgia

Marc-Thomas Schmidt, IBM Software Group

Amit Sheth, LSDIS Lab, University of Georgia

Kunal Verma, LSDIS Lab, University of Georgia

Copyright Notice

Abstract

The current WSDL standard operates at the syntactic level and lacks the semantic expressivity needed to represent the requirements and capabilities of Web Services. Semantics can improve software reuse and discovery, significantly facilitate composition of Web services and enable integrating legacy applications as part of business process integration. The Web Service Semantics technical note defines a mechanism to associate semantic annotations with Web services that are described using Web Service Description Language (WSDL). It is conceptually based on, but a significant refinement in details of, the original WSDL-S proposal [WSDL-S] from the LSDIS laboratory at the University of Georgia. In this proposal, we assume that formal semantic models relevant to the services already exist. In our approach, these models are maintained outside of WSDL documents and are referenced from the WSDL document via WSDL extensibility elements. The type of semantic information that would be useful in describing a Web Service encompass the concepts defined by the semantic Web community in OWL-S [OWL-S] and other efforts [METEOR-S, WSMO]. The semantic information specified in this document includes definitions of the precondition, input, output and effects of Web service operations. This approach offers multiple advantages over OWL-S. First, users can describe, in an upwardly compatible way, both the semantics and operation level details in WSDL- a language that the developer community is familiar with. Secondly, by externalizing the semantic domain models, we take an agnostic approach to ontology representation languages. This allows Web service developers to annotate their Web services with their choice of ontology language (such as UML or OWL) unlike in OWL-S. This is significant because the ability to reuse existing domain models expressed in modeling languages like UML can greatly alleviate the need to separately model semantics. Finally, it is relatively easy to update the existing tooling around WSDL specification to accommodate our incremental approach.

Status

This is a technical note provided for discussion purposes and to elicit feedback on approaches to adding semantics to Web services descriptions.

Table of Contents

1. Introduction

As the set of available Web Services expands, it becomes increasingly important to have automated tools to help identify services that match a requester's requirements. Finding suitable Web services depends on the facilities available for service providers to describe the capabilities of their services and for service requesters to describe their requirements in an unambiguous and ideally, machine-interpretable form. Adding semantics to represent the requirements and capabilities of Web services is essential for achieving this unambiguity and machine-interpretability. Benefits of using semantics can make them pervasive in the complete lifecycle of Web services. During development, the service provider can explicate the intended semantics by annotating the appropriate parts of the Web service with concepts from a richer semantic model. Since semantic models provide agreement on the meaning and intended use of terms, and may provide formal and informal definitions of the entities, there will be less ambiguity in the intended semantics of the provider. During discovery, the service requestor can describe the service requirements using terms from the semantic model. Reasoning techniques can be used to find the semantic similarity between the service description and the request. During composition, the functional aspect of the annotations can be used to aggregate the functionality of multiple services to create useful service compositions. More importantly, semantics can make it possible to specify mappings between data exchanged through XML-based SOAP messages, which would be extremely difficult to do with syntactic representation offered by the current standards. During invocation, mappings can be used for data transformations. Therefore, once represented, semantics can be leveraged by tools to automate service discovery, mediation, composition and monitoring. Current WS-* standards operate at the syntactic level and lack semantic representation capabilities. This poses an impediment to developing tools to assist humans and/or support semi-automatic process and application integration. In this technical note, we address this problem by applying the work of the semantic Web community to the Web services standards.

The World Wide Web Consortium (W3C) Web services architecture [W3CWSA] defines two aspects of the full description of a Web service. The first is the syntactic functional description as represented by WSDL. The second is described as the semantics of the service and is not covered by a specification. In practice, the semantic description is either missing or informally documented. By examining the WSDL description of a service, we cannot unambiguously determine what the service does. We can see the syntax of its inputs and outputs, but we do not know what these mean or what changes to the environment the service makes. We do not know the meaning of the parameters nor the terms referenced in payload documents. Indeed, two services can have the same syntactic definition but perform significantly different functions. Similarly, two syntactically dissimilar services can perform the same function.

Semantic markup of Web Services has been proposed as an approach to address the above issues. Example proposals include initiatives, projects and languages such as WSMO [WSMO], METEOR-S [METEOR-S], OWL-S [OWL-S] and SWSA/SWSL [SWSA, SWSL]. While the semantic expressivity is rich and flexible in OWL-S, arguably the most visible research

approach to date, it defines a new way to describe Web services and suffers from some important limitations. First, is not aligned with the existing Web services standards. For example, while the grounding model in OWL-S uses WSDL bindings, the OWL-S profile model duplicates the descriptions embodied in the rest of WSDL. Second, it assumes that everyone uses OWL for representing ontologies which may not always be the case. To overcome these limitations, we propose a new approach in this document. The same observations apply to the rest of the proposals identified above.

The approach described in this technical note is an evolutionary and compatible upgrade of the existing Web services standards, and more specifically Web service descriptions. It is a revision of the WSDL-S proposal [WSDL-S] from the METEOR-S group at the University of Georgia. In this approach, we augment the expressivity of WSDL with semantics by employing concepts analogous to those in OWL-S while being agnostic to the semantic representation language. In this document, we only refer to the OWL-S profile model (component of OWL-S that describes functionality of Web services) , the OWL-S process model (component of OWL-S that describes the interaction protocol of a Web services) compares with BPEL4WS and is not discussed here. The advantage of this evolutionary approach to adding semantics to WSDL is multi-fold. First, users can, in an upwardly compatible way, describe both the semantics and operation level details in WSDL- a language that the developer community is familiar with. Second, by externalizing the semantic domain models, we take an agnostic approach to ontology representation languages. This allows Web service developers to annotate their Web services with their choice of ontology language (such as UML or OWL). This is significant since the ability to reuse existing domain models expressed in modeling languages like UML can greatly alleviate the need to separately model semantics. Moreover, this approach realizes the need for the existence of multiples ontologies, either from the same or different domains to annotate a single Web service and provides a mechanism to do so. Finally, it is relatively easy to update the existing tooling around WSDL specification to accommodate our incremental approach. While it is noted that the theoretical underpinnings of OWL-S in description logic makes it a richer language for representing semantics, we believe that extending the industry standards such as WSDL to include semantics is a more practical approach for adoption. Moreover, by externalizing the semantic domain models in our proposal, we still allow for richer representations of domain concepts and relationships in languages such as OWL and UML, thereby bringing together the best of both worlds.

1.1 Terminology

We provide basic definitions for the terminology we use in this technical note.

Semantics - Semantics in this context refers to the meaning of objects or information.

Semantic Model - A semantic model captures the terms and concepts used to describe and represent an area of knowledge or some part of the world, including a software system. A semantic model usually includes concepts in the domain of interest, relationships among them, their properties, and their values. Usually this is described as an ontology that embodies agreement

Semantic Annotation - A semantic annotation is additional information in a document that defines the semantics of a part of that document. In this technical note, the semantic annotations are additional information elements in a WSDL document. They define semantics by referring to a part of a semantic model that describes the semantics of the part of the document being annotated.

Input Semantics - Input semantics is the meaning of input parameters as defined by some semantic model.

Output Semantics - Output semantics is the meaning of output parameters as defined by some semantic model.

Precondition - A precondition is a set of semantic statements (or expressions represented using the concepts in a semantic model) that are required to be true before an operation can be successfully invoked.

Effect - An effect is a set of semantic statements (or expressions represented using the concepts in a semantic model) that must be true after an operation completes execution after being invoked. Different effects can be true depending on whether the operation completed successfully or unsuccessfully.

1.2 Information Model

The WSDL [WSDL] document forms the anchor point for Web services description. Building on the descriptive capability of WSDL, we provide a mechanism to annotate the capabilities and requirements of Web services with semantic concepts referenced from a semantic model. To do this, we provide mechanisms to annotate the service and its inputs, outputs and operations. Additionally, we provide mechanisms to specify and annotate preconditions and effects of Web Services. These preconditions and effects together with the semantic annotations of inputs and outputs can enable automation of the process of service discovery.

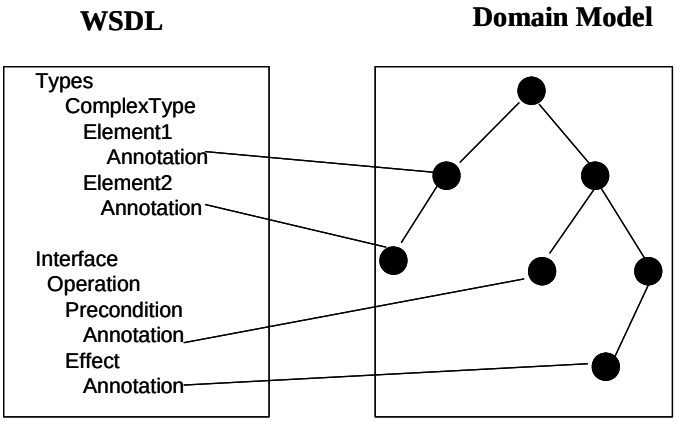


Figure 1. Externalized representation and association of semantics to WSDL elements

Figure 1 shows how semantic annotations are associated with various elements of a WSDL document (including inputs, outputs and functional aspects like operations, preconditions

and effects) by referencing the semantic concepts in an external domain semantic model. The domain model can consist of one or more ontologies.

1.3 Notational Conventions

The keywords [keywords] "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [RFC2119].

1.4 Namespaces

The XML namespace names [XMLNamespace] URIs [URIs] defined by this technical note is as follows:

Prefix	Namespace name
wssem	http://www.ibm.com/xmlns/WebServices/WSSemantics

2. Requirements for Web Service Semantics

We recommend that certain principles guide any work to define a framework for Web services semantics. Our work is guided by the following principles.

- *Build on existing Web Services standards*: The Web services standards are fast becoming a preferred technology for application integration because of the promise of their interoperability. Companies are making investments in integration projects based on Web Services. Therefore, we believe that any approach to adding semantics to Web Services should be specified in an upwardly compatible manner so as to not disrupt the existing install-base of Web Services.

- *The mechanism for annotating Web services with semantics should be independent of the semantic representation language*: There are a number of potential languages for representing semantics such as OWL [OWL], WSMO [WSMO], and UML [UML]. Each language offers different levels of semantic expressivity and developer support. Our position is that it is not necessary to tie the Web services standards to a particular semantic representation language. This is consistent with the approach prescribed by Sivashanmugam et al in their work [SVS03]. By keeping the semantic annotation mechanism separate from the representation of the semantic descriptions, the approach offers flexibility to developer community to select their favorite semantic representation language. In the next section, we will show a way such independence can be achieved.

- *The mechanism for annotating Web services with semantics should allow the association of multiple annotations written in different semantic representation languages*: As mentioned earlier, there are many potential semantic representation languages. Service providers may choose to annotate their services in multiple semantic representation languages to be discovered by multiple discovery engines. Therefore, we believe that the

mechanism for annotating Web Services with semantics should allow multiple annotations to be associated with Web Services.

- *Support semantic annotation of Web Services whose data types are described in XML schema*: A common practice in Web services-based integration is to reuse interfaces that are described in XML. The definition of business documents using XML schema is a wide-spread and successful practice. XML schemas will be an important data definition format for the foreseeable future. We believe that the semantic annotation of service inputs and outputs should support the annotation of XML schemas. WSDL 2.0 supports the use of other type systems in addition to XML Schema, so constructs in semantic models, such as classes in OWL [OWL] ontologies, could be used to define the Web service input and output data types. But an approach that does not address XML schema-based types will not be able exploit exiting assets or allow the gradual upgrade of deployed WSDL documents to include semantics.

- *Provide support for rich mapping mechanisms between Web Service schema types and ontologies*: Given our position on the importance of annotating XML schemas in Web service descriptions, attention should be given to the problem of how to map XML schema complex types to ontological concepts. Again, an agnostic approach to the selection of schema mapping languages is called for. For example, if the domain model is represented in OWL, the mapping between WSDL XSD elements and OWL concepts can be represented in any language of user's choice such as: RDF, OWL, XSLT, XQuery or any other arbitrary language as long as the chosen language is fully qualified with its own namespace.

3. An Example

We first present an example WSDL document that is annotated with semantic information to give the reader a preview of what is explained in the rest of the document. The semantic annotations are explained in section 4 with specific examples drawn from this example.

In this sample, we present a simple purchase order service. The inputs and outputs of ProcessPurchaseOrder service are annotated with semantics, two new elements namely preconditions and effects are introduced as extensibility elements to the operation construct in WSDL, and an extensibility element called category is added to the interface construct. The semantic concepts and their relationships are modeled in an OWL ontology – PurchaseOrder.owl (presented in Appendix C).

PurchaseOrder.wsdl is given below.

```
<definitions name="PurchaseOrder"
   targetNamespace="http://www.ourdemos.com/purchaseorder/wsdl/PurchaseOrder/"
   xmlns="http://www.w3.org/2004/08/wsdl"
   xmlns:tns="http://www.ourdemos.com/purchaseorder/wsdl/PurchaseOrder/"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1="http://www.ourdemos.com/purchaseorder/"
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
```

```
  xmlns:POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl">

  <types>
   <xs:import namespace=" http://www.ibm.com/xmlns/WebServices/WSSemantics"
            schemaLocation= "WSSemantics.xsd />
   <xs:import namespace=" http://www.ourdemos.com/purchaseorder/"
            schemaLocation= "POBilling.xsd />
   <xs:import namespace=" http://www.ourdemos.com/purchaseorder/"
            schemaLocation= "POItem.xsd />
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=" http://www.ourdemos.com/purchaseorder/wsdl/PurchaseOrder/"
    xmlns="http://www.ourdemos.com/purchaseorder/wsdl/PurchaseOrder/">
      <!—Semantic annotations for these complex types are given in their respective type
          definitions>
      <xs:complexType name="processPurchaseOrderRequest">
        <xs:all>
          <xs:element  name="billingInfo" type="xsd1:POBilling"/>
          <xs:element  name="orderItem" type="xsd1:POItem"/>
        </xs:all>
      </xs:complexType>
      <!—Semantic annotation is added directly to non-leaf element />
      <xs:element name= "processPurchaseOrderResponse" type="xs:string
        wssem:modelReference="POOntology#OrderConfirmation"/>
      </xs:schema>
  </types>
  <interface name="PurchaseOrder">
     <!—Category is added as an extensible element of an interface>
    <wssem:category name= “Electronics” taxonomyURI=”http://www.naics.com/”
       taxonomyCode=”443112” />
    <operation name="processPurchaseOrder” pattern=wsdl:in-out>
      <input messageLabel = ”processPurchaseOrderRequest"
       element="tns:processPurchaseOrderRequest"/>
      <output messageLabel ="processPurchaseOrderResponse"
       element="processPurchaseOrderResponse"/>
      <!—Precondition and effect are added as extensible elements on an operation>
      <wssem:precondition name="ExistingAcctPrecond"
        wssem:modelReference="POOntology#AccountExists">
      <wssem:effect name="ItemReservedEffect"
        wssem:modelReference="POOntology#ItemReserved"/>
    </operation>
  </interface>
</definitions>
```

In this WSDL document, the input *processPurchaseOrderRequest* includes complex child elements. The definition of semantic annotations for these complex types is done at the level of leaf elements in complex types in this example. Leaf level mappings are discussed later in detail. In this technical note, an alternate way to annotate complex types is also proposed. This approach associates schema mapping functions at the level of complex types. Schema mapping functions represented in XSLT and XQuery are discussed in detail in Appendices A and B respectively. Native support for OWL types (as supported in WSDL 2.0) is also possible within our framework. Details for OWL type support are available in an earlier version of this work [WSDL-S]. The XSD definitions of all the extensions defined in this document are shown in the next section.

WSSemantics.xsd is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.ibm.com/xmlns/stdwip/Web-services/WS-Semantics"
      xmlns:wssem="http://www.ibm.com/xmlns/stdwip/Web-services/WSSemantics"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

     <attribute name="modelReference" type="anyURI" use="optional"/>

     <attribute name="schemaMapping" type="anyURI" use="optional"/>

     <element name="category" maxOccurs="unbounded">
      <complexType>
       <complexContent>
          <extension base="wsdl:documented">
              <attribute name="categoryname" type="NCName" use="required""/>
             <attribute name="taxonomyURI" type="anyURI" use="required""/>
             <attribute name="taxonomyValue" type="String" use="optional"/>
             <attribute name="taxonomyCode"  type="integer" use="optional"/>
         </extension>
       </complexContent>
      </complexType>
      </element>

      <element name = "precondition">
         <complexType>
          <complexContent
              <restriction base="anyType">
                   <xsd:attribute name="name" type="string" />
                   <attribute name="modelReference" type="anyURI" />
                   <attribute name="expression" type="string" />
              </restriction>
           </complexContent>
          </complexType>
      </element>

      <element name="effect">
        <complexType>
          <complexContent
             <restriction base="anyType">
                 <xsd:attribute name="name" type="string" />
                <attribute name="modelReference" type="anyURI" />
                <attribute name="expression" type="string" />
             </restriction>
          </complexContent>
        </complexType>
       </element>
</schema>
```

This schema is referenced in the definition of the *xmlns:wssem* namespace definition. The following three XML schema documents define the input and output documents for the service.

POItem.xsd. which defines the properties of an item in a purchase order, is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified"
   elementFormDefault="unqualified"
   targetNamespace="http://www.ourdemos.com/purchaseorder/"
```

```
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1="http://www.ourdemos.com/purchaseorder/"
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
   xmlns:POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl">

<import location="WSSemantics.xsd"
    namespace=" http://www.ibm.com/xmlns/WebServices/WSSemantics"/>

   <complexType name="POItem" >
      <all>
         <element name="dueDate" nillable="true" type="dateTime"
            wssem:modelReference="POOntology#DueDate"/>
         <element name="qty" type="float" wssem:modelReference="#POOntology#Quantity"/>
         <element name="EANCode" nillable="true" type="string"
            wssem:modelReference="POOntology#ItemCode"/>
         <element name="itemDesc" nillable="true" type="string"
            wssem:modelReference="POOntology#ItemDesc" />
      </all>
   </complexType>
</schema>
```

POBilling.xsd, which defines the billing information in a purchase order, is given below. In POBilling.xsd, the elements of POBilling complex type namely shipToAddress and billToAddress are of type POAddress which is a complexType in itself. Therefore, the specification of semantic annotations for these contained complex types are deferred to the corresponding xsds – in this case POAddress.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified"
   elementFormDefault="unqualified"
   targetNamespace="http://www.ourdemos.com/purchaseorder/"
   xmlns=http://www.w3.org/2001/XMLSchema
   xmlns:xsd1=http://www.ourdemos.com/purchaseorder/
   xmlns:POOntology=">
   <include schemaLocation="POAddress.xsd"/>
  <include schemaLocation="Account.xsd"/>
   <complexType name="POBilling" Billing>
      <all>
         <element name="shipToAddress" nillable="true" type="xsd1:POAddress"/>
         <element name="billToAddress" nillable="true" type="xsd1:POAddress"/>
         <element name="accountID" nillable="true" type="xsd1:string"
           wssem:modelReference="POOntology#AccountID/>
      </all>
   </complexType>
</schema>
```

POAddress.xsd, which defines an address in a purchase order, is given below.

```
 <?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified"
   elementFormDefault="unqualified"
   targetNamespace="http://www.ourdemos.com/purchaseorder/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1="http://www.ourdemos.com/purchaseorder/"
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
```

```
  xmlns:POOntology="http://www.ibm.com/ontologies/PurchaseOrder1.owl">

   <import location="WSSemantics.xsd"
     namespace=" http://www.ibm.com/xmlns/WebServices/WSSemantics/"/>

  <complexType name="POAddress">
     <all>
        <element name="recipientInstName"  type="string"
          wssem:modelReference=”POOntology#Receiver”/>
         <element name="streetAddr1" type="string"
          wssem:modelReference=”POOntology#StreetAddress”/>
        <element name="streetAdd2" type="string"
          wssem:modelReference=”POOntology#StreetAddress”/>
        <element name="city" type="string" wssem:modelReference=”POOntology#City“/>
        <element name="zipCode" type="string"
          wssem:modelReference=”POOntology#PostalCode”/>
        <element name="state" type="string" wssem:modelReference=”POOntology#State”/>
        <element name="country" type="string"
          wssem:modelReference=”POOntology#Country”/>
     </all>
  </complexType>
</schema>
```

4. Using the Extensibility Elements of WSDL

In this section we describe how semantic annotations are added to WSDL document elements.

Conceptually, WSDL 2.0 has the following constructs to represent service descriptions: interface, operation, message, binding, service and endpoint [WSDL 2 Diff]. Of these, the first three, namely interface, operation and message constructs deal with the abstract definition of a service while the remaining three given by binding, service and endpoint constructs deal with service implementation. In this technical note, we focus on semantically annotating the abstract definition of a service to enable dynamic discovery, composition and invocation of services (it is important to note that semantic annotations would be of use in service implementations as well. For example, if a message exchange protocol A is compatible with another protocol B, such information could be represented in domain models and made use of during invocation. However, we do not address the annotation of service implementation at this time. Service level annotations are in part addressed by WS-Policy). We provide URI reference mechanisms via extensibility elements to the interface, operation and message constructs to point to the semantic annotations defined in the domain models for services.

A quick summary of the extensibility elements provided in this technical note are given below:

- an extension element, namely **modelReference**, to handle one-to-one mapping of schema elements to the concepts in a semantic model
- an extension attribute, namely **schemaMapping**, which is added to XSD complextypes and elements, for associating the schema elements of a Web service

with semantic models, such as ontologies to handle many-one and one-many mappings.

- two new elements, namely **precondition** and **effect**, which are specified as child elements of the *operation* element. Preconditions and effects are primarily used in service discovery, and are not necessarily required to invoke a given service (in this technical note, we defer the detailed representation of preconditions and effects, which could include a combination of complex expressions, to the underlying semantic domain representation models or ontologies) and
- an extension attribute on interface element, namely **category**. It consists of service categorization information that could be used when publishing a service in a Web Services registry such as UDDI. Semantic categorization of UDDI registries using ontologies was proposed in [SVS04, MWSDI]

Annotating input and output elements

In this section, we describe how to annotate the input and output elements of a WSDL document. In the purchase order example, the *processPurchaseOrder* operation had one input and one output. The input is represented by the element *processPurchaseOrderRequest* which is given by xsd:complexType *processPurchaseOrderRequest*. The output is represented by the element *processPurchaseOrderResponse.* A WSDL operation is shown below.

```
  <interface name="PurchaseOrder">
    <operation name="processPurchaseOrder" pattern=wsdl:in-out>
      <input messageLabel = "processPurchaseOrderRequest"
element="tns:processPurchaseOrderRequest"/>
      <output messageLabel ="processPurchaseOrderResponse"
element="processPurchaseOrderResponse"/>
      <!—Precondition and effect are added as extensible elements on an operation>
      <wssem:precondition name="ExistingAcctPrecond"
        wssem:modelReference="POOntology#AccountExists">
      <wssem:effect name="ItemReservedEffect"
        wssem:modelReference="POOntology#ItemReserved"/>
    </operation>
  </interface>
```

The schema that shows the input and output message definitions is given below.

```
<types>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=" http://www.ourdemos.com/purchaseorder/wsdl/PurchaseOrder/"
    xmlns="http://www.ourdemos.com/purchaseorder/wsdl/PurchaseOrder/">
     <!—Semantic annotations for these complex types are given in their respective type
       definitions>
      <xs:complexType name="processPurchaseOrderRequest">
        <xs:all>
          <xs:element  name="billingInfo" type="xs:POBilling"/>
          <xs:element  name="orderItem" type="xs:POItem"/>
        </xs:all>
      </xs:complexType>
      <!—Semantic annotation is added directly to this leaf element />
      <xs:element name= "processPurchaseOrderResponse" type="xs:string
        wssem:modelReference="POOntology#OrderConfirmation"/>
      </xs:schema>
```

```
</types>
```

In this example, the input is a complex type while the output is a simple type. To annotate simple types we use the extensibility of xsd:element. An excerpt from XML Schema for xsd:element is shown below. It indicates that an element can be extended with 'any attributes with non-schema namespace'. We use *wssem* namespace with *modelReference* attribute to associate annotations to element.

```
<element
  ….
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded)  : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  ……
  type = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, ((simpleType | complexType)?, (unique | key | keyref)*))
</element>
```

Annotating Complex Types

Complex types can be annotated in multiple ways. We propose two alternate schemes for annotating complex types:

- Bottom Level Annotation: Annotating at leaf element level
- Top Level Annotation : Annotating at complex type level

In the bottom level annotation, all the leaf elements in a complex type can be annotated. The advantage of this approach is that it is simple. It assumes that there is a corresponding concept in the domain model which maps to each leaf element. In cases where there is no corresponding concept, then its semantic annotation can be left unspecified. The disadvantage of this approach is that it assumes there is one-to-one correspondence between the schema elements and the concepts in the domain model. When the associations are one-to-many or many-to-one specifying associations at each leaf element may not be possible. In top level annotation, complex types themselves are annotated with the semantic concept. The advantage of this approach is that it allows for the specification of complex mappings between the elements contained in complex types and the concepts defined in domain models. The disadvantage of this approach is that it tends to be complex. In this technical note, we accommodate both types of annotations. Below, we describe mechanisms for both approaches.

Bottom Level Annotation: Annotating leaf elements in a complex type

In some cases, the elements of a complex type will correspond in a one-to-one fashion with the concepts in a domain model. To accommodate this case, a simple and direct method is provided. We support this by adding a *wssem:modelReference* attribute to the relevant schema element or attribute definition. We allow for multiple annotations to be associated with an element. The schema for associating a modelReference attribute is:

```
<attribute name="modelReference" type="anyURI" use=”optional”/>
```

An example of annotating the leaf nodes of complex type with *wssem:modelReference* attribute is shown below.

```
<complexType name="POItem">
      <all>
         <element name="dueDate" type="dateTime"
            wssem:modelReference="POOntology#DueDate"/>
         <element name="quantity" type="float"
            wssem:modelReference ="POOntology#Quantity"/>
         <element name="EANCode" type="string"
            wssem:modelReference ="POOntology#ItemCode"/>
         <element name="itemDesc" type="string"
            wssem:modelReference ="POOntology#ItemDesc"/>
      </all>
   </complexType>
```

Top Level Annotation: Annotating a complex type using schema mapping function

A Complex type can have a semantic annotation via *wssem:modelReference* attribute. This attribute can point to the corresponding high level concept in an ontology. This high level semantic annotation on a complex type can be used during discovery to make a preliminary determination on whether two structures that are to be matched are semantically related. For example, a complex type 'chip' in a xsd schema can have a semantic annotation 'Microprocessor Chip' from the ontology. This can provide the required context for a schema element during discovery of services. For instance, a discovery service could mistake a 'chip' to mean a chip in a gambling domain but having a semantic annotation can help clarify the context. The annotation 'Microprocess Chip' would be part of electronics domain ontology and the related properties such as electrical properties of a microprocessor etc. help set the context for discovery. At this point in the technical note, this *wssem:modelReference* attribute on a complex type does not specify semantic annotations for any elements within a complex type. Semantic annotations for elements contained within a complextype would have to specified using *wssem:schemaMapping* attribute (explained below). The schema for associating a *modelReference* attribute is:

```
<attribute name="modelReference" type="anyURI" use="optional"/>
```

To capture the semantics for the elements within a complex type, we present a schema mapping function that can be associated with a complex type. The schema for associating a *schemaMapping* function is:

```
<attribute name="schemaMapping" type="anyURI" use="optional"/>
```

An excerpt from XML Schema for a complex type is shown below. It indicates that complex types can be extended with 'any attributes with non-schema namespace'. We use *wssem* namespace with *schemaMapping* attribute to associate annotations to complex types.

```
<complexType
  abstract = boolean : false
  block = (#all | List of (extension | restriction))
  final = (#all | List of (extension | restriction))
```

```
  id = ID
  mixed = boolean : false
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleContent | complexContent | ((group | all | choice | sequence)?,
((attribute | attributeGroup)*, anyAttribute?))))
</complexType>
```

Just like the way we are agnostic the choice of the domain modeling language, we are also agnostic to the choice of schema mapping language. Our approach provides a mechanism to associate zero or more schema mapping functions with a complex type without any restrictions on the choice of the schema mapping language. The following excerpt from purchase order example shows how XSLT can be used as a schema mapping language to specify associations between XSD elements and OWL concepts. Detailed examples showing mapping using XSLT and XQuery are shown in Appendices A and B respectively.

For mapping XML entities with OWL concepts we support both the alternatives given below:

1. XML complexTypes are mapped to OWL classes
2. XML elements are mapped to OWL properties

In POBilling.xsd, the complexType POBilling corresponds directly to the OWL concept Billing (Appendix C). Hence the *modelReference* attribute is used to associate it (POBilling) to the OWL concept Billing.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified"
   elementFormDefault="unqualified"
   targetNamespace="http://www.ourdemos.com/purchaseorder/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1=http://www.ourdemos.com/purchaseorder/
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
   xmlns:POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl">

  <include schemaLocation="POAddress.xsd"/>
  <include schemaLocation="Account.xsd"/>
   <complexType name="POBilling" wssem:modelReference=”POOntology#Billing”>
      <all>
         <element name="shipToAddress" type="xsd1:POAddress" />
         <element name="billToAddress" type="xsd1:POAddress"  />
         <element name="accountID" type="xsd1:string />"
      </all>
   </complexType>
</schema>
```

In POAddress.xsd, the complexType POAddress does not map directly to the OWL concept in the ontology. Hence, the *schemaMapping* attribute is used to point to an XSLT transformation.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified"
   elementFormDefault="unqualified"
   targetNamespace="http://www.ourdemos.com/purchaseorder/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1=”http://www.ourdemos.com/purchaseorder/”
```

```
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
   xmlns:POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl">

<import location="WSSemantics.xsd" namespace="
http://www.ibm.com/xmlns/WebServices/WSSemantics/"/>

   <complexType name="POAddress"
    wssem:schemaMapping=”http://www.ibm.com/schemaMapping/POAddress.xsl”>
     <all>
          <element name="recipientInstName" type="string" />
          <element name="streetAddr1" type="string" />
          <element name="streetAdd2" type="string" />
          <element name="city" type="string" />
          <element name="state" type="string" />
          <element name="zipCode" type="string" />
          <element name="country" type="string" />
     </all>
   </complexType>
</schema>
```

The schema mapping used by POAddress.xsd, POAddress.xsl is shown below.

POAddress.xsl

```
<?xml version='1.0' ?>
<xsl:transform version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <xsl:template match="/">
    <Address rdf:ID="Address1">
          <has_Receiver rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/recepientInstName"/>
          <has_StreetAddress rdf:datatype="xs:string">
          <xsl:value-of select="concat(POAddress/streetAddr1,POAddress/streetAddr2)"/>
          </has_StreetAddress >
          <has_City rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/city"/>
          </has_City>
          <has_State rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/state"/>
          </has_State>
          <has_ZipCode rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/zipCode"/>
          </has_ZipCode>
          <has_Country rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/country"/>
          </has_Country>
    </Address>
  </xsl:template>
</xsl:transform>
```

The use of *wssem:modelReference* at the leaf level and *wssem:schemaMapping* at the complex type level is mutually exclusive within an XML schema complex type. In essence, if a schema mapping is given for the same complex type, it overrides the individual modelReferences specified at the leaf elements.

We summarize our approach to annotating complex types below.

1. A complex type can have an optional semantic annotation via wssem:modelReference. The primary purpose of this is to enable a quick/rough match during discovery without going into the details of the structure.
2. The elements within a complex type can have semantic annotations specified in one of two ways.
    a. Via wssem:modelReference at each leaf element contained in a complex type and/or
    b. Via wssem:schemaMapping attribute at the complex type that specifies complex mappings between the elements contained in a complex type and the concepts in an external domain model/ontology

We suggest the following rules to resolve any potential conflicts.

1. If ws:schemaMapping is specified for a complex type and wssem:modelReference is used at each leaf element level within the complex type, then the schemaMapping overrides the leaf element level modelReferences.
2. The semantic annotation given at the complex type using ws:modelReference applies only to the complex type itself – no inferences can be made about the semantic annotations for elements within a complex type even if the semantic concept specified at the complex type potentially has seemingly corresponding concepts to elements in the complex type. To specify annotations for elements within a complex type, either wssem:modelReference has to be used at the leaf elements of a complex type or ws:schemaMapping has to be used to specify semantic annotation for multiple elements at once.
3. If there is a complex type contained within a complextype, then a combination of wssem:modelReference and wssem:schemaMapping can be used. For example, POBilling.xsd has two complex elements shipToAddress and billToAddress each of type POAddress and an accountID element of type string. In such a case, accountID element can have a wssem:modelReference to directly point to a semantic concept in the ontology while the semantic annotations for shipToAddress and billToAddress can be specified once at the level of POAddress complex type.

In this section, we have provided a mechanism for associating annotations with the wsdl:input and wsdl:output elements in a WSDL document. In the next section, we explain how a wsdl:operation can be extended to accommodate preconditions and effects.

4.2 Preconditions

A precondition defines a set of assertions that must be met before a Web service operation can be invoked. They can specify requirements that must be met, such as “must have an existing account with this company,” or restrictions, such as, “Only US customers can be served”. Preconditions are specified as child elements of the operation for which the precondition is defined. The schema for a precondition is shown below.

```
<xsd:element name=precondition>
  <xsd:complexType>
```

```
        <xsd:complexContent
                <xsd:restriction base="xsd:anyType">
                        <xsd:attribute name=”name” type=”xsd:string” />
                        <xsd:attribute name="modelReference" type="xsd:anyURI" />
                        <xsd:attribute name="expression" type="xsd:string" />
                </xsd:restriction>
        </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

The *precondition* element is defined as follows:

/precondition

This element specifies the semantic annotation for the parent operation.

/precondition/@name

The *name* attribute specifies an identifier unique within the set of preconditions in the WSDL document.

/precondition/@modelReference

The modelReference attribute specifies the URI of the part of a semantic model that describes the precondition. The modelReference attribute and the expression attribute are mutually exclusive.

/precondition/@expression

This is an expression defining the precondition. The format of the expression is defined by the semantic representation language used to express the semantic model. The modelReference attribute and the expression attribute are mutually exclusive.

```
<interface name="PurchaseOrder">
 <operation name="processPurchaseOrder” pattern=wsdl:in-out>
       <input messageLabel = ”processPurchaseOrderRequest"
         element="tns:processPurchaseOrderRequest"/>
       <output messageLabel ="processPurchaseOrderResponse"
         element="processPurchaseOrderResponse"/>
       <!—Precondition and effect are added as extensible elements on an operation>
       <wssem:precondition name="ExistingAcctPrecond"
          wssem:modelReference="POOntology#AccountExists">
       <wssem:effect name="ItemReservedEffect"
          wssem:modelReference="POOntology#ItemReserved"/>
     </operation>
</interface>
```

Preconditions and effects are specified as child elements of the *operation* element. Each operation may have at most one *precondition* and as many *effects as possible*. We allow for at most one precondition to keep the specification simple. We believe that complex or conditional preconditions should be expressed in the domain/semantic model. For example, in OWL, a set of preconditions (effects) may be defined via logical expressions to facilitate operations such as ‘and’, ‘or’, ‘xor’, etc. in evaluating these expressions. In this technical note, we assume that the underlying (semantic) representation language supports capturing such multiple preconditions into a single high-level precondition that can be referenced at the *operation* element level within a WSDL specification.

Developing markup languages for representing preconditions and effects is an area of active research. Different communities are working on this. For example, semantic Web community is working on Semantic Web Rule Language (SWRL) [SWRL] and the modeling community is working on Object Constraint Language (OCL) [OCL] etc. While we recognize the importance of preconditions and effects in this document and provides hooks for accommodating them while describing Web Services, we believe that it is still very early to use them in production systems. Much more research and more concrete examples are required to help better understand matching and evaluation of preconditions and effects while (semi) automatically matching Web Services descriptions. We defer such a discussion to a later revision of this document.

The schema for a request-response operation with optional preconditions and effects is shown below.

```
<group name="request-response-operation">
    <sequence>
      <element ref="wsdl:input"/>
      <element ref="wsdl:output"/>
      <element ref="wsdl:fault" minOccurs="0" maxOccurs="unbounded"/>
      <element ref=”wssem:precondition” minOccurs="0" maxOccurs="1""/>
      <element ref=”wssem:effect” minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </group>
```

The grammar for a request-response operation with optional preconditions and effects is shown below.

```
<wsdl:operation name="nmtoken" parameterOrder="nmtokens">
   <wsdl:input name="nmtoken"? message="anyURI"/>
   <wsdl:output name="nmtoken"? message="anyURI"/>
   <wssem:precondition name=”nmtoken”? [modelReference=”anyURI”/> |
expression=”string”]?
   <wssem:effect name=”nmtoken”? [modelReference=”anyURI”/> | expression=”string”]*
   <wsdl:fault name="nmtoken" message="anyURI" modelReference=”anyURI” />*
</wsdl:operation>
```

4.3 Effects

An effect defines the result of invoking an operation. It can simply state that the output is returned or it can make statements about what changes in the state are expected to occur upon invocation of the service. For example, “the new account balance will be available” or “the credit card account will be debited.”

```
<xsd:element name=”effect”>
  <xsd:complexType>
        <xsd:complexContent
                <xsd:restriction base="xsd:anyType">
                        <xsd:attribute name=”name” type=”xsd:string” />
```

```
                        <xsd:attribute name="modelReference" type="xsd:anyURI" />
                        <xsd:attribute name="expression" type="xsd:string" />
                </xsd:restriction>
        </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

The *effect* element is a child of the *operation* element or the *fault* element. The *effect* element is defined as follows:

/effect

This element specifies the semantic annotation of the effect that applies to the parent operation.

/precondition/@name

The *name* attribute specifies an identifier unique within the set of effects in the WSDL document.

/effect/@modelReference

The *modelReference* attribute specifies the URI of the part of a semantic model that describes the effect. The *modelReference* attribute and the *expression* attribute are mutually exclusive.

/effect/@expression

This is an expression defining the effect. The format of the expression is defined by the semantic representation language used to express the semantic model. The *modelReference* attribute and the *expression* attribute are mutually exclusive.

An effect can be associated with a fault. For example, an effect of 'credit card not processed' might occur when a fault 'InvalidCreditCardError' occurs. The fault effect is attached to the operation's child *fault* element via a *modelReference* attribute.

Note: Please see the note in Preconditions section (3.3) to see how to use the 'annotation' element in *operation* to capture effects if you are using an older version of WSDL schema where *operation* does not support extensibility elements.

4.4 Service Categorization

The purpose of annotating services is to enable dynamic discovery of services. This is possible when services are published, catalogued and annotated with semantics. In this technical note, so far, we focused on how to annotate services. We now present a mechanism to add categorization information to services which could be used while publishing services in registries such as UDDI. This aids in service discovery by narrowing the range of candidate services. The categorization can be used as input when the service is published in a UDDI registry or it can constitute the effective categorization when the service is made available via Web Services Inspection Language [WSIL] or some other solution-specific means. Service categorization is also aimed at supporting specialized taxonomies of middleware or utility services such as mediators. Our objective in this technical note is to ensure that there is basic and high-level categorization information about a service and leave the details of actual categorization system and maintenance of taxonomies, classifications, etc, to the underlying service registries. As has been noted

earlier, this concept of associating service categorization information is borrowed from OWL-S but here it is adapted to work within the parameters of WSDL specification.

We model a service category using the extensibility elements on a WSDL interface element. This assumes that all operations in the interface of a given WSDL document belong to the same category. The schema for the service categorization element is given below.

```
<element name=”category” maxOccurs="unbounded">
  <complexType>
         <complexContent
                  <extension base=”wsdl:interface”>
                     <attribute name=”categoyname” type=”NCName” use=”required”/>
                     <attribute name=”taxonomyURI” type=”anyURI” use=”required”/>
                     <attribute name="taxonomyValue" type="String" use="optional"/>
                     <attribute name="taxonomyCode" type="integer" use="optional"/>
   </complexContent>
  </complexType>
</element>
```

/category/@categoryName
: The name of the category within a taxonomy.

/category/@taxonomyURI
: A URI reference to the taxonomy definition. It is generally the URL where the taxonomy can be obtained.

/category/@taxonomyValue
: The value associated with a category in the taxonomy.

/category/@taxonomyCode
: The code associated with a category in the taxonomy.

Multiple *category* elements can be used to specify that the service falls into multiple categories. A *category* element specifies one categorization. For example, the following categorization references the NAICS taxonomy and specifies that the services falls into the 443112 - Electronics category.

```
<wssem:category name= “Electronics” taxonomyURI=”http://www.naics.com/” taxonomyCode=”
443112” />
```

4.5 Interpreting Semantic Annotations

Input
An annotation on an element or complex type is considered to be an input semantic annotation if the message is referenced by an element attribute of an input element. The semantic annotation is associated with the operation defined by the parent element of this input element. If the message is referenced by multiple operations, the semantic annotation applies to each. If the annotation is applied to a complex type that is referenced by multiple messages from multiple messages, the semantic annotation applies each operation that references any of those messages for input.

Output
An annotation on an element or complex type is considered to be an output semantic annotation if the message is referenced by an element attribute of an output element. The semantic annotation is associated with the operation defined by the parent element of this output element. If the message is referenced by multiple operations, the semantic annotation applies to each. If the annotation is applied to a complex type that is referenced by multiple messages from multiple messages, the semantic annotation applies each operation that references any of those messages for output.

Precondition
A precondition annotation referenced from an *operation* element applies to that operation.

Effect
An effect annotation referenced from an *operation* element applies to that operation. An effect annotation on a fault applies to the operation for which the fault is defined.

4.6 Publishing Web Services with Semantics in UDDI

This section is non-prescriptive and is provided to describe how this technical note relates to Web services publication and discovery.

Finding suitable Web Services depends on the quality of the search facilities available for service requesters. Web services registries can play an important role in describing the available services and providing these search facilities for finding suitable services. As an industry-backed registry for Web Services, UDDI plays a central role in helping requesters find suitable services. Unfortunately, the current search functions in UDDI are limited in their support for making automatic service selection decisions. Its keyword and category based search facilities are insufficient for selecting suitable services for a given requirement. As of version 3.0, UDDI does not capture the relationships between entities in its directory and therefore is not capable of making use of the semantic information to infer relationships during search. Moreover, UDDI currently does not facilitate matching at the service capability level. Ways to address these limitations are being discussed in the context of upcoming versions of UDDI specifications.

One topic under discussion is the use of OWL ontologies to represent UDDI taxonomies, thereby enabling semantic matching at the category level. The semantic annotations defined in this document both complement and extend such an approach. The semantic annotations enhance the functional description so that while the classification can effectively narrow the set of possible matches, the WSDL annotations can allow specific functional matching and enable the discovery of composite services.

UDDI already specifies how to publish WSDL files to the registry. Semantically annotated WSDL files can be published in similar fashion. A specific approach to performing such semantic matching in UDDI is presented in [Semantic Matching in UDDI; SVSM03].

4.7 Future Extensions

OWL-S prescribes a notion of conditional outputs and conditional effects. We are currently assessing methods for representing these concepts.

5. WSDL 1.1 Support

The mechanism for semantic annotation described in this specification can also be applied to WSDL 1.1 conformant Web services descriptions. All the XML attributes and elements defined in this specification apply without modification to the WSDL 1.1 descriptions. However, in some cases they are applied to different elements in the WSDL document structure.

Input and Output

Schema mapping of XML Schema types is the same. In addition, a *schemaMapping* or *modelReference* attribute may be added to a *part* element (under a *message* element) to specify an input or output annotation that applies to the entire message part. These elements are part of the *portType* structure in WSDL1.1 which generally corresponds to the WSDL2.0 *interface* structure.

Preconditions and Effects

The *precondition* and *effect* attributes are child attributes of the *operation* element in a *porttype*. Operation is an extensible element in schema http://schemas.xmlsoap.org/wsdl/. However, if you are using an older version of XML schema for WSDL, then an *operation* of a *portType* is not extensible. Therefore, we recommend using the documentation element on an operation to capture the semantics of preconditions and effects. That is shown below.

```
<portType name="PurchaseOrder">
  <operation name="processPurchaseOrder" parameterOrder="billingInfo orderItem">
      <documentation>
        <wssem:precondition name="PreExistingAcctPrecond"
          wssem:modelReference="POOntology#AccountExists">
        <wssem:effect name="ItemReservedEffect"
          wssem:modelReference="POOntology#ItemReserved"/>
      </documentation>
      <input message="tns:processPurchaseOrderRequest"
          name="processPurchaseOrderRequest"/>
      <output message="tns:processPurchaseOrderResponse"
          name="processPurchaseOrderResponse"/>
  </operation>
</portType>
```

Faults

In WSDL 1.1, faults are specified as messages that are generated when a particular condition arises. Annotations for fault messages are done as any other output message.

6. References

[keywords]

S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[METEOR-S]

METEOR-S: Semantic Web Services and Processes, http://lsdis.cs.uga.edu/Projects/METEOR-S/

[OWL]

OWL Web Ontology Language Overview, http:w3.org/TR/owl-features/

[OWL-S]

Web Ontology Language for Web Services, http://www.daml.org/services

[Semantic Matching in UDDI]

"External Matching in UDDI" J. Colgrave, R. Akkiraju, R. Goodwin 2004. In the proceedings of IEEE International Conference on Web Services (ICWS) July 2004. San Diego. USA.

[SOAP]

SOAP Version 1.2 Part 1: Messaging Framework, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[SVSM03]

Sivashanmugam, K., Verma, K., Sheth, A., Miller, J., Adding Semantics to Web Services Standards, Proceedings of the 1st International Conference on Web Services (ICWS'03), Las Vegas, Nevada (June 2003) pp. 395-401.

[SWSA]

Semantic Web Services Initiative Architecture Committee (SWSA), http://www.daml.org/services/swsa/

[SWSL]

Semantic Web Services Language (SWSL) Committee. http://www.daml.org/services/swsl/

[SVS04]

K. Sivashanmugam, K. Verma, A. P. Sheth, Discovery of Web Services in a Federated Registry Environment, Proceedings of IEEE Second International Conference on Web Services, June, 2004, pp. 270-278

[URI]

T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998, http://www.ietf.org/rfc/rfc2396.txt

[MWSDI]

K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and J. Miller, METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Journal of Information Technology and Management, Special Issue on Universal Global Integration, Vol. 6, No. 1 (2005) pp. 17-39. Kluwer Academic Publishers.

[Web Services Semantics]

Web Services Semantics: A View of Semantics in Services Oriented Architecture. WebServicesSemanticsWhitePaper.htm

[WSDL]

W3C Note, Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[WSDL2]

W3C Working Draft, Services Description Language (WSDL) Version 2.0 Part 1: Core Language specification, http://www.w3.org/TR/2004/WD-wsdl20-20040803/

[WSDL2 Diff]

What's new in WSDL 2.0 http://Webservices.xml.com/lpt/a/ws/2004/05/19/wsdl2.html

[WSDL-S]

WSDL-S: Adding semantics to WSDL – White paper, http://lsdis.cs.uga.edu/projects/WSDL-S/wsdl-s.pdf

[WSIL]

Web Services Inspection Language (WS-Inspection). Nov. 2001. http://www-106.ibm.com/developerworks/Webservices/library/ws-wsilspec.html

[W3CWSA]

W3C Note, Web Services Architecture, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[WSMO]

Web Service Modeling Ontology, http://www.wsmo.org/

[XMI]

OMG, XML Metadata Interchange Specification 2.0, http://www.omg.org/docs/formal/03-05-02.pdf

[UML]

OMG, Unified Modeling language (UML) Version 1.5, http://www.omg.org/technology/documents/formal/uml.htm

[XPath]

XML Linking Language (XLink) Version 1.0, http://www.w3.org/TR/2001/REC-xlink-20010627/

[XMLNamespace]

W3C Recommendation, Namespaces in XML, http://www.w3.org/TR/1999/REC-xml-names-19990114

[XML]

Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/REC-xml

[XMLSchema1]

W3C Recommendation, XML Schema Part 1: Structures, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[XMLSchema2]

W3C Recommendation, XML Schema Part 2: Datatypes, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[SWRL]

W3C Member Submission, May 21, 2004, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

[OCL]

OMG, UML 2.0 OCL Specification, http://www.omg.org/docs/ptc/03-10-14.pdf

7. Appendix A: Specifying Schema mapping Using XSLT

In this section, we show how XSLT can be used to represent the schema mapping between XSD and OWL elements.

POAddress.xsd is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified"
   elementFormDefault="unqualified"
   targetNamespace="http://www.ourdemos.com/purchaseorder/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1=”http://www.ourdemos.com/purchaseorder/”
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
   xmlns:POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl">

<import location="WSSemantics.xsd" namespace="
http://www.ibm.com/xmlns/WebServices/WSSemantics/"/>

   <complexType name="POAddress"
    wssem:schemaMapping=”http://www.ibm.com/schemaMapping/POAddress.xsl#input-
doc=doc(“POAddress.xml”)”>
      <all>
           <element name="recipientInstName" type="string" />
          <element name="streetAddr1" type="string" />
          <element name="streetAdd2" type="string" />
          <element name="city" type="string" />
          <element name="state" type="string" />
          <element name="zipCode" type="string" />
          <element name="country" type="string" />
      </all>
   </complexType>
</schema>
```

The schema mapping used by POAddress.xsd, **POAddress.xsl is shown below**

POAddress.xsl

```
<?xml version='1.0' ?>
<xsl:transform version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns:Address="http://www.ibm.com/schemaMapping/POAddress.xsl"
xmlns:POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl">
  <xsl:template match="/">
    <POOntology:Address rdf:ID="Address1">
          <POOntology:has_Receiver rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/recepientInstName"/>
          </POOntology:has_Receiver>
          <POOntology:has_StreetAddress rdf:datatype="xs:string">
          <xsl:value-of select="concat(POAddress/streetAddr1,POAddress/streetAddr2)"/>
          </POOntology:has_StreetAddress >
          <POOntology:has_City rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/city"/>
          </POOntology:has_City>
          <POOntology:has_State rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/state"/>
          </POOntology:has_State>
          <POOntology:has_Country rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/country"/>
          </POOntology:has_Country>
          <POOntology:has_ZipCode rdf:datatype="xs:string">
          <xsl:value-of select="POAddress/zipCode"/>
          </POOntology:has_ZipCode>
    </POOntology:Address>
```

```
  </xsl:template>
</xsl:transform>
```

POItem.xsd is given below

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified"
   elementFormDefault="unqualified"
   targetNamespace="http://www.ourdemos.com/purchaseorder/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1="http://www.ourdemos.com/purchaseorder/"
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
   xmlns:POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl">

<import location="WSSemantics.xsd"
namespace="http://www.ibm.com/xmlns/WebServices/WSSemantics/"/>
<complexType name="POItem"
wssem:schemaMapping="http://www.ibm.com/schemaMapping/POItem.xsl#input-
doc=doc("POItem.xml")">
      <all>
         <element name="dueDate" type="dateTime" />
         <element name="quantity" type="float" />
         <element name="EANCode" type="string" />
         <element name="itemDesc" type="string" />
       </all>
</complexType>
</schema>
```

The POItem schema is also updated to include semantic references. The schema mapping used by POItem.xsd, **POItem.xsl is shown below**.

POItem.xsl

```
<?xml version='1.0' ?>
<xsl:transform version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns:Item="http://www.ibm.com/schemaMapping/POItem.xsl"
xmlns:POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl">
<xsl:template match="/">
          <POOntology:Item rdf:ID="Item1">
          <POOntology:has_dueDate rdf:datatype="xs:dateTime">
          <xsl:value-of select="POItem/dueDate"/>
          </POOntology:has_dueDate >
          <POOntology:has_quantity rdf:datatype="xs:float">
          <xsl:value-of select="POItem/quantity"/>
          </POOntology:has_quantity>
          <POOntology:has_EANCode rdf:datatype="xs:string">
          <xsl:value-of select="POItem/EANCode"/>
          </POOntology:has_EANCode >
          <POOntology:has_itemDesc rdf:datatype="xs:string">
          <xsl:value-of select="POItem/itemDesc"/>
          </POOntology:has_itemDesc>
          </POOntology:Item>
</xsl:template>
</xsl:transform>
```

8. Appendix B: Specifying Schema mapping using XQuery

This appendix shows examples of how mappings between XSD schema elements and concepts in domain model can be specified using XQuery. The mappings written as XQuery functions are organized as modules to enable reusability. Once specified, we show how these mappings can be invoked using an XQuery.

We show the mapping for POAddress.xsd in this section.

POAddress.xsd is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified"
   elementFormDefault="unqualified"
   targetNamespace="http://www.ourdemos.com/purchaseorder/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1=”http://www.ourdemos.com/purchaseorder/”
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics">

<import location="WSSemantics.xsd" namespace="
http://www.ibm.com/xmlns/WebServices/WSSemantics/"/>

   <complexType name="POAddress"
    wssem:schemaMapping=”http://www.ibm.com/schemaMapping/POAddress.xq#input-
doc=doc(“POAddress.xml”)”>
      <all>
          <element name="streetAddr1" type="string" />
          <element name="streetAdd2" type="string" />
          <element name="poBox" type="string" />
          <element name="city" type="string" />
          <element name="zipCode" type="string" />
          <element name="state" type="string" />
          <element name="country" type="string" />
          <element name="recipientInstName" type="string" />
      </all>
   </complexType>
</schema>
```

POAddress.xsd defines an address in a purchase order. It contains a *wssem:schemaMapping* attribute that points to a schema mapping that shows how the elements of the POAddress complex type are defined by concepts in a semantic model. This mapping defines the meaning of the information carried in the XML element.

The schema mapping used by POAddress.xsd, **POAddress.xq** is shown below.

POAddress.xq

```
xquery version "1.0";
declare namespace Address = "http://www.ibm.com/schemaMapping/POAddress.xq";
declare namespace xs = "http://www.w3.org/2001/XMLSchema" ;
declare namespace POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl" ;
declare namespace owl="http://www.w3.org/2002/07/owl#";
declare namespace rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#";
for $a in doc("POAddress.xml")/POAddress
return
```

```
        <POOntology:Address rdf:ID="Address1">
        <POOntology:has_StreetAddress rdf:datatype="xs:string">
                { fn:concat($a/streetAddr1 , " ", $a/streetAddr2 ) }
        </POOntology:has_StreetAddress>
        <POOntology:has_City rdf:datatype="xs:string">
                { fn:string($a/city) }
        </POOntology:has_City>
        <POOntology:has_State rdf:datatype="xs:string">
                { fn:string($a/state) }
        </POOntology:has_State>
        <POOntology:has_Country rdf:datatype="xs:string">
                { fn:string($a/country) }
        </POOntology:has_Country>
        <POOntology:has_POBox rdf:datatype="xs:string">
                { fn:string($a/poBox) }
        </POOntology:has_POBox>
        <POOntology:has_ZipCode rdf:datatype="xs:string">
                { fn:string($a/zipCode) }
        </POOntology:has_ZipCode>
        <POOntology:has_Receiver rdf:datatype="xs:string">
                { fn:string($a/recipientInstName) }
        </POOntology:has_Receiver>
        </POOntology:Address>
```

Result of the mapping

```
<POOntology:Address rdf:ID="Address1">
<POOntology:has_StreetAddress rdf:datatype="xs:string">
224 Boyd
</POOntology:has_StreetAddress>
<POOntology:has_City rdf:datatype="xs:string">
Athens
</POOntology:has_City>
<POOntology:has_State rdf:datatype="xs:string">
Georgia
</POOntology:has_State>
<POOntology:has_Country rdf:datatype="xs:string">
US
</POOntology:has_Country>
<POOntology:has_POBox rdf:datatype="xs:string">
897656
</POOntology:has_POBox>
<POOntology:has_ZipCode rdf:datatype="xs:string">
30602
</POOntology:has_ZipCode>
<POOntology:has_Receiver rdf:datatype="xs:string">
XYZ
</POOntology:has_Receiver>
</POOntology:Address>
```

POItem.xsd is given below

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified"
   elementFormDefault="unqualified"
   targetNamespace="http://www.ourdemos.com/purchaseorder/"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1="http://www.ourdemos.com/purchaseorder/"
```

```
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics">

<import location="WSSemantics.xsd" namespace="
http://www.ibm.com/xmlns/WebServices/WSSemantics/"/>

   <complexType name="POItem"
      wssem:schemaMapping=”http://www.ibm.com/schemaMapping/POItem.xq#input-
doc=doc(“POItem.xml”)”>
      <all>
         <element name="dueDate" type="dateTime" />
         <element name="quantity" type="float" />
         <element name="EANCode" type="string" />
         <element name="itemDesc" type="string" />
      </all>
   </complexType>
</schema>
```

The POItem schema is also updated to include semantic references. The schema mapping used by POItem.xsd, **POItem.xq** is shown below.

POItem.xq

```
xquery version "1.0";
declare namespace Item = "http://www.ibm.com/schemaMapping/POItem.xq";
declare namespace xs = "http://www.w3.org/2001/XMLSchema" ;
declare namespace POOntology="http://www.ibm.com/ontologies/PurchaseOrder.owl" ;
declare namespace owl="http://www.w3.org/2002/07/owl#";
declare namespace rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#";
for $a in doc("POItem.xml")/POItem
return
          < POOntology:Item rdf:ID="Item1">
          < POOntology:has_quantity rdf:datatype="xs:float">
                    { fn:string($a/quantity) }
          </ POOntology:has_quantity>
          < POOntology:has_dueDate rdf:datatype="xs:dateTime">
                    { fn:string($a/dueDate) }
          </ POOntology:has_dueDate>
          < POOntology:has_EANCode rdf:datatype="xs:string">
                    { fn:string($a/EANCode) }
          </ POOntology:has_EANCode>
          < POOntology:has_itemDesc rdf:datatype="xs:string">
                    { fn:string($a/itemDesc) }
          </ POOntology:has_itemDesc>
          </ POOntology:Item>
```

Result of the mapping

```
<POOntology:Item rdf:ID="Item1">
<POOntology:has_quantity rdf:datatype="xs:float">
5.0
</POOntology:has_quantity>
<POOntology:has_dueDate rdf:datatype="xs:dateTime">
2001-11-19T00:00:00.00000
</POOntology:has_dueDate>
<POOntology:has_EANCode rdf:datatype="xs:string">
A6253SAW
```

```
</POOntology:has_EANCode>
<POOntology:has_itemDesc rdf:datatype="xs:string">
Belkin Wireless Router
</POOntology:has_itemDesc>
</POOntology:Item>
```

9. Appendix C: Purchase Order Ontology

PurchaseOrder.owl

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns="http://www.ibm.com/ontologies/PurchaseOrder.owl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.ibm.com/ontologies/PurchaseOrder.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="ItemDesc"/>
  <owl:Class rdf:ID="Item"/>
  <owl:Class rdf:ID="DueDate"/>
  <owl:Class rdf:ID="PreCondition"/>
  <owl:Class rdf:ID="PostalCode"/>
  <owl:Class rdf:ID="City"/>
  <owl:Class rdf:ID="Billing"/>
  <owl:Class rdf:ID="AccountExists">
    <rdfs:subClassOf rdf:resource="#PreCondition"/>
  </owl:Class>
  <owl:Class rdf:ID="Country"/>
  <owl:Class rdf:ID="Account"/>
  <owl:Class rdf:ID="Quantity"/>
  <owl:Class rdf:ID="Address"/>
  <owl:Class rdf:ID="Effect"/>
  <owl:Class rdf:ID="OrderConfirmation"/>
  <owl:Class rdf:ID="StreetAddress"/>
  <owl:Class rdf:ID="State"/>
  <owl:Class rdf:ID="ItemReserved">
    <rdfs:subClassOf rdf:resource="#Effect"/>
  </owl:Class>
  <owl:Class rdf:ID="Receiver"/>
  <owl:Class rdf:ID="ItemCode"/>
  <owl:Class rdf:ID="ZipCode">
```

```
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf rdf:resource="#PostalCode"/>
  </owl:Class>
  <owl:Class rdf:ID="EanCode">
    <rdfs:subClassOf rdf:resource="#ItemCode"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="has_billingAddress">
    <rdfs:range rdf:resource="#Billing"/>
    <rdfs:domain rdf:resource="#Billing"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="has_account">
    <rdfs:domain rdf:resource="#Billing"/>
    <rdfs:range rdf:resource="#Account"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="has_shippingAddress">
    <rdfs:range rdf:resource="#Address"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Billing"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="has_StreetAddress">
    <rdfs:domain rdf:resource="#Address"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="has_City">
    <rdfs:domain rdf:resource="#Address"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="has_AccountID">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Account"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="has_State">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Address"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="has_itemDesc">
```

```
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Item"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="has_Quantity">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Item"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="has_POBox">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:FunctionalProperty rdf:ID="has_ZipCode">
    <rdfs:domain rdf:resource="#Address"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="has_Country">
    <rdfs:domain rdf:resource="#Address"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="has_Receiver">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Address"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="has_DueDate">
    <rdfs:domain rdf:resource="#Item"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="has_EANCode">
    <rdfs:domain rdf:resource="#Item"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

10. Appendix D: Mapping Choices

The focus of research in Semantic Web services has been on using ontologies for Web service discovery and composition [SVSM03, WSMO, OWL-S]. It is useful to use ontologies, as they provide semantic normalization between service requests and advertisements, for higher level tasks like finding an appropriate service (discovery) or composing services to aggregate their functionalities. However, during execution of such compositions, just semantic annotations are not enough to automatically map the data (inputs and outputs) of these services. For a valid interaction, the mappings must also be specified as annotations.

There has been significant work in the database area during 1980s and early 1990s on recognizing the need for data interoperability. There have been numerous efforts in schema mapping/merging/transformations, semantic heterogeneity, and use of ontology and description logics for schematic and semantic integration (e.g., see the discussion in [S04]). This was followed by substantial work on schema matching and mapping as part of the Model Management initiative [BM]. There is ongoing work in the above areas especially in the context of the new Web Service technologies and Semantic Web languages (XML, RDF/RDFS, OWL) [POSV04, KS03, SM01, DMDH02, N04, FB02, HG05].

Conceptually, data interoperability can be divided into two parts – schema matching and schema/data mapping. The words matching and mapping have often been used interchangeably in the literature. The rest of this appendix follows the definition given below for schema matching and mapping. Schema matching is the process of finding semantic correspondences between elements of two schemas and mapping deals with the physical representation of the matches established by schema matching and rules for transforming elements of one schema to that of other. In the next two sections, we seek to briefly describe the major advances in this field, as well as to point out where the approach suggested in this specification stands in this space.

Schema matching:

Research in schema matching seeks to provide automated support to the process of finding semantic matches between two schemas. This process is made harder due to heterogeneities at the following levels [CKSTD04, SK93, S99]:

- Syntactic heterogeneity - differences in the language used for representing the elements
- Structural heterogeneity - differences in the types, structures of the elements
- Model / Representational heterogeneity – differences in the underlying models (database, ontologies) or their representations (relational, object-oriented, RDF, OWL)
- Semantic heterogeneity - where the same real world entity is represented using different terms or vice-versa

Approaches to schema matching can be broadly classified as approaches that exploit either just schema information [MWJ99, MZ98*, PSU98, CDD01, MBR01] or schema and instance level information [LC94, DDH01, MHH00]. The implementation of such approaches can be classified as being either rule based [MZ98a, MZ98b, MWJ99, MBR01, MGR02] or learner based techniques [WCL00, DDH01, DLDHD04, NHTHM02, BM01a, BM01b, OTSV04]. The complementary nature of these different approaches has instigated a number of applications

[DLDHD04, DR02, EJX01, RDM04, MHHYHFP01] to use a combination of techniques depending on the nature of the domain or application under consideration. Comprehensive surveys of automatic schema matching approaches are presented in [RB01, DH04].

Much of the work described above is based on homogeneous models (database schemas, ontologies etc) used to represent schemas with the heterogeneity at the syntax, structure or semantic level. Web services are autonomous applications, whose data (inputs/outputs) are defined using XML schema. For interoperation with other Web services, their data elements should be mapped to existing domain models, which are typically represented using OWL, RDF/S or UML. Any attempt at automatically matching Web service schemas to OWL, RDF/S or UML models leads to the problem of heterogeneous models. Transforming from a less expressive model to a more expressive model would usually require humans to supply additional semantics, while transformation in the other direction can be lossy at best. Current work in the area of model management [M04, M05] has focused on developing a generic infrastructure that abstracts operations on models (i.e., schemas) and mappings between models as high level operations which are generic and independent of the data model and application of interest. In the area of Web services, [POSV04] addresses the expressiveness difference between OWL concepts and XML elements by normalizing both the representation to a common graph format.

In this specification, we do not deal with automated schema matching and generation of mappings, but assume that the user provides the mappings that are referenced by the schemaMapping attribute.

Schema/Data Mapping:

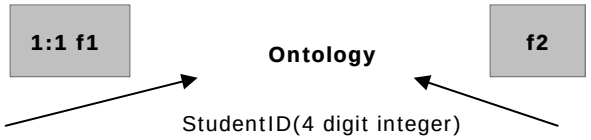
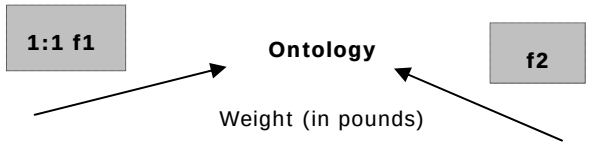
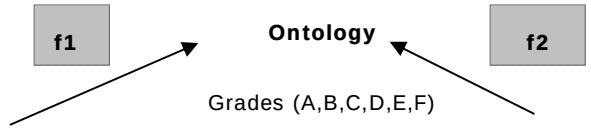
Mapping is the process of representing the matches found as mapping expressions, required to make the mappings operational. Some of the past approaches to representing mappings have been:

- Queries or views using SQL, XSL, XQUERY: global-as-view (GAV) and local as-view (LAV) [CGL01].
- Mapping tables: [KAM03] specifying the dependencies between entities of two schemas using a ‘mapping-table’, an extensionally defined table of value correspondences.
- Bridging axioms in first order logic: [MBDH02], OntoMerge [DMQ03]. The correspondence between two ontologies is expressed as a set of bridging axioms relating classes and properties of the two ontologies.
- Instances in an ontology of mappings: MAFRA [MMSV02], [CM03]. Use ontologies to define the structure of specific mappings and the transformation functions to transfer instances from one ontology to another.
- Languages like Datalog, F-Logic, DLR, Well-founded Object Language suggested by Davison et al. [DKB95], C-OWL [BGHSS02], KIF, LOOM
- Frameworks like [SK93] that illustrate the notion of capturing the semantic proximity between objects by means of abstractions and [LA86] that illustrate the concept of

semantic and dynamic attributes, capturing four types of mapping/translation techniques ranging from syntactic, table, functional and program based mappings.

This specification does not specify a single mapping language to use. However, as domain models like UML, OWL and XSD are represented in XML, the examples in the document represent mappings either as transformations using XSLT or queries using XQuery. Using these mappings from xml schema elements to concepts in a domain model, it is possible to map instances of Web service schemas from one representation to another. We also recognize that the process of capturing mappings can be complicated by heterogeneities at various levels and that XQuery / XSLT might not be suited for all cases. Also, our approach supports the use of languages such as RDF/S and OWL for specifying mappings since it is agnostic to the mapping language used for specifying the mappings.

Table1 illustrates the schema/data conflicts that most mapping representation languages are capable of handling and how instances of one schema can be mapped to instances of another using a common ontology.

Schema/Data Conflicts	**Description/ Example**	**Nature of mapping function**
Data Representation conflict	Different data types / representations  StudentID (4 digit integer) StudentID(9 digit integer)	The mapping function f2 will largely depend on application / domain requirements. *Note: While mapping in the direction of f2 can be well defined, $f2^{-1}$ can not.
Data Scaling conflict	Representations using different units and measures  **WS1** **WS2** Weight (in pounds) Weights (in kilograms)	The mapping function f2 or its inverse $f2^{-1}$ can be automatically generated using a look up table and are well defined.
Data Precision conflict	Represented using different precisions  **WS1** **WS2** Marks (1-100) Grades (A,B,C,D,E,F)	The mapping function f1 will largely depend on application requirements. Example: A (81-100); B (61-80); C (41-60); D (21-40); E (1-20)

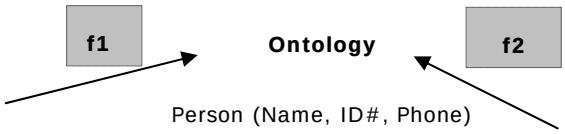
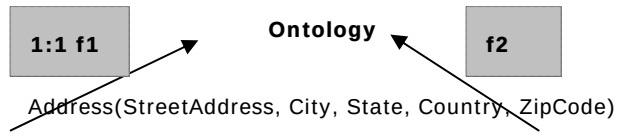
Schema Isomorphism conflict	Schema of similar elements have different number of attributes  **WS1** Person(Name, SSN, Home Phone, Work Phone) **WS2** Person(Name, SSN, Phone)	The mapping function f1 will largely depend on application requirements. f1 may be defined as 1. Home Phone (WS1) -> Phone(Ontology) or 2. Work Phone (WS2) -> Phone(Ontology) Note: While mapping in the direction of f1 can be well defined, $f1^{-1}$ can not.
Generalization conflict	Representation at different levels of generalization  **WS1** Grad-Student(ID, Name, Major) **WS2** Student(ID, Name, Major, Type)	While the Type information in the mapping function f2 can be ignored while mapping to the ontology instance, $f2^{-1}$ is not well defined and will depend on the value in the Type attribute of Student instance.
Aggregation conflict	Aggregation of source entities to a target entity  **WS1** Address(StreetAddress, City, State, Country, ZipCode) **WS2** Address(StrAdd1, StrAdd2, City, State, Country, ZipCode)	The mapping function f2 = concat (StrAdd1 , StrAdd2) -> StreetAddress can be defined whereas the mapping in the other direction $f2^{-1}$ is not precise.

Table1: Possible schematic / data conflicts between xml input/output messages

*****Although the mapping function is well defined in one direction, from the WSDL element to the Ontology concept, it is not well defined in the reverse direction. Although converting a 5 digit StudentID to a 9 digit StudentID is conceivable through use of a look up table, the transformation not a well defined function in itself.

Legend: We use WS1, WS2 to denote Web Services 1 and 2 and f1 and f2 to denote mapping functions from the WSDL elements to the ontology.

In addition to mapping representation languages, another area of focus has been the task of automating the process of generating mappings. Generating mapping expressions are essential in enabling semantic integration applications like query processing, data integration or exchange of xml messages between Web services. Semi-automating the process of elaborating matches has been discussed in [MHHYHFP01, MBR01, DR02].

Summary:

Although the loosely coupled nature of Web services has reduced a lot of heterogeneity between interoperating systems at the syntax level, issues of semantic and model/representational heterogeneity are even more complex than before and remain to be addressed adequately. Semantically annotating WSDL elements partly addresses this issue by dissolving the ambiguities in their schemas, structures and syntaxes, to aid in service discovery and composition. That however, does not suffice to achieve complete interoperability that is critical for service invocation or process execution. It is for these reasons that we recognize the value add that schema/data mapping brings to the Web service descriptions. This appendix has therefore been an attempt to point the reader to existing work in the areas of schema matching and schema/data mapping, recognize issues that need to be addressed in the context of Web services, and briefly characterize the space of solutions to this matching and mapping challenge. The technical note chose to use the point in this space that seemed most practical to us at this time.

References to Appendix D:

[BGHSS02]Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., and Stuckenschmidt, H., “C-OWL: Contextualizing Ontologies", Proceedings of the Second International Semantic Web Conference, 2003, pp.164—179.

[CDD01] Castano S, De AntonellisV, De Capitani diVemercati S (2001) Global viewing of heterogeneous data sources. IEEE Trans Data Knowl Eng 13(2):277–297

[CGL01] D. Calvanese, G. Giacomo, and M. Lenzerini. Ontology of integration and integration of ontologies. In *Description Logic Workshop (DL 2001)*, pages 10–19, 2001.

[CKSTD04] V. Christophides, I. Koffina, G. Serfiotis, V. Tannen, A. Deutsch, Integrating XML Data Sources using RDF/S Schemas: The ICS-FORTH Semantic Web Integration Middleware (SWIM), Dagstuhl Seminar (2004): Semantic Interoperability and Integration

[CM03] M. Crub´ezy and M. A. Musen. Ontologies in support of problem solving. In S. Staab and R. Studer, editors, Handbook on Ontologies, pages 321–342. Sringer, 2003.

[BM01a] J. Berlin, A. Motro: Autoplex, Automated Discovery of Content for Virtual Databases: CoopIS 2001,108-122

[BM01b] Jacob Berlin and Amihai Motro: Database Schema Matching Using Machine Learning with Feature Selection, November 2001, ISE-TR-01-06.

[DDH01] AnHai Doan, Pedro Domingos, Alon Y. Halevy, Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach: (2001) SIGMOD Conference

[DH04] Doan A, Halevy A., Semantic Integration Research in the Database Community: A Brief Survey: SIGMOD Record, 33(1):138-140, 2004. A related version appeared in AI Magazine, Spring 2004.

[DLDHD04] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, Domingos P, iMAP: Discovering Complex Semantic Matches between Database Schemas: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (pp. 383-394), 2004. Paris, France: ACM Press.

[DKB95] S.B. Davidson, A. Kosky, and P. Buneman, Semantics of Database Transformations: Semantics in Databases 1995: 55-91

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, Alon Y. Halevy, Learning to map between ontologies on the semantic web: WWW 2002: 662-673

[DMQ03] D. Dou, D. McDermott, and P. Qi, Ontology translation on the semantic web: In International Conference on Ontologies, Databases and Applications of Semantics, 2003.

[DR02] Hong Hai Do, Erhard Rahm, COMA - A System for Flexible Combination of Schema Matching Approaches: VLDB 2002: 610-621 (2002)

[EJX01] D. Embley, D. Jackman, L. Xu, Multifaceted Exploitation of Metadata for Attribute Match Discovery in Information Integration: WIIW 2001

[FB02] D. Fensel, C.Bussler : The Web Service Modeling Framework WSMF. In: Electronic Commerce Research and Applications, Vol. 1, Issue 2, Elsevier Science B.V., Summer 2002

[HG05] Farshad Hakimpour and Andreas Geppert: Resolution of Semantic Heterogeneity in Database Schema Integration Using Formal Ontologies, Information Technology and Management 2005

[KAM03] Anastasios Kementsietsidis, Marcelo Arenas, Renée J. Miller: Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. SIGMOD Conference 2003: 325-336

[KS03] Y.Kalfoglou, M.Schorlemmer, Ontology mapping: the state of the art: The Knowledge Engineering Review 18(1):1--31, January 2003

[LA86] Witold Litwin, Abdelaziz Abdellatif: Multi-database Interoperability. IEEE Computer 19(12): 10-18 (1986).

[M04] S. Melnik, Generic Model Management: Concepts and Algorithms, Ph.D. Dissertation: University of Leipzig, Springer LNCS 2967, 2004

[M05] Sergey Melnik, Model Management: First Steps and Beyond:German Database Conference (BTW) 2005 (invited paper)

[MBDH02] Jayant Madhavan, Philip A. Bernstein, Pedro Domingos, and Alon Halevy, Representing and Reasoning about Mappings between Domain Models, The Eighteenth National Conference on Artificial Intelligence (AAAI'2002), Edmonton, Canada

[MBR01] Madhavan, Jayant;Bernstein, Philip A.;Rahm, Erhard, Generic Schema Matching with Cupid: Proc. 27th Int. Conf. on Very Large Data Bases (VLDB 2001)

[MGR02] Melnik, S., Garcia-Molina, H., Rahm, E. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching: In: Proc. 18th ICDE, San Jose, CA (2002)

[MHH00] Miller RJ, Haas L, Hern'andez MA, Schema mapping as query discovery: In Proc 26th International Conference On Very Large Data Bases (2000)

[MHHYHFP01] Renee J. Miller, Mauricio A. Hernandez, Laura M. Haas, Lingling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa, The Clio project: managing heterogeneity: SIGMOD Rec., 30(1):78--83, 2001.

[MMSV02] A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA - a mapping framework for distributed ontologies. In 13th European Conference on Knowledge Engineering and Knowledge Management EKAW, Madrid, Spain, 2002.

[MWJ99] Prasenjit Mitra, Gio Wiederhold, Jan Jannink, Semi-automatic Integration of Knowledge Sources: Proceedings of Fusion '99, Sunnyvale, USA, July 1999, Pages:572-581.

[MZ98a] Milo and Zohar, TranScm System:1998

[MZ98b] T. Milo, S. Zohar, Using Schema Matching to Simplify Heterogeneous Data Translation: VLDB 98, August 1998.

[NHTHM02] Felix Naumann, Ching-Tien Ho, Xuqing Tian, Laura Haas, and Nimrod Megiddo, Attribute classification using feature analysis: In Proc. of the Int'l Conf. on Data Eng., San Jose, CA, 2002.

[N04] Natalya Fridman Noy, Semantic Integration: A Survey Of Ontology-Based Approaches: SIGMOD Record 33(4): 65-70 (2004)

[OTSV04] Nicole Oldham, Christopher Thomas, Amit P. Sheth, Kunal Verma, METEOR-S Web Service Annotation Framework with Machine Learning Classification: SWSWPC 2004: 137-146

[OWL-S] Web Ontology Language for Web Services, http://www.daml.org/services

[POSV04] Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, Kunal Verma, Meteor-s web service annotation framework: WWW 2004: 553-562

[PSU98] Palopoli L, Sacca D, Ursino D (1998) Semi-automatic, semanticdisc overy of properties from database schemas. In: Proc Int. Database Engineering and Applications Symp. (IDEAS), IEEE Comput, pp. 244–253

[RB01] Rahm, E., and P. A. Bernstein, A Survey of Approaches to Automatic Schema Matching: VLDB Journal 10, 4 (Dec. 2001),

[RDM04] E. Rahm, H. Do, S. Massmann, Matching large XML schemas: SIGMOD Record 2004

[S03] A. Sheth, Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration, Invited Talk, WWW 2003 Workshop on E-Services and the Semantic Web, Budapest, Hungary, May 20, 2003. Abstract Slides: pdf powerpoint-show htm

[S04] A. Sheth, Early work in database research on schema mapping/merging/ transformation, semantic heterogeneity, and use of ontology and description logics for schematic and semantic integration, discussion at the Dagstuhl Seminar on Semantic Interoperability and Integration, September 2004, http://www.dagstuhl.de/files/Proceedings/04/04391/04391.SWM2.Other.htm

[S99] A. Sheth, Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics: in Interoperating Geographic Information Systems. M. F. Goodchild, M. J. Egenhofer, R. Fegeas, and C. A. Kottman (eds.), Kluwer, Academic Publishers, 1999, pp. 5-30.

[SK93] A. Sheth and V. Kashyap, So Far (Schematically) yet So Close (Semantically): Proceedings of the DS-5 Conference on Semantics of Interoperable Database Systems, Lorne, Australia, Elsvier Publishers, November 1992; Elsevier North Holland, Amsterdam 1993.

[SM01] G. Stumme and A. M¨adche. FCA-Merge: Bottom-up merging of ontologies. In 7th Intl. Conf. on Artificial Intelligence (IJCAI '01), pages 225–230, Seattle, WA,2001.

[SVSM03] Kaarthik Sivashanmugam, Kunal Verma, Amit P. Sheth, John A. Miller, Adding Semantics to Web Services Standards: ICWS 2003: 395-401

[WCL00] W.Li, C. Clifton, S.Y. Liu, Database Integration Using Neural Networks: Implementation and Experiences: Knowledge and Information Systems 2: 1, 2000

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